UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Aldelir Fernando Luiz

PROTOCOLOS TOLERANTES A FALTAS BIZANTINAS PARA TRANSAÇÕES EM BANCOS DE DADOS

Florianópolis

2015

Aldelir Fernando Luiz

PROTOCOLOS TOLERANTES A FALTAS BIZANTINAS PARA TRANSAÇÕES EM BANCOS DE DADOS

Tese submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas como parte dos requisitos para a obtenção do Grau de Doutor em Engenharia de Automação e Sistemas.
Orientador: Prof. Lau Cheuk Lung, Dr.

Florianópolis

2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Luiz, Aldelir Fernando
Protocolos Tolerantes a Faltas Bizantinas para Transações em Bancos de Dados / Aldelir Fernando Luiz ; orientador, Lau Cheuk Lung - Florianópolis, SC, 2015.
344 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências

1. Engenharia de Automação e Sistemas. 2. Sistemas Computacionais. 3. Sistemas Distribuídos. 4. Tolerância a Faltas Bizantinas. 5. Transações. I. Lung, Lau Cheuk. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

PROTOCOLOS TOLERANTES A FALTAS BIZANTINAS PARA TRANSAÇÕES EM BANCOS DE DADOS

Aldelir Fernando Luiz

Tese submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Doutor em Engenharia de Automação e Sistemas.

Prof. Lau Cheuk Lung, Dr.
Orientador

Prof. Rômulo Silva de Oliveira, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Automação e Sistemas

Banca Examinadora:

Prof. Lau Cheuk Lung, Dr.
Presidente

Prof. Angelo Roncalli Alencar Brayner, Dr.

Prof. Carlos Barros Montez, Dr.

Prof.ª Fabíola Gonçalves Pereira Greve, Dra.

Prof. Laércio Lima Pilla, Dr.

Prof. Miguel Nuno Dias Alves Pupo Correia, Dr.

À minha esposa Lilian, pelo amor, incentivo e apoio incondicionais...

Aos meus pais Juvenal e Vera, pelos valores, ensinamentos e exemplo de vida...

À todos os meus familiares...

AGRADECIMENTOS

Antes de tudo quero agradecer à Deus, pelas bênçãos recebidas, as quais me permitiram realizar com êxito todas as etapas deste trabalho; obrigado por sua misericórdia imensa para comigo.

À minha esposa, Lilian, pelo constante apoio e incentivo durante esta difícil jornada; e sobretudo, pelas motivações e injeções de ânimo recebidas, nos momentos em que eu duvidava ter condições de seguir adiante.

Aos meus pais, Juvenal e Vera, por acreditarem em mim desde sempre, e pelas lições que me ensinaram ao longo de toda minha vida (e ainda ensinam!); tudo o que sou devo a eles.

Aos meus irmãos, Andre, Adriana e Fabiana, pelo incentivo que tenho recebido ao longo de todos estes anos, mas principalmente pela compreensão quanto às muitas ausências nos eventos familiares, neste conturbado período.

Gostaria de agradecer a todos os meus familiares, pelas boas energias e incentivo; mas em especial ao meu cunhado Ricardo (vulgo índio velho), à tia Mary e ao tio Castanheira, cujo apoio logístico e hospedagem foram imprescindíveis para que eu lograsse êxito neste trabalho.

Agradeço de coração ao prof. Lau, meu orientador, não apenas pelos vários anos de parceria acadêmica, mas também pela verdadeira relação de amizade construída. Não obstante, o agradeço pela confiança depositada, principalmente devido à minha condição durante a maior parte do doutoramento; e pelo apoio recebido em todas as etapas desta jornada, que foi determinante para o meu crescimento como pesquisador.

Um agradecimento mais do que especial ao prof. Miguel Correia, que embora não tenha sido formalmente um co-orientador, dotou de muita presteza e dedicação durante a concepção deste trabalho, em sua totalidade. Pelos valiosos comentários, considerações e ideias, bem como pela ótima recepção em Portugal, o meu mais profundo agradecimento.

Não posso deixar de agradecer ao prof. Joni Fraga, pela sugestão que culminou no tema desta tese. É uma grande pessoa, com a qual tive o prazer de conviver e discutir as ideias iniciais deste trabalho.

Agradeço imensamente e de coração aos amigos Valdir Stumm Junior, Hylson Vescovi Netto, Jeovani Schmitt e Maria Conceição Coppete (Cuca). Ao Valdir, pelas valiosas e sucessivas discussões durante a concepção dos trabalhos, e por sua pronta disposição em ler e su-

gerir inúmeras correções neste documento. Ao Hylson, pelo companheirismo, pelas inúmeras discussões que não se limitaram apenas ao campo técnico, mas também sobre experiências de vida; pelas muitas caronas solidárias e pelas diversas sugestões recebidas quanto à redação final deste documento. Ao Jeovani, que além do companheirismo, não mediu esforços em prover orientações quanto aos procedimentos estatísticos adequados para a verificação dos resultados desta tese. À querida Cuca, pelo otimismo, incentivo e pelas sábias palavras sempre por ela proferidas.

Também quero agradecer a todo o corpo que constitui o PPGEAS (docentes, servidores, etc.), pelo excelente nível de qualidade e comprometimento. Em especial ao prof. Eugenio Castelan, pela bolsa oferecida no período inicial do curso, a qual não pude aceitar devido às circunstâncias da época. Ao Enio e aos profs. Rômulo e Jomi; por sua acessibilidade e prontidão nos momentos em que precisei de vossa ajuda.

Aos professores que compuseram a banca examinadora, nomeadamente Angelo Brayner, Fabíola Greve, Carlos Montez, Laércio Lima Pilla, cujas críticas, sugestões e contribuições foram essenciais para aprimorar esta tese. Em especial ao prof. Angelo, relator da tese, pela seriedade e rigor manifestados em seu parecer – seus apontamentos engrandeceram significativamente este trabalho.

Eu seria uma pessoa muito ingrata, se deixasse de agradecer aos colegas Nilson Telles Jr., Wagner Xavier, Nilson Wolfgramm e Alexandre Krammel – colegas da TOTVS. Os agradeço por terem viabilizado o inicio desta difícil jornada, por meio da flexibilização de meus horários de trabalho, no período em que trabalhei convosco.

Ao Instituto Federal Catarinense, por ter provido o afastamento integral para capacitação, num período que foi determinante para a conclusão deste trabalho.

Aos colegas do LaPeSD, com os quais tive o prazer de conviver durante esta jornada, são eles: Tulio, Dettoni, Marcelo, Leandro, Paulo TXT, Paulo Pinho, Jim, Eliza, Ramon, André, Sérgio, ...

Enfim, meu eterno agradecimento à todos que contribuíram para a realização deste trabalho; que Deus os abençoe!

Se você acreditar que uma coisa é impossível, você a tornará impossível.

Bruce Lee

RESUMO

No âmbito de sistemas computacionais, a noção de transações constitui um dos elementos mais fundamentais para a especificação e implementação de aplicações com requisitos de confiabilidade e consistência, quanto à manipulação de dados. Ao longo dos anos, os sistemas de gerenciamento de banco de dados relacionais (SGBDR) têm sido considerados como componentes chave para o processamento de transações em sistemas computacionais; e, embora algumas alternativas aos SGBDRs tenham surgido nos últimos anos, há perspectivas de que um número significativo de sistemas computacionais permaneçam a utilizar os SGBDRs nos anos vindouros. Neste sentido, é imperioso que requisitos como confiabilidade, disponibilidade, desempenho, tolerância a faltas e consistência, sejam mantidos no SGBDR, com vista para o correto processamento de transações. Tais atributos podem ser obtidos por meio de replicação, sendo que a literatura é vasta em termos de soluções que visam a disponibilidade dos dados a despeito de faltas por parada intermitentes ou permanentes. Todavia, faltas oriundas da corrupção de dados em disco ou em memória RAM devido a efeitos físicos, ou decorrente de *bugs* no SGBDR, não afetam a disponibilidade dos dados, mas sim a integridade e a consistência destes. Estas faltas, que são caracterizadas como bizantinas, historicamente têm sido encontradas em SGBDRs, e elas afetam o processamento de transações comprometendo não apenas a exatidão, mas também o estado do banco de dados. A literatura dispõe de poucas soluções para lidar com faltas bizantinas em SGBDRs, onde algumas são baseadas em hipóteses difíceis de serem substanciadas na prática; ou em modelos de consistência mais relaxados, que podem causar problemas de integridade, consistência ou até mesmo corrupção de dados. Isto posto, elas não atendem plenamente todos os tipos de aplicações. Neste ensejo, esta tese versa sobre problemas relacionados ao processamento e a terminação de transações em SGBDRs e sistemas distribuídos, em ambientes com sujeição a faltas bizantinas. Para isso, esta tese apresenta duas grandes contribuições no âmbito de transações em SGBDRs. A primeira consiste num protocolo que permite o processamento e terminação de transações, a despeito de faltas bizantinas nas réplicas de bancos de dados – o primeiro da literatura a explorar a semântica de consistência mais forte de transações –, a partir de um protocolo de replicação tolerante a faltas bizantinas totalmente distribuído. A segunda investiga o problema da terminação

de transações distribuídas em ambientes sujeitos a faltas bizantinas – um problema sem solução pela literatura –, que é conhecido como Validação Atômica Não-Bloqueante (NBAC) e visa assegurar uma decisão uniforme para os participantes da transação acerca da validação ou anulação das operações executadas num ambiente distribuído de banco de dados. A contribuição para este problema se baseia na investigação dos aspectos práticos necessários para resolvê-lo, onde é introduzido o conceito de Gerenciador Colaborativo de Transações em substituição ao Gerenciador de Transações, tradicionalmente empregado como agente num protocolo NBAC. A solução proposta para resolver o NBAC com faltas bizantinas baseia-se num conceito novo, que adota a tecnologia de virtualização como suporte para a especificação de uma arquitetura de sistema distribuído que permite circunscrever o problema. As soluções propostas foram comparadas de maneira analítica com soluções encontradas na literatura, bem como através de ensaios experimentais, a fim de comprovar a viabilidade das mesmas.

Palavras-chave: processamento de transações, bancos de dados relacionais, tolerância a faltas bizantinas, problemas de acordo em sistemas distribuídos.

ABSTRACT

Within computer systems, the concept of transaction is one of the most fundamental elements for the specification and implementation of applications with reliability and consistency requirements concerning handling of data. Over the years, relational database management systems (RDBMS) have been considered key components for transaction processing in computer systems; and although some alternatives to RDBMSs have emerged in recent years, there are prospects that a significant number of computer systems will continute to use RDBMSs in coming years. In this sense, the need to provide reliability, availability, performance, fault tolerance and consistency, regarding transaction processing in RDBMS is imminent. Such attributes may be obtained through database replication. The literature is extensive in terms of solutions for data availability regardless of crash faults (e.g. intermittent or permanent). However, faults arising from the disk data corruption or RAM data corruption due to physical effects, or due to bugs in the RDBMS, do not affect the availability of data, though they affect their integrity and consistency. These faults, which are known as Byzantine, have historically been found in RDBMSs, and they affect transaction processing undermining not only the accuracy but also the database state. The literature offers few solutions to deal with Byzantine faults in RDBMSs, where some are based on difficult cases to be used in practice; or more relaxed consistency models, which can cause integrity, consistency or even data corruption problems. So, they are not addressed to all types of applications. As it is, this thesis deals with problems related to transaction processing and transaction termination on RDBMS and distributed systems, in environments subject to Byzantine fault. To this end, this thesis presents two major contributions to the transaction level in RDBMSs. The first is a protocol that allows the transaction processing and transaction, despite Byzantine faults in databases replicas – the first in literature that explores the strongest consistency semantics of transactions (e.g. serializability) –, by means of a fully distributed Byzantine fault tolerant database replication protocol. The second investigates the agreement problem related to transaction termination in distributed systems, also in environments subject to Byzantine fault – an unsolved problem in the literature. The Non–Blocking Atomic Commitment (NBAC), as it known, aims to ensure a uniform decision for the transaction participants about the

operations performed in a distributed database environment, that is, commit or abort them. The contribution to this problem is based on the investigation of practical and necessary conditions, to solve it. So, this thesis introduces the Collaborative Transaction Manager to replace the Transaction Manager, traditionally used as an agent on a NBAC protocol. The solution proposed to solve the NBAC with Byzantine fault is based on a new concept, adopting virtualization technology as a support for the specification of a distributed system architecture which makes the problem feasible. The proposed solutions were compared analytically with solutions found in the literature as well as through experimental tests in order to prove their feasibility.

Keywords: transaction processing, relational databases, byzantine fault tolerance, agreement problems in distributed systems.

LISTA DE FIGURAS

Figura 1 Arquitetura típica de um SGBD. 45
Figura 2 Máquina de estados finita para uma transação. 49
Figura 3 Exemplo de execução/escalonamento de transações. 52
Figura 4 Relacionamento entre as noções de serialização. 54
Figura 5 O fenômeno *dirty write*. 72
Figura 6 O fenômeno *dirty read*. 72
Figura 7 O fenômeno *non-repeatable read*. 73
Figura 8 O fenômeno *lost update*. 74
Figura 9 A anomalia/fenômeno *phantom read*. 75
Figura 10 A anomalia *read skew*. 75
Figura 11 A anomalia *write skew*. 76
Figura 12 A anomalia *read-only*. 77
Figura 13 Cenário com situação de impasse. 78
Figura 14 Exemplo de um sistema distribuído. 83
Figura 15 Monitor de máquinas virtuais Tipo I. 102
Figura 16 Monitor de máquinas virtuais Tipo II. 103
Figura 17 Impossibilidade do acordo bizantino com $n < 3f + 1$ 108
Figura 18 Acordo bizantino com $n \geq 3f + 1$. 109
Figura 19 Interseção de quóruns de leitura e de escrita em RME. . 113
Figura 20 O protocolo PBFT em execução normal. 115
Figura 21 O protocolo Zyzzyva em execução livre de falhas. 117
Figura 22 O *pipeline* de execução do protocolo Chain. 119
Figura 23 Cenários de execução do protocolo Q/U. 120
Figura 24 Execução de escritas no protocolo HQ. 122
Figura 25 Dinâmica de funcionamento do protocolo DBSM. 127
Figura 26 Arquitetura de *middleware* do Byzantium. 134
Figura 27 Execução do protocolo 3PC. 141
Figura 28 O protocolo ACP–UTRB. 144
Figura 29 Diagrama de execução do protocolo DNB–AC. 146
Figura 30 Padrão de comunicação empregado no protocolo MD3PC. 148
Figura 31 Passos de comunicação e fase do protocolo NB–2PC. 150
Figura 32 Execução de transações a partir de uma RME BFT. 162

Figura 33 Relação de precedência quanto à validação da transação. 165
Figura 34 Diagrama de execução do protocolo em três fases. 172
Figura 35 Máquina de estados que denota as etapas da transação. 176
Figura 36 Cenário de conflito na execução de transações. 194
Figura 37 Fenômeno induzido pela ação de clientes bizantinos. . . . 200
Figura 38 Arquitetura básica do protótipo de *middleware*. 213
Figura 39 Elementos que compõem a arquitetura de *middleware*. . 216
Figura 40 Modelo típico de cenário de implementação de um protocolo NBAC. 239
Figura 41 Modelo de cenário para implementação da solução NBWAC proposta. 241
Figura 42 Modelo de cenário para implementação da solução NBWAC pelo conceito de CTM. 242
Figura 43 Detalhamento do modelo de arquitetura proposta para resolver o NBWAC com faltas bizantinas. 244
Figura 44 Colaboração entre os TMs de um mesmo sítio (CTM). . 245
Figura 45 Diagrama de execução e fases do HB-NBWAC. 250
Figura 46 Diagrama ER para o *schema* do TPC-C. 282
Figura 47 Desempenho verificado para a carga padrão do TPC-C. 286
Figura 48 Latência observada para a carga padrão do TPC-C. 289
Figura 49 Transações processadas vs. validadas no TPC-C. 290
Figura 50 Desempenho para a carga padrão do TPC-C com faltas. 295
Figura 51 Latência para a carga padrão do TPC-C com faltas. . . . 296
Figura 52 Modelo adotado para a implementação da arquitetura. . 301
Figura 53 Latência observada para operações na *PostBox*. 305
Figura 54 Latência observada para operações de assinatura. 306
Figura 55 Capacidade do protocolo em termos de vazão. 308

LISTA DE QUADROS

Quadro 1 Compatibilidade entre os tipos de bloqueio. 59
Quadro 2 Compatibilidade entre os tipos de bloqueio no 2V2PL. 67
Quadro 3 Compatibilidade entre os tipos de bloqueio no MV2PL. 68
Quadro 4 Níveis de isolamento baseados no padrão ANSI SQL. . . 80
Quadro 5 Níveis de isolamento conforme Berenson *et al.* (1995). 81

LISTA DE TABELAS

Tabela 1 *Bugs*/faltas observado(a)s em SGDBs comerciais. 35
Tabela 2 Limiar de faltas toleradas em um protocolo de replicação. 112
Tabela 3 Comparação analítica do estado-da-arte em BFT 124
Tabela 4 Análise dos protocolos de replicação de bancos de dados. 138
Tabela 5 Eficiência dos protocolos de validação de transações. . . . 156
Tabela 6 Avaliação acerca do processamento da transação. 278
Tabela 7 Anulações ocorridas pela execução do TPC-C. 291
Tabela 8 Custo verificado em protocolos de validação atômica. . . 298

LISTA DE ALGORITMOS

1 Execução otimista da transação – cliente c_i 180
2 Processamento da transação – réplica r_j 185
3 Terminação da transação – réplica r_j 190
4 Certificação e validação da transação – réplica r_j 192
5 Especificações para o protocolo – sítio S_i – processo p_j . . 253
6 Tarefa principal do protocolo – sítio S_i – processo p_j. . . . 256
7 Etapa de decisão do protocolo – sítio S_i – processo p_j . . . 257

LISTA DE ABREVIATURAS E SIGLAS

2PC	*Two-Phase Commit*
2PL	*Two-Phase Locking*
3PC	*Three-Phase Locking*
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
BFT	*Byzantine Fault Tolerance*
CTM	*Collaborative Transaction Manager*
DBSM	*DataBase State Machine Approach*
DM	*Data Manager*
ECC	*Error Checking and Correction*
ERP	*Enterprise Resource Planning*
FIFO	*First In First Out*
JDBC	*Java DataBase Connectivity*
NBAC	*Non-Blocking Atomic Commitment*
NBWAC	*Non-Blocking Weak Atomic Commitment*
NFS	*Network File System*
NIST	*National Institute of Standards and Technology*
MAC	*Message Authentication Code*
RM	*Resource Manager*
RME	Replicação de Máquina de Estados
SBD	Sistema de Banco de Dados
SGBD	Sistema de Gerência de Banco de Dados
SGBDR	Sistema de Gerência de Banco de Dados Relacional
SI	*Snapshot Isolation*
SMR	*State Machine Replication*
SQL	*Structured Query Language*
TM	*Transaction Manager*
VMM	*Virtual Machine Monitor*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 31
1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO 32
1.2 OBJETIVOS 37
1.3 METODOLOGIA 38
1.4 CONTRIBUIÇÕES 39
1.5 ORGANIZAÇÃO DO DOCUMENTO 40
2 FUNDAMENTOS 43
2.1 CONCEITOS EM BANCOS DE DADOS E TRANSAÇÕES 43
2.1.1 Bancos de Dados 43
2.1.2 Transações 47
2.1.3 Serialização 51
2.1.3.1 Serialização em Bancos de Dados Não-Replicados 58
2.1.3.2 Serialização em Bancos de Dados Replicados 69
2.1.4 Problemas Relacionados à Concorrência 70
2.1.4.1 Fenômenos e Anomalias de Concorrência 71
2.1.4.2 Impasses (*Deadlocks*) 78
2.1.4.3 Inanição (*Starvation*) 79
2.1.5 Isolamento de Transações 79
2.2 CONCEITOS EM SISTEMAS DISTRIBUÍDOS 82
2.2.1 Caracterização de Sistemas Distribuídos 82
2.2.2 Ambiente de Computação Distribuída 84
2.2.2.1 Modelo de Sistema 84
2.2.2.2 Processos 85
2.2.2.3 Canais de Comunicação 85
2.2.2.4 Modelos de Falhas 87
2.2.3 Acordo em Sistemas Distribuídos 89
2.2.3.1 Consenso 89
2.2.3.2 Validação Atômica Não-Bloqueante 92
2.2.3.3 Difusão de Mensagens 94
2.2.3.4 Replicação de Máquina de Estados 97
2.3 TECNOLOGIA DE VIRTUALIZAÇÃO 99
2.4 CONSIDERAÇÕES DO CAPÍTULO 103
3 CONTEXTUALIZAÇÃO 105
3.1 CONCEITOS EM TOLERÂNCIA A FALTAS BIZANTINAS 105
3.1.1 Acordo Bizantino 106
3.1.2 Tolerância a Faltas Bizantinas através de Replicação 109
3.1.2.1 Limites Empregados na Replicação para BFT 111

3.1.2.2 Estado-da-arte em Protocolos de Replicação para BFT .. 113
3.1.3 Comparação dos Protocolos de Replicação BFT 123
3.2 TRABALHOS RELACIONADOS 125
3.2.1 Soluções Baseadas na Replicação de Bancos de Dados 125
3.2.1.1 Replicação de Máquina de Estados para Bancos de Dados 126
3.2.1.2 Aplicação do Acordo Bizantino em Bancos de Dados 128
3.2.1.3 Uso de Diversidade como Suporte à Replicação de Bancos de Dados 130
3.2.1.4 Escalonamento Baseado em Barreira de Validação para Replicação Tolerante a Falta Bizantinas de Bancos de Dados 131
3.2.1.5 Replicação de Bancos de Dados Tolerante a Faltas Bizantinas com Suporte ao *Snapshot Isolation* 133
3.2.1.6 Tolerância a Faltas Bizantinas no Processamento de Transações em Sistemas Baseados em Armazenamento Chave-Valor 136
3.2.1.7 Avaliação Analítica dos Protocolos BFT para Replicação de Bancos de Dados 138
3.2.2 Validação Atômica e Distribuída de Transações..... 139
3.2.2.1 Protocolos de Validação Atômica Não-Bloqueante Tolerantes a Faltas por Parada 140
3.2.2.2 Protocolos de Validação Distribuída Tolerantes a Faltas Bizantinas 151
3.2.2.3 Eficiência dos Protocolos de Validação Atômica e Distribuída 155
3.3 CONSIDERAÇÕES DO CAPÍTULO 157
4 PROTOCOLO TOLERANTE A FALTAS BIZANTINAS PARA O PROCESSAMENTO DE TRANSAÇÕES EM BANCOS DE DADOS RELACIONAIS.... 159
4.1 INTRODUÇÃO 159
4.2 DEFINIÇÕES PRELIMINARES 163
4.2.1 Predicados, Bloqueios, Conflitos e Precedência 163
4.2.2 Modelo de Sistema 166
4.3 VISÃO GERAL DO PROTOCOLO 169
4.4 DINÂMICA DE FUNCIONAMENTO DO PROTOCOLO .. 171
4.5 ESPECIFICAÇÃO E DETALHES DO PROTOCOLO 175
4.5.1 Declaração de Transações 175
4.5.2 Princípio de Funcionamento 176
4.5.3 Base Algorítmica 177
4.5.3.1 Início e Execução da Transação 178
4.5.3.2 Processamento da Transação 184

4.5.3.3 Terminação, Certificação e Validação da Transação 187
4.6 DISCUSSÃO 197
4.7 CORREÇÃO DOS ALGORITMOS 202
4.8 ESPECIFICAÇÃO DE *MIDDLEWARE* 212
4.8.1 Aspectos de Implementação 212
4.8.2 Componentes Arquiteturais do *Middleware* 215
4.8.2.1 Gerenciador de Conexão 215
4.8.2.2 Gerenciador de Requisição 217
4.8.2.3 Pré-processador de Requisição 217
4.8.2.4 Gerenciador de Transações 218
4.8.2.5 Módulo de Certificação 219
4.8.2.6 Gerenciador de Bloqueios 220
4.8.2.7 Escalonador de Requisição 220
4.8.2.8 Gerenciador de Autenticação 220
4.9 CONSIDERAÇÕES DO CAPÍTULO 221
5 VALIDAÇÃO ATÔMICA NÃO-BLOQUEANTE COM FALTAS BIZANTINAS 223
5.1 INTRODUÇÃO 223
5.2 CARACTERIZAÇÃO DO PROBLEMA 226
5.3 UM BREVE PANORAMA SOBRE SISTEMAS HÍBRIDOS 228
5.3.1 Definição de Ambiente Híbrido 228
5.3.2 Hibridização através de Componentes Confiáveis 229
5.3.3 Hibridização através de Virtualização 230
5.3.4 Outras Abordagens para Hibridização 232
5.4 CONCEITOS PRELIMINARES 233
5.4.1 Tecnologia de Virtualização 233
5.4.2 Assinaturas de Limiar 234
5.4.3 Detecção de Falhas 236
5.4.4 Memória Compartilhada 237
5.5 ASPECTOS GERAIS DA SOLUÇÃO PROPOSTA 239
5.6 ESPECIFICAÇÃO DO PROTOCOLO 246
5.6.1 Modelo de Sistema 247
5.6.2 Princípio de Funcionamento do HB-NBWAC 249
5.6.3 Base Algorítmica do Protocolo HB-NBWAC 251
5.7 CORREÇÃO DO PROTOCOLO 266
5.8 CONSIDERAÇÕES DO CAPÍTULO 274
6 AVALIAÇÃO DAS SOLUÇÕES PROPOSTAS 277
6.1 AVALIAÇÃO DO PROTOCOLO DE REPLICAÇÃO DE BANCOS DE DADOS TOLERANTE A FALTAS BIZANTINAS 277
6.1.1 Avaliação Analítica 277

6.1.2 Avaliação Experimental 280
6.1.2.1 Metodologia 281
6.1.2.2 Resultados Obtidos 283
6.2 AVALIAÇÃO DO PROTOCOLO DE VALIDAÇÃO ATÔMICA FRACA NÃO-BLOQUEANTE 297
6.2.1 Avaliação Analítica 297
6.2.2 Avaliação Experimental 301
6.3 CONSIDERAÇÕES DO CAPÍTULO 309
7 CONCLUSÃO 311
7.1 REVISÃO DOS OBJETIVOS E REALIZAÇÕES 312
7.2 RESULTADOS E PUBLICAÇÕES 316
7.3 PERSPECTIVAS DE TRABALHOS FUTUROS 318
REFERÊNCIAS 321

1 INTRODUÇÃO

Os avanços tecnológicos ocorridos nas últimas décadas nas áreas de computação e comunicação têm estimulado o surgimento de novas aplicações que, cada vez mais, demandam por sistemas de bancos de dados para prover confiabilidade no que tange o armazenamento de dados. Sistemas de informações colaborativas, de controle automático, de transações financeiras e bancárias, serviços de comércio eletrônico são alguns exemplos de uma ampla gama de aplicações avançadas, que utilizam sistemas de bancos de dados como suporte subjacente para o armazenamento de dados. O acesso ao sistema de bancos por parte destas aplicação se dá em conformidade com a semântica de transações (GRAY, 1981), no intuito de preservar rigorosamente a consistência dos dados utilizados no decurso das tarefas por eles realizadas. As transações são usadas para representar operações executadas sobre um banco de dados por programas de aplicação.

A noção de transação consiste em um meio factível para preservar a consistência de dados, a despeito de falhas parciais na execução de operações, ou de acessos concorrentes aos mesmos dados (GRAY, 1981). No âmbito de sistemas computacionais, uma transação pode ser vista como um mecanismo bastante útil para especificar e implementar, de maneira confiável, diversos tipos de aplicações no tocante a manipulação de dados. Ao mesmo tempo, a abstração de transação suaviza a tarefa do projetista, ao ocultar toda a complexidade oriunda da concorrência, ou mesmo do gerenciamento de falhas. Para tanto, as transações fornecem uma interface de alto nível ao projetista da aplicação, que abstrai tanto os efeitos de concorrência (ou simultaneidade) entre diferentes atividades que ocorrem no sistema, como aqueles que podem ser causados por falhas (WEIKUM; VOSSEN, 2002).

Devido à sua maturidade, o modelo relacional de dados (CODD, 1970) tem sido considerado como um padrão *de facto*, no que concerne ao projeto e especificação de sistemas computacionais que requerem alguma necessidade de persistência quanto à manipulação de dados. Da mesma maneira, os bancos de dados relacionais consistem numa tecnologia extremamente madura e que tem sido empregada há décadas, no que diz respeito ao armazenamento de dados estruturados. A despeito do surgimento de abordagens alternativas aos bancos de dados relacionais para prover o armazenamento de dados no âmbito de sistemas informáticos, tais como bancos de dados baseados em objetos (HARRINGTON, 2000), bancos de dados XML (WILLIAMS et al., 2000) e

os NoSQLs (TIWARI, 2011), os bancos de dados relacionais ainda hoje podem ser considerados como a tecnologia predominante e promitente quanto ao armazenamento de dados em sistemas computacionais.

É importante salientar que, no cerne dos bancos de dados relacionais (BDR)[1], estão os sistemas de gerência de bancos de dados relacionais (SGBDR)[2]. Um sistema de gerência de banco de dados (SGBD) é empregado para prover o acesso uniforme aos objetos armazenados em um banco de dados, e também para assegurar sua consistência a partir do gerenciamento e execução de transações. Isto implica que os usuários e seus programas de aplicação realizam o acesso aos dados de um banco de dados, a partir da execução de transações no SGBD. Ao longo dos anos, os sistemas de gerência de bancos de dados relacionais têm se intensificado como uma espécie de *mind share* no âmbito do desenvolvimento de aplicações (STONEBRAKER et al., 2007), principalmente por serem encarados como o suporte mais usual e adequado para armazenar os dados manipulados por sistemas computacionais, das mais diversas áreas de negócio.

Embora alguns autores apontem para a existência de desvantagens quanto ao uso dos bancos de dados relacionais, e consequentemente dos SGBDRs para determinados tipos de aplicações (STONEBRAKER et al., 2007), espera-se que os bancos de dados relacionais ainda se mantenham no auge de utilização para os anos vindouros, principalmente por parte de sistemas legados.

No que segue, este capítulo apresenta uma breve contextualização da proposta a ser desenvolvida nesta tese, seguida pela elucidação dos objetivos, da metodologia utilizada, das contribuições realizadas, e, por fim, da organização do documento.

1.1 CONTEXTUALIZAÇÃO E MOTIVAÇÃO

Sistemas computacionais baseados no processamento de transações em sistemas de bancos de dados são aplicações inerentemente complexas. De um modo geral, estes sistemas tipicamente contam com a confiabilidade inerente à semântica transacional e com a disponibilidade dos dados armazenados no banco de dados. Por outro lado, a literatura nos mostra que independentemente do tipo de aplicação de processamento de transações e do ambiente no qual ela está sendo executada, sempre há alguma susceptibilidade a falhas (HAERDER; REUTER, 1983;

[1] bancos de dados denotam coleções de dados

[2] conjuntos de programas que gerenciam os bancos de dados

RAMAKRISHNAN; GEHRKE, 2003). Evidentemente que, como as aplicações num sistema de banco de dados acessam/manipulam os dados por meio de transações, elas tiram vantagens da semântica transacional para fins de recuperação do estado da aplicação, no caso de falhas (p. ex.: pela reversão das operações). Com isso, o único impacto sofrido pela aplicação é aquele decorrente da inatividade momentânea do banco de dados.

Isto posto, a tolerância a faltas também podem ser entendida como uma propriedade desejável para os sistemas de suporte aos bancos de dados, no caso, os SGBDs. A literatura também descreve que, em geral, as falhas mais comuns ocorridas no âmbito de um sistema de banco de dados são as falhas de transação, falhas de mídia e falhas de sistema (HAERDER; REUTER, 1983; BRAYNER, 1999; RAMAKRISHNAN; GEHRKE, 2003; ÖZSU; VALDURIEZ, 2011). Com exceção às falhas de transação, que são naturalmente recuperadas pela anulação desta, as falhas de mídia e de sistema conduzem o SGBD, e consequentemente as aplicações executadas sobre ele, a uma parada imediata – de maneira análoga a uma falta por parada ocorrida num sistema distribuído (i.e., *crash* (JALOTE, 1994)).

Embora as faltas por parada sejam, indiscutivelmente, as faltas mais frequentes ocorridas no âmbito de sistemas computacionais, tem-se verificado que faltas de natureza arbitrária são bastante comuns na atualidade, e principalmente uma realidade nestes sistemas (SCHROEDER; GIBSON, 2007; SCHROEDER; PINHEIRO; WEBER, 2009; NIGHTINGALE; DOUCEUR; ORGOVAN, 2011). De um modo geral, as faltas por parada apenas comprometem a execução de uma aplicação. Todavia, faltas oriundas da corrupção de dados em disco ou em memória RAM devido à efeitos físicos (SCHROEDER; PINHEIRO; WEBER, 2009; NIGHTINGALE; DOUCEUR; ORGOVAN, 2011) ou ainda na codificação do *software* devido a *bugs* (GASHI; POPOV; STRIGINI, 2007), são faltas que muitas vezes não causam uma parada na aplicação, tampouco comprometem a execução das mesmas. Estas faltas são caracterizadas como faltas arbitrárias/bizantinas (LAMPORT; SHOSTAK; PEASE, 1982) e na maioria das vezes não conduzem o sistema a uma parada imediata, mas sim uma corrupção do estado da aplicação. Intuitivamente, as faltas bizantinas podem comprometer não apenas a execução, mas também o estado da aplicação.

A literatura tem demonstrado que faltas bizantinas são uma realidade no contexto de sistemas computacionais (SCHROEDER; GIBSON, 2007; SCHROEDER; PINHEIRO; WEBER, 2009; NIGHTINGALE; DOUCEUR; ORGOVAN, 2011). O que corrobora para tal afirmação são os resultados

obtidos por alguns estudos realizados, os quais evidenciaram que, por ano, pelos menos 8% dos módulos de memória DRAM dos servidores em *data centers* da Google sofreram erros, sendo que tais erros não foram detectados/corrigidos/mascarados nem pelos códigos de correção de erros (ECC) presentes nos módulos de memória (SCHROEDER; PINHEIRO; WEBER, 2009). Alguns destes erros também são ocasionados por *bit flips* (i.e., inversão de bits), os quais são oriundos de erros causados em decorrência de interferências eletromagnéticas, variações de tensão, etc. Um fenômeno que também pode causar faltas bizantinas e é bastante comum na infra-estrutura de sistemas computacionais é aquele conhecido como raios cósmicos (do inglês, *cosmic ray*), que consiste numa espécie de perturbação gerada por um campo eletromagnético. Estas pesquisas também evidenciaram que CPUs e *chipsets* presentes em *hardware commodity* são alvos bastante frequentes de falhas (NIGHTINGALE; DOUCEUR; ORGOVAN, 2011).

É lícito salientar que, curiosamente, diversos *bugs* têm sido encontrados em SGBDs comerciais nos últimos tempos (GASHI; POPOV; STRIGINI, 2004, 2007), e estes por sua vez afetam a exatidão acerca do processamento de transações, bem como comprometem a execução das aplicações sobre sistemas de bancos de dados. Os dados apresentados na Tabela 2, adaptada de (GASHI; POPOV; STRIGINI, 2004) e (VANDIVER et al., 2007) corroboram para tal afirmação. Nesta tabela são caracterizados alguns dos efeitos causados pelo mau funcionamento de diversos SGBDs comerciais, cujos *bugs* observados manifestaram comportamentos arbitrários (i.e., faltas de natureza arbitrária). É digno de nota, que os dados ali reportados compreendem apenas ao período de 09/1998 à 11/2006. Estes dados evidenciam nitidamente que, na maioria das vezes, em vez de causar a parada do SGBD os *bugs* manifestam faltas bizantinas decorrentes da execução incorreta de operações em uma transação. No âmbito da Tabela 2, o símbolo ↑ significa que o valor observado está incluído no valor imediatamente anterior (i.e., da linha anterior daquela coluna), enquanto que o símbolo * expressa um valor desconhecido.

Uma consequência danosa ao sistema de processamento de transações, decorre do fato de que, se estas faltas não violarem a semântica de consistência do banco de dados – o que é plausível num ambiente sujeito a faltas bizantinas –, resultados incorretos podem ser declarados aos clientes e itens de dados corrompidos e/ou inválidos podem ser armazenados no banco de dados, se a transação vier a ser validada após a ocorrência da(s) falta(s). De outro modo, mesmo que um *bug* termine por causar uma parada abrupta no SGBD e nas aplicações, é

Tabela 1 – *Bugs*/faltas observado(a)s em SGDBs comerciais.

Bug/Falta Manifestada	SGBD Analisado					
	DB2	**Oracle**	**MySQL**	**InterBase**	**PGSQL**	**MSSQL**
Falhas Observadas	**251**	**50**	**124**	**47**	**52**	**39**
Parada do SGBD	120	21	60	7	11	5
Falta Arbitrária	131	29	64	40	41	34
Resultado(s) Incorreto(s)	81	24	63	27	34	27
Corrupção do BD	40	4	↑	*	*	*
Desempenho	0	1	0	3	0	6
Outro(a)s Comportamento(s)	10	0	1	10	7	1

difícil estimar se num estágio intermediário entre a parada e a manifestação da falta bizantina, as operações executadas pelas transações foram efetuadas de maneira correta.

Uma primeira elucidação acerca de faltas bizantinas em bancos de dados ocorreu há várias décadas (GARCIA-MOLINA; PITTELLI; DAVIDSON, 1984). Porém, a especificação das primeiras soluções capazes de lidar com este tipo de faltas em bancos de dados ocorreu apenas num passado recente (VANDIVER et al., 2007; GASHI; POPOV; STRIGINI, 2007; GARCIA; RODRIGUES; PREGUIÇA, 2011). Por outro lado, pesquisas envolvendo tolerância a faltas bizantinas em sistemas computacionais, é algo bem estabelecido no âmbito da comunidade de pesquisa em sistemas distribuídos (CASTRO; LISKOV, 1999; CORREIA et al., 2002; CORREIA; NEVES; VERISSIMO, 2004; ABD-EL-MALEK et al., 2005; COWLING et al., 2006; KOTLA et al., 2007; VERONESE et al., 2010), cujas soluções para o problema estão em torno do uso de replicação de *software* (GUERRAOUI; SCHIPER, 1997). Não obstante, é importante salientar que os sistemas distribuídos têm tido um papel fundamental na atualidade, como base para a implementação de uma gama de serviços e aplicações utilizadas no cenário real. Isto se deve ao fato de que muitos destes serviços e aplicações exigem garantias de disponibilidade, desempenho, corretude – o que pode ser satisfeito por meio do uso de técnicas inerentes ao projeto de sistemas distribuídos, tais como tolerância a faltas, segurança, etc.

Todavia, apesar da maturidade da tecnologia de replicação, a especificação de soluções tolerantes a faltas bizantinas baseadas na replicação de bancos de dados é, ainda, um problema desafiador, de modo que a literatura carece de discussões mais aprofundadas sobre o tema. O que corrobora para tal afirmação é que até o momento, a literatura dispõe de apenas três trabalhos que tratam especificamente de faltas bi-

zantinas em bancos de dados relacionais; o trabalho de Garcia-Molina *et al.* (1984, 1986), o HRDB (VANDIVER et al., 2007) e o Bizantium (PREGUIÇA et al., 2008; GARCIA; RODRIGUES; PREGUIÇA, 2011), dentre os quais, apenas o HRDB e o Bizantium são passíveis de uso em ambientes reais. Em face disso, pode-se afirmar que o tema em questão ainda pode ser visto com sendo incipiente/seminal.

Note que aspectos relacionados à confiabilidade no que tange o processamento de transações em bancos de dados, são intensificados principalmente em aplicações responsáveis por executar serviços/sistemas críticos. Como exemplo de tais aplicações, pode-se citar os sistemas de processamento de transações que lidam com processos industriais, controle de tráfego, informações financeiras, gestão de recursos de manufatura, etc. Nesta classe de aplicações, uma falha ocorrida no decurso de uma transação, pormenor que seja, pode comprometer todo o funcionamento do sistema. O caso é ainda mais agravado quando faltas bizantinas são permitidas, cujo comportamento arbitrário implica que a(s) fonte(s) de dados não seja(m) fidedigna(s), o que, portanto, pode não comprometer o sistema, mas violar a caracterização semântica da aplicação. Não obstante, a ocorrência de falhas no processamento de transações nestes tipos de aplicação, pode ocasionar danos de grandes proporções ao sistema, principalmente quando relacionados à questões vitais.

Neste sentido, como há a iminente necessidade de prover tolerância a faltas, e manter escalabilidade e desempenho no âmbito de um ambiente de banco de dados; para o provimento de confiabilidade, no que diz respeito ao processamento de transações, é justificada a realização do objeto de estudo desta tese. Em decorrência dos desafios encontrados na especificação e implementação de estratégias e protocolos para replicação de bancos de dados capazes de tolerar faltas bizantinas, o tema em lide tem despertado interesse em termos de investigação, pelas comunidades de sistemas distribuídos e de bancos de dados. Portanto, isso é o que se busca demonstrar ao longo desta tese.

E, finalmente, a considerar que os poucos trabalhos existentes que tratam do tema em questão, ou são baseadas premissas difíceis de serem substanciadas na prática, ou adotam critérios que não são adequados para todos os tipos de aplicações; em linhas gerais, esta tese visa investigar o desenvolvimento de soluções capazes de lidar com a sujeição a faltas bizantinas não apenas no processamento de transações, mas também na terminação de transações distribuídas.

1.2 OBJETIVOS

Em face aos aspectos relatados e da motivação apresentada acerca do problema, este trabalho tem como objetivo geral investigar o desenvolvimento de soluções capazes de lidar com a sujeição a faltas bizantinas no âmbito de transações em bancos de dados, de modo que seja possível realizar o processamento e terminação destas, de maneira confiável, a despeito da manifestação de faltas bizantinas. Para tanto, se pretende: (*i*) especificar um suporte para o processamento de transações baseado na replicação do banco de dados, que seja passível de implementação e integração à SGBDs reais e; (*ii*) especificar uma solução arquitetural e algorítmica para prover a confiabilidade na terminação de transações distribuídas.

Para alcançar este objetivo, alguns objetivos de âmbito específico foram determinados, sendo que estes são:

1. Especificar, projetar e formalizar um protocolo tolerante a faltas bizantinas para a replicação de bancos de dados relacionais que:

   (a) proveja um suporte tolerante a faltas bizantinas para a execução de transações pelas aplicações usuárias daquele banco de dados;

   (b) mantenha a corretude do ambiente, de modo que o sistema se porte adequadamente mesmo que réplicas venham a falhar de maneira bizantina;

   (c) forneça um modelo forte de consistência em conformidade com a semântica de serialização de transações, bem como a integridade dos dados no âmbito do ambiente replicado;

   (d) mantenha todas as réplicas não faltosas com estados lógicos equivalentes;

   (e) forneça às aplicações clientes uma visão de como se o sistema não fosse replicado;

   (f) proveja um nível aceitável de concorrência acerca do processamento de transações, a despeito da consistência requerida e da ocorrência de faltas.

2. Definir e propor uma arquitetura de *middleware* como suporte para a replicação não intrusiva e tolerante a faltas bizantinas de um ambiente de banco de dados relacional, que proveja o suporte

necessário para o processamento de transações em SGBDs *off-the-shelf* reais;

3. Investigar modelos e ambientes de sistemas distribuídos com características mais realistas, de modo a buscar as condições necessárias para resolver o problema da validação atômica não-bloqueante – a despeito da impossibilidade demonstrada acerca de sua resolução, em ambientes sujeitos a faltas bizantinas;

4. Formalizar uma arquitetura e um modelo de sistema distribuído, que seja capaz de prover as condições necessárias para especificar um protocolo capaz de tolerar faltas bizantinas no decurso da terminação de transações, através de uma solução baseada no problema de acordo de validação atômica não-bloqueante;

5. Avaliar todas as propostas desenvolvidas por meio de um conjunto de ensaios experimentais, realizados a partir da implementação dos respectivos protótipos;

6. Divulgar as propostas e respectivos resultados por meio de publicações em veículos de divulgação científica atinentes à área da pesquisa.

1.3 METODOLOGIA

Uma pesquisa pode ser vista como um conjunto de ações e proposições que visam obter uma solução para um problema, a partir de procedimentos sistemáticos. Neste sentido, pode-se dizer que um dos propósitos da metodologia é nortear uma pesquisa, no que diz respeito as formas a serem adotadas para se alcançar os resultados esperados (SILVA; MENEZES, 2005). No campo das ciências, uma pesquisa pode ser classificada a partir (SILVA; MENEZES, 2005): (*i*) **de sua natureza**; (*ii*) **da abordagem do problema**; (*iii*) **dos objetivos**, e; (*iv*) **dos procedimentos técnicos**.

E neste ensejo, a partir da classificação descrita por Silva e Menezes (2005), o trabalho desenvolvido nesta tese está enquadrado da seguinte forma.

Com Base na Natureza: aplicada, pois visa a proposição de soluções específicas para a aplicação prática na resolução de problemas, e

no caso, relacionados à confiabilidade em transações em bancos de dados relacionais.

Com Base na Abordagem do Problema: quantitativa, já que se utiliza de parâmetros estatísticos obtidos a partir de experimentos, para analisar e qualificar os modelos e soluções propostas.

Com Base nos Objetivos: exploratória e explicativa, pois visa o aprimoramento de conceitos, definições e ideias, a partir da verificação bibliográfica do estado-da-arte, e da análise de experimentos.

Com Base nos Procedimentos Técnicos: bibliográfica e experimental, pois é elaborada tomando como base um conjunto de materiais existentes, e os resultados são avaliados a partir de experimentos.

1.4 CONTRIBUIÇÕES

Esta tese visa propor soluções para o provimento de confiabilidade no processamento e na terminação de transações em bancos de dados relacionais, de modo que estas sejam capazes de tolerar faltas bizantinas no decurso das computações, e a despeito de comportamento(s) não previsto(s) na(s) especificação(ões) daquele(s) sistema(s). Neste sentido, esta tese apresenta duas novas grandes contribuições para a área de tolerância a faltas bizantinas em bancos de dados.

A primeira consiste em um protocolo para replicação de bancos de dados relacionais, que provê consistência forte para as aplicações e também a confiabilidade quanto ao processamento e terminação de transações a despeito de faltas bizantinas. Tal solução apresenta um custo satisfatório/moderado em termos de desempenho, em relação à robustez e aos benefícios providos pela mesma, para as aplicações. A segunda contribuição reside na proposição de uma arquitetura de sistema distribuído baseada numa abordagem que estende o modelo de falhas do ambiente, e permite restringir/limitar o poder dos processos que falham de maneira bizantina de exibir determinados comportamentos. Tal arquitetura é especificada a partir de uma nova abordagem baseada em tecnologia de virtualização, que se estabelece a partir de um ambiente híbrido sobre um modelo factível e realista, a fim de possibilitar a especificação de uma solução para a validação atômica não-bloqueante, em que alguns agentes envolvidos possam falhar de maneira bizantina.

A proposição das contribuições introduzidas por esta tese reque-

reram alguns trabalhos intermediários, os quais também resultaram em contribuições e, portanto, merecem ser elencadas:

(a) proposição de um protocolo para replicação tolerante a faltas bizantinas de banco de dados, cujo processamento de transações ocorre de acordo com o critério de corretude baseado em serialização;

(b) investigação e identificação de uma anomalia que afeta a progressão de transações num ambiente replicado de banco de dados relacional, decorrente da combinação do controle de concorrência baseado em bloqueios e de clientes que falham de maneira bizantina;

(c) especificação, proposição e implementação de uma arquitetura de middleware para replicação tolerante a faltas bizantinas de bancos de dados, sobre SGBDs *off-the-shelf*;

(d) investigação de um modelo de computação para a resolução do problema da validação atômica não-bloqueante (NBAC) com faltas bizantinas;

(e) proposição de uma arquitetura de sistema baseada numa abordagem nova e especificada a partir de um modelo híbrido de falhas, para projetar algoritmos tolerantes a faltas;

(f) especificação de uma nova abordagem para prover a terminação de transações distribuídas em conformidade com a semântica do NB-WAC, que tolera faltas bizantinas;

(g) desenvolvimento de um protocolo NBWAC tolerante a faltas bizantinas, baseado na arquitetura de sistema e abordagem introduzidas por esta tese.

1.5 ORGANIZAÇÃO DO DOCUMENTO

Este capítulo descreveu de maneira abrangente, o contexto no qual está inserido esta tese, a motivação para a realização do estudo e os objetivos da tese. No que segue, o restante deste documento está organizado da seguinte maneira:

Capítulo 2) apresenta uma visão geral acerca dos fundamentos que compõem o arcabouço teórico desta tese, nas perspectivas de sistemas

de bancos de dados, processamento de transações e sistemas de computação distribuída.

Capítulo 3) discorre sobre a contextualização do cenário no qual a tese está inserida, onde se apresentam alguns conceitos essenciais em tolerância a faltas bizantinas, e também uma revisão da literatura em termos de trabalhos correlacionados à proposta desenvolvida.

Capítulo 4) apresenta a proposta de um novo protocolo tolerante a faltas bizantinas para replicação de bancos de dados relacionais, cujo propósito é prover a confiabilidade no processamento de transações sobre um ambiente de banco de dados relacional. Neste capítulo é descrito o princípio de funcionamento do protocolo, o que ocorre através da especificação de um conjunto de algoritmos. Uma proposta de arquitetura de *middleware* também é apresentada, como um resultado parcial da implementação do protocolo proposto.

Capítulo 5) descreve a proposição de uma nova arquitetura de sistema distribuído baseada numa perspectiva de hibridização do ambiente/modelo de sistema. Esta arquitetura é especificada com o propósito de circunscrever algumas condições que culminam na impossibilidade de resolução do problema da validação atômica não-bloquente para a terminação de transações, com sujeição a faltas bizantinas. Não obstante, tal arquitetura visa prover um suporte para resolução deste problema no âmbito do modelo de transações distribuídas, no qual faltas bizantinas são permitidas em determinados elementos do sistema.

Capítulo 6) aborda os aspectos de implementação dos protótipos desenvolvidos para as soluções especificadas pela tese, as avaliações realizadas sobre os mesmos, e descreve as considerações sobre testes de desempenho realizados pela execução dos protótipos.

Capítulo 7) apresenta as conclusões acerca dos resultados obtidos com o desenvolvimento da tese, as contribuições gerais e sugestões para a realização de trabalhos futuros.

2 FUNDAMENTOS

Este capítulo tem como propósito apresentar os principais conceitos que amparam o desenvolvimento desta tese. Como esta tese visa o desenvolvimento de protocolos tolerantes a faltas bizantinas para o processamento de transações em bancos de dados relacionais, inicialmente são apresentados os conceitos atinentes a bancos de dados, transações, e todo o aparato necessário para a formalização dos protocolos objetos desta tese. Além disso, como o trabalho consiste em uma interseção das áreas de transações e de sistemas distribuídos, são apresentados os conceitos a respeito do(s) modelo(s) de computação distribuída adotado(s) para a realização dos trabalhos que compõem o desenvolvimento desta tese.

2.1 CONCEITOS EM BANCOS DE DADOS E TRANSAÇÕES

Nesta seção são apresentadas as definições acerca dos conceitos de bancos de dados e de transações. Neste sentido, é dada ênfase para as definições formais dos termos, conceitos e aspectos necessários para o provimento de um arcabouço teórico, no que diz respeito ao modelo de transações. Ademais, a apresentação preliminar destes se faz necessária, já que eles serão amplamente adotados ao longo desta tese.

2.1.1 Bancos de Dados

Historicamente, o termo "banco de dados" foi mencionado pela primeira vez na literatura no ano de 1964 (SWANSON, 1963), em que o mesmo foi utilizado para descrever uma coleção de entradas de dados que continham itens de informação. Todavia, na mesma época Donald Michie (MICHIE, 1968) sugeriu uma definição mais apropriada para o termo, na qual sua menção ao "banco de dados" referia-se a uma coleção de dados generalizada e não ligada a apenas um conjunto de questões funcionais. Em referências posteriores (DATE, 2004; GARCIA-MOLINA; ULLMAN; WIDOM, 2009; SILBERSCHATZ; KORTH; SUDARSHAN, 2011), a literatura definiu banco de dados como uma coleção de dados relacionados, na qual os dados são definidos como fatos conhecidos que possuem significado e podem ser registrados, os quais são acessíveis, por possivelmente, vários usuários concorrentes.

De outro modo, o termo "banco de dados relacional" (CODD, 1970) foi introduzido na literatura na década de 70 por Edgar Frank Codd, em que o mesmo foi usado para descrever um modelo no qual os dados eram organizados em relações – também conhecidas como tabelas. Tipicamente, um banco de dados relacional é acessado por meio de um sistema de gerência de banco de dados (SGBD), o qual define uma interface para prover um acesso uniforme aos objetos contidos no(s) banco(s) de dados. Este acesso é realizado por parte dos clientes/usuários do banco(s) de dados e ocorre tipicamente por meio de instruções escritas em uma linguagem de alto nível, tal como a SQL (*Structured Query Language*) (GARCIA-MOLINA; ULLMAN; WIDOM, 2009).

Formalmente, um banco de dado consiste em um conjunto finito de itens de dados nomeados (DATE, 2004; ELMASRI; NAVATHE, 2010), em que tais itens são denotados por d_i, onde $i \leq 1 \leq n$ e cada item de dados d_i possui um valor v associado. Os valores associados aos itens de dados, em qualquer tempo, compreendem o estado do banco de dados. Dentre os possíveis estados de um banco de dados, existe um conjunto de estados que reflete a informação correta de uma aplicação, sendo estes estados corretos denominados por "estados consistentes" (DATE, 2004; ELMASRI; NAVATHE, 2010). Uma maneira mais adequada de se expressar formalmente um banco de dados, é a partir da seguinte terminologia.

Definição 2.1 *Um banco de dados $\mathcal{D}$ é uma coleção (ou um conjunto) de itens de dados d_i, tal como descreve a expressão a seguir.*

1. $\mathcal{D} = \bigcup_{i=1}^{n} d_i$ tal que $\forall i \leq n, j \leq n, i \neq j \ : \ d_i \neq d_j$

Um sistema de banco de dados (SBD) pode ser entendido como um ambiente de sistema computacional onde está presente uma componente de software, nomeadamente, o sistema de gerência de bancos de dados (SGBD), e um ou mais bancos de dados (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007). De outro modo, um SGBD consiste em uma coleção de programas que visa auxiliar aos usuários nas tarefas de manutenção e utilização de um banco de dados (RAMAKRISHNAN; GEHRKE, 2003), de modo a fornecer meios convenientes e eficientes para o armazenamento e recuperação de informações em um banco de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 2011). Embora se verifique a existência de diversos tipos de sistemas de gerência de bancos de dados (p. ex.: relacional, baseado em objetos, XML, etc.), o tipo de SGBDs predominante na atualidade é o relacional (STONEBRAKER et al., 2007). A Figura 1, adaptada de (RAMAKRISHNAN; GEHRKE, 2003) e

(HELLERSTEIN; STONEBRAKER; HAMILTON, 2007), ilustra de maneira simplificada a estrutura típica de um SGBD baseado no modelo relacional. Uma breve explanação acerca dos elementos da arquitetura de um SGBD, nos termos da Figura 1, é realizada no que segue.

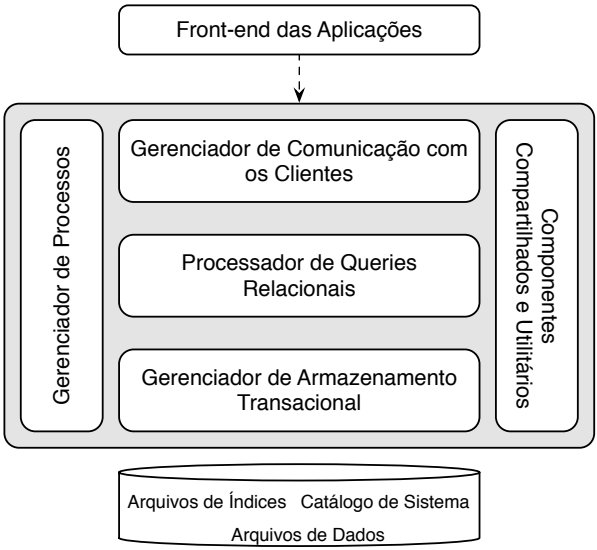


Figura 1 – Arquitetura típica de um SGBD.

O ***Front-end* das aplicações** pode ser entendido como a(s) interface(s) de acesso ao sistema de banco de dados, por meio da(s) qual(is) o(s) usuário(s) interage com o banco de dados (p. ex.: um sistema de informações ERP, um formulário Web, etc.). Para se comunicar com o ambiente de banco de dados, as aplicações tipicamente utilizam de uma API de comunicação com o banco de dados, a qual permite estabelecer conexões com o SGBD através do **Gerenciador de Comunicação com os Clientes**. Note que uma conexão pode ser estabelecida diretamente entre o cliente e o servidor de banco de dados, por meio de APIs como ODBC e JDBC. Além do estabelecimento da conexão, este componente estabelece as credenciais de segurança dos clientes e mantém os estados de cada conexão, a fim de permitir o encaminhamento correto dos comandos a serem processados pelo SGBD.

O **Gerenciador de Processos** entra em ação a partir do momento em que o cliente emite o primeiro comando a ser executado pelo SGBD, onde é criada uma *thread* para atender a requisição, que

é vinculada ao canal de comunicação estabelecido pelo gerenciador de comunicação. Este componente também realiza tarefas como o controle de admissão (p. ex.: processa a requisição imediatamente ou a posterga), escalonamento e despacho dos comandos pela invocação ao **Processador de *Queries* Relacionais**. O **Processador de *Queries* Relacionais** realiza uma série de tarefas que dizem respeito ao comando fornecido para execução, tais como: (*i*) verificação de autorização do usuário para executar aquele comando; (*ii*) compilação do comando em um plano de execução interno; (*iii*) otimização do comando para sua execução.

Em relação ao **Gerenciador de Armazenamento Transacional**, este é o componente responsável por gerenciar todo o acesso aos dados (i.e., leituras) e chamadas realizadas por operações de manipulação de dados. No sistema de armazenamento estão incluídos os algoritmos e estruturas de dados para organização e acesso aos dados presentes no meio de armazenamento persistente (métodos de acesso). Outras tarefas que competem ao componente em questão são: (*i*) o gerenciamento de *buffers*, de modo a verificar quando e como os dados devem ser transferidos entre o meio de armazenamento persistente e memória; (*ii*) o gerenciamento de bloqueios, para assegurar a execução correta das operações em face a concorrência; e (*iii*) o gerenciamento de *logging*, a fim de assegurar que a transação será durável se validada, ou completamente desfeita se anulada.

E por fim, o módulo denominado **Componentes Compartilhados e Utilitários** compreende a um conjunto de ferramentas essenciais para o funcionamento pleno do SGBD. Dentre tais ferramentas estão o gerenciador de catálogo e o gerenciador de memória, ambos invocados durante o processamento da transação; ferramentas de administração e monitoramento do(s) banco(s) de dados, usadas para ajustar e manter o banco de dados; serviços de replicação, dentre outros. Por sua vez, o catálogo é peça vital para a operação do SGBD, sendo usado na análise, otimização e autenticação das operações executadas no banco de dados. É importante salientar que o propósito desta tese não é adentrar em detalhes a respeito da construção de SGBDs, mas apenas prover uma base para o entendimento do leitor. Para uma discussão mais aprofundada, sugere-se a leitura de (RAMAKRISHNAN; GEHRKE, 2003; DATE, 2004; HELLERSTEIN; STONEBRAKER; HAMILTON, 2007).

É digno de nota que a tecnologia empregada nos SGBDs da atualidade incorporam décadas de pesquisa acadêmica e industrial, além de um intenso desenvolvimento de software corporativo. Na atualidade, a tecnologia de banco de dados é algo presente no centro de grande parte

da infra-estrutura de aplicações existentes no mundo. Esta é uma das razões pela qual ao longo dos últimos 30 anos, a tecnologia de banco de dados tem sido considerada como elemento-chave para a concepção de sistemas computacionais, dos mais diversos domínios de aplicação (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007). Não obstante, existem perspectivas de que grande parte dos sistemas computacionais, principalmente os sistemas legados, permaneçam a utilizar os SGBDs para os anos vindouros.

2.1.2 Transações

Nesta tese, estamos interessados em prover a confiabilidade no que concerne ao processamento e terminação de transações em bancos de dados relacionais. Uma transação pode ser vista como uma unidade lógica de trabalho dentro de um SGBD, sendo que esta representa também, uma unidade de interação com esse sistema. Uma transação T é composta por um número arbitrário de operações de leitura (r) e de escrita (w), sendo finalizada por meio de sua validação (*commit*) ou anulação (*abort*). Por construção, uma transação deve satisfazer as propriedades de **atomicidade**, **serialização** e **durabilidade** (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Neste sentido, **atomicidade** significa que, a despeito de qualquer falha durante sua execução, uma transação é executada completamente com êxito, ou do contrário não é executada. De outro modo, **serialização** significa que, mesmo que um conjunto de transações sejam executadas simultaneamente e estas realizem operações sobre o(s) mesmo(s) item(ns) de dados, a execução destas transações será equivalente à execução sequencial em alguma ordem, e as transações terão o mesmo que teriam se tivessem sido executadas uma após a outra, na ordem estabelecida. A **durabilidade** diz respeito ao fato de que os efeitos de uma transação completada e validada não podem mais ser revertidos, mesmo que falhas posteriores ocorram no banco de dados.

Para a apresentação do modelo de transações a partir de sua especificação formal, adotaremos as terminologias e notações de Bernstein *et al.* (1987) e de Weikum e Vossen (2002). A partir daí, é dito que uma transação T_i representa uma sequência finita de operações (o_{ij}) sobre os itens de dados pertencentes a um banco de dados $\mathcal{D}$. Uma operação pode ser uma leitura ($read - r_i$), que retorna o último valor válido e consistente para aquele item de dados; ou uma escrita ($write - w_i$), que atualiza o item de dados com um valor fornecido na

operação. No modelo tradicional, uma transação é iniciada por uma operação de início *begin* (denotada por b_i) e finalizada por uma operação de validação (*commit* – c_i) ou de anulação (*abort* – a_i), estas últimas indicam se a transação foi executada com êxito ou não, respectivamente (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Uma transação é classificada como de "somente-leitura" (*read-only*) se ela não contém nenhuma operação de escrita; ou do contrário, é classificada como transação de "atualização" (*update*).

Para fins de formalização do conceito de transações, considere que $\mathcal{D} = \{d_1, d_2, ..., d_n\}$ é o banco de dados sobre o qual serão executadas as transações em um sistema de banco de dados. Considere também, que para qualquer transação T_i e um item de dados $d_k \in \mathcal{D}$, $r_i[d_k]$ denota uma operação de leitura executada por T_i sobre o item de dados d_k, ao tempo que $w_i[d_k, v]$ denota uma operação de escrita do valor v no item de dados d_k, pela mesma transação. Formalmente, a sequência de operações o_{ij} emitidas pela transação T_i – j denota a j-ésima operação da transação –, pode ser representada por meio de um conjunto parcialmente ordenado $(O_i, <_i)$ (WEIKUM; VOSSEN, 2002; SCHEINERMAN, 2006), tal que:

- $o_{ij} \in \{r_j[d_k], w_j[d_k, v] \mid d_k \in \mathcal{D}\}$
- $O_i = \bigcup_{j=1}^{n<\infty} o_{ij}$ – todas as operações que ocorrem no decurso da transação T_i;

A semântica expressa pela relação de ordem $<_i$ é restrita e linear, e denota a ordenação das operações no tempo, de modo que, $\forall o_{ij}, o_{ik} \in O_i, j \neq k$, se $o_{ij} <_i o_{ik}$ então o_{ij} é executada antes de o_{ik}.

A partir destas definições preliminares, é possível formalizar o modelo de transações a partir da Definição 2.2, apresentada no que segue (BERNSTEIN; HADZILACOS; GOODMAN, 1987; WEIKUM; VOSSEN, 2002).

Definição 2.2 *Uma transação T_i é uma ordem parcial $T_i = \{\mathcal{O}_i, <_i\}$, em que $\mathcal{O}_i$ representa o conjunto de todas as operações executadas em T_i com a relação de ordem $<_i$, onde:*

- $\mathcal{O}_i \subseteq \{b_i\} \cup O_i \cup \{c_i, a_i\}$ – o conjunto das operações $\mathcal{O}_i$ é formado pela união de todas as operações emitidas na transação, com as operações de início e de terminação;
- $c_i \in \mathcal{O}_i \Leftrightarrow a_i \notin \mathcal{O}_i$ – para cada transação, há apenas um *commit* ou um *abort*, não ambos;

- seja a operação o_{ij} a primeira operação emitida pela transação (p. ex.: uma leitura ou uma escrita) e $o_{ik} = b_i$; se $o_{ik}, o_{ij} \in \mathcal{O}_i$, então $o_{ik} <_i o_{ij}$;

- seja a operação $o_{ij} = c_i$ ou $o_{ij} = a_i$ (i.e., aquela que ocorre em T_i), $\forall o_{ik} \in \mathcal{O}_i, j \neq k \Rightarrow o_{ik} <_i o_{ij}$; e

- para quaisquer duas operações $o_{ij}, o_{ik} \in \mathcal{O}_i$, se $o_{ij} = r[d_i]$ e $o_{ik} = w[d_i, v]$ para algum item de dados d_i, então $o_{ij} <_i o_{ik}$ ou $o_{ik} <_i o_{ij}$.

É importante destacar que durante a execução de uma transação, esta passa por uma sequência de estados até ser, de fato, concluída. Neste caso, uma transação também pode ser modelada a partir de uma máquina de estados finita, na qual alguns eventos pré-definidos realizam as transições de um estado a outro. A Figura 2 apresenta a ilustração de uma transação a partir de um diagrama de estados, de acordo com aquele apresentado em (CONNOLLY; BEGG, 2005; ELMASRI; NAVATHE, 2010).

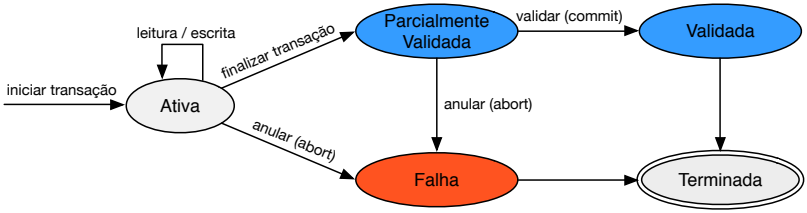


Figura 2 – Máquina de estados finita para uma transação.

No que segue, são descritos alguns aspectos relacionados aos estados mapeados para uma transação, nos termos da Figura 2.

Ativa: logo após seu início, a transação vai para o estado **ativo**, onde permanece durante a execução das operações que a compõem. Se ao término das operações elas foram concluídas com êxito, a transação vai para o estado de **parcialmente validada** onde aguarda o pedido de validação por parte da aplicação. Caso tenha ocorrido alguma falha, a transação segue para o estado de **falha**.

Parcialmente Validada: quando a transação vai para este estado, o sistema de processamento de transações executa alguns protocolos subjacentes para assegurar que, caso haja alguma falha no sistema, ele

seja capaz de recuperar a transação e realizar sua validação (*commit*); ou do contrário, realizar o desfazimento dos efeitos intermediários causados pela execução da transação.

Validada: se a transação atingiu este estado é porque ela teve êxito em suas operações, de modo que elas podem ser validadas de maneira segura, e não mais ser desfeitas.

Falha: se a transação chegou a este estado é porque não houve êxito de execução (em parte ou no todo), e neste caso, todos os efeitos intermediários causados pela execução das operações devem ser desfeitos.

Uma primeira discussão acerca das propriedades de transações em bancos de dados foi elucidada por Jim Gray (GRAY, 1981). Neste sentido, o modelo clássico de transações estabelece que uma transação deve satisfazer as propriedades básicas conhecidas na literatura por ACID (HAERDER; REUTER, 1983). Em suma, estas propriedades visam assegurar a integridade dos dados, a despeito da simultaneidade e de interferências que possam ocorrer durante o processamento da transação. São elas:

- **A**tomicidade: as alterações efetuadas por uma transação são atômicas. Ou todas elas são efetivadas ou nenhuma o será.
- **C**onsistência: uma transação é uma transformação correta dos dados, isto é, as ações da transação não devem violar restrições de integridade, bem como devem conduzir os dados de um estado consistente para outro estado também consistente.
- **I**solamento: a despeito da simultaneidade no processamento de transações, as transações devem ser completamente isoladas umas das outras, de modo que não haja interferências de ações concomitantes entre elas. Cada transação deve ser executada da mesma maneira como se houvesse apenas uma transação em execução.
- **D**urabilidade: se ao término da transação ela for validada, as alterações realizadas por ela tornam-se permanentes e não podem mais ser desfeitas.

É digno de nota que, caso as propriedades ACID não sejam respeitadas, o processamento simultâneo (ou concorrente) de transações pode acarretar em resultados incorretos para as operações. E estes por sua vez, podem incorrer em consequências danosas à base de dados,

mais precisamente no que diz respeito à consistência e integridade dos dados.

2.1.3 Serialização

Em um ambiente onde múltiplas transações podem ser executadas de maneira simultânea, é comum que ocorra a execução intercalada das operações que compõem estas transações. A serialização (*serializability*) é um formalismo utilizado para descrever o comportamento correto e desejado para uma execução simultânea de transações (BERNSTEIN; HADZILACOS; GOODMAN, 1987), a qual define que uma sequência de ações intercaladas por várias transações deve corresponder à execução sequencial destas ações em alguma ordem, a despeito da simultaneidade ocorrida no processamento das mesmas (PAPADIMITRIOU, 1979). Em outros termos, do ponto de vista de uma única transação a noção de serialização é baseada no mesmo princípio da propriedade de **isolamento** definida pelo acrônimo ACID (HAERDER; REUTER, 1983), onde uma transação deve ser executada de maneira completamente isolada de outras simultâneas, de tal maneira que os efeitos parciais de uma transação não sejam visíveis às outras.

Para que seja possível compreender a essência do conceito de serialização, é importante salientar que a sequência de execução (ou ordem de execução) das operações de uma transação denota o que é conhecido por **escalonamento** ou **histórico** (BERNSTEIN; HADZILACOS; GOODMAN, 1987; WEIKUM; VOSSEN, 2002). Assim, seja E uma sequência de execução das transações $T_1, T_2, ..., T_n$, é dito que E é uma execução serial (ou escalonamento serial) (WEIKUM; VOSSEN, 2002) se as transações são executadas sem nenhuma concorrência em E; isto é, cada transação é executada completamente antes da próxima a ser iniciada. É evidente que toda execução serial é correta (BERNSTEIN; HADZILACOS; GOODMAN, 1987), pois, numa execução serial cada transação é executada de maneira independente e sem qualquer interferência de operações de outras transações. Uma vez que cada transação é executada do início ao fim, sem a interferência de outras transações, ela parte de um estado consistente do banco de dados e o conduz para outro estado consistente. Esta é a razão pela qual cada transação produz um resultado correto em sua execução, numa execução serial.

Por outro lado, é dito que uma execução é não-serial (ou escalonamento não-serial) (WEIKUM; VOSSEN, 2002), se as operações de um conjunto de transações são intercaladas – executadas simultaneamente.

A Figura 3 ilustra um exemplo de execução/escalonamento para as transações T_1 e T_2, a partir da notação apresentada em (BERNSTEIN; HADZILACOS; GOODMAN, 1987), em que $r_j[d_i]$ denota uma operação de leitura sobre o item de dados d_i, e $w_j[d_i, v]$ denota uma operação de escrita do valor v sobre o item de dados d_i.

$$T_1 = r_1[d_i], w_1[d_i, d_i + 10], c_1$$
$$T_2 = r_2[d_i], r_2[d_j], w_2[d_j, d_i + d_j], c_2$$
$$E \Rightarrow r_1[d_i], r_2[d_i], w_1[d_i, d_i + 10], r_2[d_j], w_2[d_j, d_i + d_j], c_1, c_2$$

Figura 3 – Exemplo de execução/escalonamento de transações.

Um escalonamento não-serial é serializável se ele produz o mesmo estado que um escalonamento serial, isto é, se tanto o resultado produzido por ele como o efeito causado sobre os dados for igual aquele produzido a partir de uma execução serial. De maneira intuitiva, a execução simultânea de um conjunto de transações deve obter resultados equivalentes àqueles obtidos a partir da execução sequencial para o mesmo conjunto de transações, isto é, uma após a outra, em alguma ordem (STEARNS; LEWIS; ROSENKRANTZ, 1976). Formalmente, esta verificação é realizada a partir do que é definido na literatura como "execução serializável" ou "escalonamento serializável" (STEARNS; LEWIS; ROSENKRANTZ, 1976; BERNSTEIN; HADZILACOS; GOODMAN, 1987). Para tanto, algumas definições devem ser apresentadas, a fim de prover o arcabouço necessário para o entendimento acerca destes conceitos.

Todavia, antes de adentrar na discussão acerca da teoria da serialização, é importante salientar que no âmbito da literatura pertinente, os termos **histórico** e **escalonamento** são utilizados para denotar o mesmo conceito (BERNSTEIN; HADZILACOS; GOODMAN, 1987; WEIKUM; VOSSEN, 2002; ELMASRI; NAVATHE, 2010), em que ambos compreendem a formalismos que permitem modelar a execução simultânea (ou concorrente) de transações. Por questões de homogeneidade, nesta tese utilizaremos o termo **histórico** por estar em maior consonância com as especificações a serem apresentadas ao longo deste documento.

Como dito, no contexto da teoria da serialização, um formalismo que permite modelar a execução simultânea (ou concorrente) de transações é a noção de histórias (ou históricos) (BERNSTEIN; HADZILACOS; GOODMAN, 1987), a qual denota a ordem na qual operações de diferentes transações foram executadas – i.e., indica a ordenação das operações das transações (ELMASRI; NAVATHE, 2010). Mais precisamente, um histórico compreende a uma lista que contém todas as

operações executadas pelas transações, da forma como elas foram intercaladas em suas respectivas execuções. Neste sentido, um histórico pode ser definido pelo que segue:

Definição 2.3 *Seja* $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ *um conjunto de transações executadas simultaneamente, para o qual* $\mathcal{H}$ *denota o histórico completo de todas as operações executadas em* $\mathcal{T}$*. Então, tem-se que* $\mathcal{H}$ *é uma ordem parcial* $<_H$*, de operações executadas pelo conjunto de transações em* $\mathcal{T}$*, tal que:*

- $\forall T_i \in \mathcal{T}, \exists(\mathcal{O}_i, <_i)$
- $\mathcal{H} = \bigcup_{i=1}^{n} \mathcal{O}_i$ – todas as operações de cada transação T_i;
- $\bigcup_{i=1}^{n} <_i \subseteq <_H$
- $\forall o_{ij}, o_{ik} \in \mathcal{O}_i, j \neq k$: se $o_{ij} <_i o_{ik}$ em T_i, então $o_{ij} <_H o_{ik}$;
- se em T_i é lido algum item de dados d_i escrito por T_j, então $w_j[d_i, v] <_H r_i[d_i]$

A partir destas definições preliminares, é possível estabelecer a relação entre histórico e serialização, em que um histórico de execução $\mathcal{H}$ é serializável se e somente se produzir um estado igual ao que seria produzido por alguma execução serial das transações em $\mathcal{H}$ (STEARNS; LEWIS; ROSENKRANTZ, 1976; BERNSTEIN; HADZILACOS; GOODMAN, 1987; WEIKUM; VOSSEN, 2002). Note que os efeitos (ou estado) consistem nos valores produzidos por operações de escrita realizadas pelas transações não anuladas. Outrossim, é dito que um sistema fornece isolamento serializável se todos os históricos de execução que ele permite são serializáveis. Ainda sobre históricos, dois aspectos elementares que permitem identificar a corretude destes sobre a execução de transações são: (*i*) toda transação, se executada de maneira isolada, conduzirá um banco de dados com estado consistente, para outro banco de dados com estado também consistente (GRAY, 1981); (*ii*) todo escalonamento serial pode ser considerado como correto (BERNSTEIN; HADZILACOS; GOODMAN, 1987).

Na literatura são encontradas diversas maneiras de se verificar a corretude quanto à execução de transações, sendo todas baseadas no conceito de serialização, porém, que conduzem a diferentes noções de execução serializável (STEARNS; LEWIS; ROSENKRANTZ, 1976; WEIKUM; VOSSEN, 2002). Estas formas de serialização são definidas a partir do conceito de equivalência de históricos (ou de escalonamentos),

de modo que, a partir destas é possível verificar se um escalonamento é serializável (i.e., se é equivalente a um escalonamento serial) (WEIKUM; VOSSEN, 2002). Neste contexto, é dito que dois históricos $\mathcal{H}$ e $\mathcal{H}'$ são equivalentes se envolvem o mesmo conjunto de transações $\mathcal{T}$ e produzem os mesmos efeitos no banco de dados. As formas mais comuns de se estabelecer a equivalência de históricos para verificação da serialização são a **equivalência por estado final**, a **equivalência em visão** e, a **equivalência em conflito** (STEARNS; LEWIS; ROSENKRANTZ, 1976; BERNSTEIN; HADZILACOS; GOODMAN, 1987; BRAYNER, 1999; WEIKUM; VOSSEN, 2002; BERNSTEIN; NEWCOMER, 2009; ÖZSU; VALDURIEZ, 2011; SILBERSCHATZ; KORTH; SUDARSHAN, 2011). A Figura 4 ilustra o relacionamento entre as noções de serialização mais comuns da literatura (WEIKUM; VOSSEN, 2002).

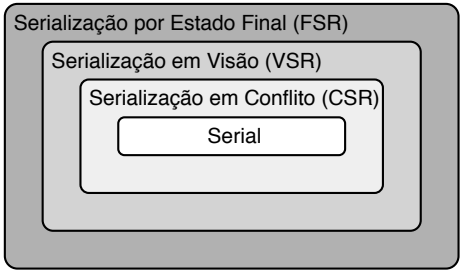


Figura 4 – Relacionamento entre as noções de serialização.

A "equivalência por estado final" (ou de resultado) define a serialização por estado final (do inglês, *Final State Serializability* – FSR) (BRAYNER, 1999; WEIKUM; VOSSEN, 2002; ELMASRI; NAVATHE, 2010), em que dois históricos $\mathcal{H}$ e $\mathcal{H}'$ são equivalentes, se para um mesmo conjunto de transações e um mesmo estado inicial do banco de dados, produzem o mesmo resultado final. A noção de "equivalência por estado final" visa identificar históricos que realizam as mesmas transições de estados, quando executadas sobre um mesmo estado inicial do banco de dados. Note que esta é a forma mais simples, porém, pouca satisfatória para determinar a corretude de escalonamentos/históricos. Embora descaracterizado pela definição de serialização por estado final (vide Definição 2.4), casos particulares podem induzir a situações em que dois históricos completamente diferentes – i.e., que executam diferentes transações – acidentalmente resultem em um mesmo estado final. A serialização por estado final é formalizada pela Definição 2.4.

Definição 2.4 *A execução concorrente de um conjunto de transações denotada por um histórico $\mathcal{H}$ é serializável por estado final se existe um histórico serial $\mathcal{H}'$ tal que, $\mathcal{H}$ e $\mathcal{H}'$ são construídos a partir do mesmo conjunto de transações e, a partir de um mesmo estado inicial, o estado final produzido por ambos os históricos é o mesmo (i.e., $\mathcal{H} \equiv_f \mathcal{H}'$).*

Um segundo tipo de equivalência conhecido por "equivalência em visão", é utilizada para definir o conceito de serialização em visão (*view serializability*) (STEARNS; LEWIS; ROSENKRANTZ, 1976), que diferente do que ocorre na FSR, leva em consideração a relação entre as operações de leitura e de escrita realizadas pelas transações contidas no histórico de execução $\mathcal{H}$. Neste sentido, o formalismo intermediário que permite estabelecer a relação entre as ações das transações, é a relação "*reads-from*" (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Esta relação pode ser definida pela seguinte asserção.

Definição 2.5 *Uma transação T_i lê um item de dados d_i de T_j, se T_j é a transação que efetuou a última operação de escrita em d_i e teve sua validação ocorrida antes da leitura do valor de d_i por T_i. Portanto, é dito que a relação reads-from ocorre para duas transações $T_i, T_j \in \mathcal{H}$ se:*

1. $w_j[d_i, v] <_H r_i[d_i]$;
2. $c_j \in \mathcal{O}_j \wedge c_j <_H r_i[d_i]$; e
3. se $\exists w_k[d_i, v] \in \mathcal{O}_k : w_j[d_i, v] <_H w_k[d_i, v] <_H r_i[d_i]$, então $a_k \in \mathcal{O}_k$ e $a_k <_H r_i[d_i]$.

Deste modo, é dito que T_i lê de T_j em $\mathcal{H}$ (i.e., T_i *reads-from* T_j), se T_i lê algum item de dados escrito por T_j no histórico $\mathcal{H}$ (p. ex.: o último valor escrito naquele item de dados). Note que a relação *reads-from* é utilizada para denotar apenas a relação entre operações de transações distintas, de modo que operações realizadas no âmbito de uma mesma transação não são consideradas. Esta distinção se faz necessária porque uma transação pode ler itens de dados escritos por ela mesma (p. ex.: $w_i[d_i, v] <_i r[d_i]$).

Como premissa básica da teoria da serialização, considera-se que um histórico $\mathcal{H}$ é serializável se, para cada duas transações $T_i, T_j \in \mathcal{H}$, todas as operações de T_i ocorrem antes de todas as operações de T_j, ou vice-versa (i.e., $\mathcal{O}_i <_H \mathcal{O}_j$ ou $\mathcal{O}_j <_H \mathcal{O}_i$). Não obstante, a verificação acerca da equivalência em visão é realizada a partir de dois históricos (ou escalonamentos) $\mathcal{H}$ e $\mathcal{H}'$, de modo que algumas condições

devem ser satisfeitas (BERNSTEIN; HADZILACOS; GOODMAN, 1987; SILBERSCHATZ; KORTH; SUDARSHAN, 2011). Para tanto, tem-se a seguinte definição:

Definição 2.6 *Dois históricos $\mathcal{H}$ e $\mathcal{H}'$ são equivalentes em visão (i.e., $\mathcal{H} \equiv_v \mathcal{H}'$), se:*

1. eles são relacionados a um mesmo conjunto de transações;
2. para cada item de dados d_i, se a transação T_i lê um valor inicial de d_i em $\mathcal{H}$, então a transação T_i também lê o mesmo valor inicial de d_i em $\mathcal{H}'$;
3. para cada item de dados d_i, tal que $1 \leq i \leq n$, se $w_j[d_i, v]$ é a última escrita sobre d_i em $\mathcal{H}$, então $w_j[d_i, v]$ é também a última escrita sobre d_i em $\mathcal{H}'$.

Note que o princípio por trás da noção de equivalência em visão é norteado pelo fato de que, como cada operação de leitura de uma transação lê o valor resultante da mesma operação de escrita, as operações de escrita de cada transação devem produzir o mesmo estado final no banco de dados.

Definição 2.7 *A execução concorrente de um conjunto de transações denotada por um histórico $\mathcal{H}$ é serializável em visão se existe um histórico serial $\mathcal{H}'$ possível para o mesmo conjunto de transações tal que, em ambas as execuções, cada transação lê os mesmos valores de itens de dados, sejam estes iniciais ou resultantes de operações de escrita, e os estados finais produzidos são os mesmos. Ou seja, se $\mathcal{H} \equiv_v \mathcal{H}'$.*

Uma discussão sobre o terceiro e último tipo de equivalência a ser apresentado requer o entendimento acerca do conceito de conflito. Neste sentido, a partir do critério estabelecido por Papadimitriou (PAPADIMITRIOU, 1986), um conflito entre duas transações pode ser definido da seguinte forma:

Definição 2.8 *Seja T_i e T_j duas transações, e o_{ik} e o_{jl} operações de T_i e T_j, respectivamente. o_{ik} e o_{jl} estão em conflito se ambas operam sobre o mesmo item de dados d_i e, pelo menos, uma delas é uma operação de escrita, tal que:*

- $T_i, T_j \in \mathcal{H}, i \neq j : \exists o_{ik}[d_i] \in \mathcal{O}_i, o_{jl}[d_i] \in \mathcal{O}_j$; e
- $o_{ik}, o_{jl} \in \{r, w\}$; e

- $(o_{ik} = r \wedge o_{jl} = w) \vee (o_{ik} = w \wedge o_{jl} = r) \vee (o_{ik} = w \wedge o_{jl} = w)$

E finalmente, a serialização em conflito (*conflict serializability*) (STEARNS; LEWIS; ROSENKRANTZ, 1976) é definida em termos da noção de "equivalência em conflito" (do inglês, *conflict equivalence*) (GRAY; LORIE; PUTZOLU, 1975). Esta noção de equivalência estipula que um histórico $\mathcal{H}$ – que denota a execução concorrente de um conjunto de transações – é "equivalente em conflito" a uma execução sequencial do mesmo conjunto de transações (denotado por $\mathcal{H}'$), se tanto no histórico $\mathcal{H}$ como na execução sequencial $\mathcal{H}'$, a ordenação das operações em conflito ocorrem da mesma maneira, isto é, se em ambos os casos elas têm a mesma relação de precedência para as operações em conflito (STEARNS; LEWIS; ROSENKRANTZ, 1976). Em outros termos, diz-se que dois históricos de execução são equivalentes em conflito, se as operações que estão em conflito nas transações não anuladas no histórico, aparecem na mesma ordem em ambos os históricos (i.e., em $\mathcal{H}$ e $\mathcal{H}'$) (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Assim, é dito que um histórico $\mathcal{H}$ é serializável em conflito, se for equivalente em conflito a um histórico serial.

Definição 2.9 *A execução concorrente de um conjunto de transações num histórico $\mathcal{H}$ é serializável em conflito se existe um histórico serial $\mathcal{H}'$ possível para o mesmo conjunto de transações tal que, em ambas as execuções, a ordem das operações em conflito é a mesma. Isto é, se $\mathcal{H} \equiv_c \mathcal{H}'$.*

A análise de serialização em conflito sobre um histórico de execução $\mathcal{H}$ ocorre por meio de um grafo direcionado denominado por "grafo de serialização" (ou *serialization graph* – SG) (BERNSTEIN; HADZILACOS; GOODMAN, 1987). No caso, as transações $\mathcal{T}$ que ocorrem em $\mathcal{H}$ são mapeadas para o grafo $SG(\mathcal{H}) = (V, E)$, onde $V = \mathcal{T}$ e $T_i \rightarrow T_j \in E \Leftrightarrow (T_i, T_j \in \mathcal{T}) \wedge (\exists o_{ik} \in \mathcal{O}_i, o_{jl} \in \mathcal{O}_j)$, tal que o_{ik} conflita com o_{jl} e $o_{ik} <_H o_{jl}$. A seta $\rightarrow$ denota a relação de precedência entre as transações T_i e T_j. Deste modo, se $T_i \rightarrow T_j$ é porque T_i executa uma operação sobre um determinado item de dados d_i que está em conflito mas precede alguma operação executada em T_j, sobre o mesmo item de dados d_i. A partir daí, o teorema da serialização demonstrado por Bernstein *et al.* (1987) estabelece que um conjunto de transações em um histórico $\mathcal{H}$ é serializável em conflito, se e somente se $SG(\mathcal{H})$ é um grafo acíclico.

Por fim, é importante salientar que a noção de equivalência mais adotada comumente é serialização em conflito (ELMASRI; NAVATHE,

2010; SIPPU; SOISALON-SOININEN, 2014). Isso decorre do fato de que, embora a serialização em conflito seja mais restritiva do que a serialização por estado final e serialização em visão (vide diagrama da Figura 4), o algoritmo usado para verificar históricos corretos a partir da serialização em conflito tem complexidade polinomial, ao passo que os baseados na serialização por estado final e na serialização em visão são considerados como problemas NP-completos (PAPADIMITRIOU, 1986).

2.1.3.1 Serialização em Bancos de Dados Não-Replicados

Em um ambiente convencional de banco de dados, a serialização de transações (i.e., a execução correta) tipicamente é obtida por meio de um mecanismo de controle de concorrência. O controle de concorrência pode ser entendido como a atividade de coordenar ações das transações que ocorrem de maneira simultânea, que operam sobre um mesmo conjunto de dados, em que tais ações interferem potencialmente umas nas outras (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Neste sentido, o controle de concorrência é um aspecto elementar para o processamento de transações, que visa gerenciar a execução de operações simultâneas sobre dados, de modo que elas não interfiram umas nas outras, a fim de não violar a integridade do banco de dados (CONNOLLY; BEGG, 2005). Na literatura são encontradas diversas técnicas para a realização do controle de concorrência de transações, sendo que as mais comumente verificadas são o bloqueio (*locking*), ordenação por marca de tempo (*timestamp ordering*), otimista e multiversão (BERNSTEIN; HADZILACOS; GOODMAN, 1987; THOMASIAN, 1998; RAHIMI; HAUG, 2010; ÖZSU; VALDURIEZ, 2011).

Controle de Concorrência Baseado em Bloqueio

O controle de concorrência baseado em bloqueios é um dos métodos mais empregados para implementar o isolamento de transações em bancos de dados (GRAY; REUTER, 1992). Os bloqueios consistem em travas que são associadas aos itens de dados de um banco de dados, a fim de evitar acessos concomitantes sobre um mesmo item de dados e prevenir as transações de entrar em situações de conflito. Com um bloqueio é possível assegurar que um item de dados compartilhado por operações conflitantes é acessado por apenas uma operação em um dado momento. Mais precisamente, esta abordagem atrasa a execução de operações em conflito, por meio da aquisição de uma trava sobre o

item de dados, antes de realizar a operação desejada. Nesta abordagem, dois tipos de bloqueios são permitidos: o bloqueio compartilhado (*SL – shared lock*); e o bloqueio exclusivo (*XL – exclusive lock*). O bloqueio compartilhado pode ser usado simultaneamente por mais de uma transação, para realizar uma operação de leitura sobre um mesmo item de dados. Por outro lado, um bloqueio exclusivo não permite outros bloqueios sobre um mesmo item de dados, sendo usado para operações de escrita e de leitura. O Quadro 1 mostra a compatibilidade entres os tipos de bloqueio mencionados.

Quadro 1 – Compatibilidade entre os tipos de bloqueio.

transação T_i / **transação** T_j	$leitura(d_i) \Rightarrow SL(d_i)$	$escrita(d_i) \Rightarrow XL(d_i)$
$leitura(d_i) \Rightarrow SL(d_i)$	sempre permitido	se $i \neq j$ rejeitado se $i = j$ permitido
$escrita(d_i) \Rightarrow XL(d_i)$	se $i \neq j$ rejeitado se $i = j$ permitido	se $i \neq j$ rejeitado se $i = j$ permitido

O esquema de bloqueio mais referenciado na literatura é o bloqueio em duas fases (do inglês, *Two-Phase Locking* – 2PL), o qual foi introduzido por Eswaran *et al.* (1976). O 2PL pode ser visto como um padrão em termos de controle de concorrência, dada sua profusão no âmbito da indústria de software para SGBDs comerciais (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007; BERNSTEIN; NEWCOMER, 2009). O termo "duas fases" advém da forma como é realizado o gerenciamento dos bloqueios nesta abordagem, que conforme sugere o nome, ocorre em duas fases. Neste caso, a aquisição dos bloqueios ocorre durante a **fase de crescimento**, enquanto que a liberação é realizada durante a **fase de encolhimento** (BERNSTEIN; NEWCOMER, 2009; CONNOLLY; BEGG, 2005). A especificação básica do 2PL define que não pode ocorrer nenhuma liberação de bloqueio durante a fase de crescimento, do mesmo modo que não pode haver qualquer aquisição de bloqueio na fase de encolhimento. Além disso, pela mesma especificação três regras são estabelecidas (BERNSTEIN; HADZILACOS; GOODMAN, 1987):

1. Quando uma solicitação de bloqueio é recebida, é verificado se o bloqueio ora solicitado conflita com algum bloqueio já concedido. Se afirmativo, o pedido de bloqueio é armazenado em uma fila, a fim de retardar a transação até que aquele pedido possa ser atendido. Do contrário, o bloqueio é concedido.

2. Uma vez que um bloqueio sobre um item de dados tenha sido

concedido a uma dada transação, ele não pode ser liberado até que a operação correspondente ao bloqueio em questão tenha sido processada.

3. Uma vez que um bloqueio tenha sido liberado para uma determinada transação, posteriormente, nenhum outro bloqueio pode ser obtido para aquela transação.

Além do 2PL básico descrito na presente seção, outras variações do 2PL são encontradas na literatura (ELMASRI; NAVATHE, 2010): (*i*) 2PL conservador; (*ii*) 2PL restrito; e (*iii*) 2PL rigoroso. O 2PL conservador é caracterizado pela maneira na qual ocorre a aquisição dos bloqueios, isto é, de maneira pré-declarada, em que todos os itens de dados utilizados pela transação devem ser bloqueados antes da execução desta. Assim, se algum item de dados não puder ser bloqueado no momento da solicitação, nenhum bloqueio é adquirido e a transação é atrasada até que todos os itens de dados estejam disponíveis. Por outro lado, o 2PL restrito não permite a liberação de bloqueios exclusivos (i.e., de escrita) até que a transação seja validada ou anulada – o que significa que nenhuma outra transação pode ler ou escrever sobre o(s) mesmo(s) item de dados, até que a transação que detém a posse do(s) bloqueio(s) sobre eles seja finalizada. Por fim, o 2PL rigoroso consiste em uma derivação do 2PL restrito, em que todos os bloqueios (i.e., de leitura e de escrita) são mantidos até o término da transação, sendo estes liberados após a validação ou anulação da transação. Note que tanto o 2PL restrito como o 2PL rigoroso só permitem a leitura de itens de dados escritos por transações já validadas, isto é, que já são estáveis. Isto implica que nunca será necessário anular uma transação em decorrência da leitura de itens de dados obsoletos; o que por conseguinte, evita o que é conhecido na literatura como anulação em cascata (*cascade abort*) (BERNSTEIN; NEWCOMER, 2009).

Controle de Concorrência Baseado em Ordenação por Marca de Tempo

A ideia da ordenação por marca de tempo (i.e., *timestamp ordering*) foi proposta inicialmente por Lamport (1978), para ordenar eventos em sistemas de computação distribuída. No contexto de sistemas de banco de dados, diferente do que ocorre na abordagem baseada em bloqueio, o controle de concorrência baseado na ordenação por marca de tempo não busca obter a serialização através de exclusão mútua. Em vez disso, esta abordagem determina uma ordem de serialização, e

executa as transações em consonância com tal ordem. Neste sentido, para estabelecer esta ordem, cada transação T recebe uma marca de tempo única na inicialização – denotada por $ts(T)$ –, em que o valor de $ts(T)$ é anexado a cada operação emitida pela transação T.

Numa execução baseada nesta abordagem de controle de concorrência, as operações em conflito são ordenadas de acordo com as marcas de tempo das respectivas transações, de modo a respeitar a seguinte regra básica (RAMAKRISHNAN; GEHRKE, 2003):

- se $o_i(d_k)$ e $o_j(d_k)$ são operações conflitantes, então o processamento ocorre na ordem $o_i(d_k) <_H o_j(d_k)$ se e somente se $ts(T_i) < ts(T_j)$.

Para que seja possível implementar um esquema que atenda a regra ora apresentada, para cada item de dados $d_k \in \mathcal{D}$ são associadas marcas de tempo relativas às operações de leitura e de escrita, denotados por $rts(d_k)$ e $wts(d_k)$, respectivamente. No caso, quando uma transação T_i emitir uma operação de leitura ou de escrita sobre o item de dados d_k, é realizada uma comparação entre os valores de $ts(T_i)$ e $rts(d_k)$ e $wts(d_k)$, a fim de assegurar que a ordem relativa a marca de tempo de execução da transação não seja violada. Se a comparação em questão verifica uma violação na ordem, a transação T_i é anulada e deve ser reiniciada com uma nova marca de tempo. Do mesmo modo, se alguma transação T_j leu algum item de dados escrito por T_i, T_j deve ser anulada e assim sucessivamente – um efeito de anulação em cascata (BERNSTEIN; NEWCOMER, 2009). Mais precisamente, se T_i executa uma operação de **leitura** sobre o item de dados d_k, a seguinte verificação é realizada:

- **se** $wts(d_k) > ts(T_i)$, a operação é rejeitada e T_i é anulada, pois alguma transação com marca de tempo posterior, e portanto, mais recente, atualizou o valor de d_k;
- **senão**, T_i lê o valor de d_k e atualizada marca de tempo relativa à leitura de d_k tal que $rts(d_k) = max(rts(d_k), ts(T_i))$.

Por outro lado, se T_i executa uma operação de **escrita** sobre d_k, a condição a seguir deve ser verificada:

- **se** $rts(d_k) > ts(T_i)$ ou $wts(d_k) > ts(T_i)$, a operação é rejeitada e T_i é anulada, pois alguma transação com marca de tempo posterior, e portanto, mais recente, leu ou atualizou o valor de d_k – condição que viola a ordenação baseada na marca de tempo;

- **senão**, T_i escreve o valor desejado em d_k e atualizada marca de tempo relativa à escrita de d_k tal que $wts(d_k) = ts(T_i)$.

Note que a rejeição de operações em conflito, nos termos das condições ora apresentadas, pode causar a anulação de muitas transações, de modo que algumas delas podem vir a sofrer por inanição (i.e., *starvation* – vide Seção 2.1.4.3) devido a sucessivos conflitos e anulações. Em decorrência disso, uma pequena variação na especificação desta abordagem permite reduzir o número de rejeições, variação esta que é conhecida por **regra de escrita de Thomas** (ou *Thomas Write Rule*) (THOMAS, 1979). No caso, a regra de escrita de Thomas modifica a condição para evitar a rejeição de uma operação de escrita sobre o item de dados d_k, emitida pela transação T_i da seguinte maneira:

- **se** $rts(d_k) > ts(T_i)$, a operação é rejeitada e T_i é anulada;
- **se** $wts(d_k) > ts(T_i)$, então T_i não executa a operação de escrita e T_i não é anulada, pois alguma transação com marca de tempo posterior, e portanto, mais recente, atualizou o valor de d_k. Por esta razão, a operação pode ser ignorada porque o valor a ser escrito em d_k já está obsoleto;
- **se** nenhuma das condições anteriores é verificada, então T_i escreve o valor desejado em d_k e atualizada marca de tempo relativa à escrita de d_k tal que $wts(d_k) = ts(T_i)$

A regra de escrita de Thomas parte do princípio de que a operação de escrita ignorada nunca será vista por qualquer outra transação, e, portanto, não violará a serialização.

Controle de Concorrência Otimista

O controle de concorrência otimista, também conhecido por controle de concorrência baseado em validação ou certificação, foi proposto por Kung e Robinson (1981). Nesta abordagem, diferente do que ocorre nas técnicas para controle de concorrência anteriormente apresentadas, nenhuma verificação acerca da serialização é realizada durante o processamento da transação, de modo que é permitido que múltiplas transações efetuem operações de leitura e de escrita concomitantes sobre um mesmo item de dados, sem qualquer tipo de bloqueio. A verificação quanto ao isolamento, serialização e conflitos é realizada *a posteriori*, quando a transação requisita a validação, e por isso, cada transação mantém um histórico de suas leituras e escritas, a serem utilizados

nesta verificação. Neste sentido, se porventura a(s) verificação(ões) realizada(s) na fase de validação determinar(em) que a transação em questão viola a serialização, esta é penalizada por meio de sua anulação não-espontânea[1].

Note que esta abordagem parte da premissa de que as transações não entrarão em conflito umas com as outras, de modo que as transações ocorrem em três fases:

1. **Leitura**: a transação é iniciada nesta fase, onde também ocorre todas as operações de leituras para os itens de dados desejados, e as operações de escrita, que são executadas num espaço temporário privado e local;

2. **Validação**: se a transação manifesta sua intenção em ser validada (i.e., *commit*), é realizado um teste de validação para determinar se as operações que foram realizadas no espaço temporário podem ser refletidas sobre o estado do banco de dados, de modo a preservar a serialização. Se é verificada a existência de algum conflito ou situação que possa violar a serialização, a transação é anulada e todo o conteúdo do espaço temporário é descartado;

3. **Escrita**: se a transação obteve sucesso em sua validação, então as escritas realizadas sobre os itens de dados no espaço temporário são copiadas para os respectivos itens de dados do banco de dados.

Em se tratando especificamente da fase de validação – que consiste numa das etapas mais importantes desta técnica –, cada transação recebe uma marca de tempo ao adentrar nesta fase, onde um critério de validação verifica se a ordem da marca de tempo das transações é uma ordem serial equivalente. Assim, para cada par de transações T_i e T_j, tal que $ts(T_i) < ts(T_j)$, o critério de validação leva em conta algumas condições de validação, sendo que ao menos uma delas deve ser satisfeita (RAMAKRISHNAN; GEHRKE, 2003):

1. a transação T_i termina todas as fases antes do início de T_j;

2. a transação T_i termina todas as fases antes que T_j inicie sua fase de escrita, e T_i não escreve em nenhum item de dados que foi lido por T_j;

[1] No âmbito desta tese, o termo 'não-espontânea' refere-se a uma ação contrária àquela que foi solicitada pela transação.

3. a transação T_i termina sua fase de leitura antes que T_j termine sua fase de leitura, e T_i não escreve em nenhum item de dados que seja lido ou escrito por T_j.

Note que cada uma destas condições assegura que as escritas realizadas por T_j não sejam visíveis para T_i. Para tanto, para validar uma transação T_j, pelo menos uma destas condições deve ser válida em relação às transações T_i já validadas, em que $ts(T_i) < ts(T_j)$. A primeira condição permite que T_j pode ler as escritas efetuadas por T_i, ao tempo que estabelece uma relação de ordem estritamente serial quanto à execução de T_i em relação à T_j. De outro modo, a segunda condição permite que T_j leia valores para itens de dados, enquanto T_i ainda está a escrever sobre outros itens de dados. E embora seja permitido que T_j escreva sobre os mesmos itens de dados escritos por T_i, todas as escritas de T_i precedem todas as escritas de T_j. Por fim, a terceira condição permite que T_i e T_j realizem operações de escrita de maneira concorrente, porém, os conjuntos dos itens de dados escritos pelas duas transações são disjuntos. Deste modo, se qualquer uma destas condições for satisfeita, nenhum conflito de leitura/escrita, escrita/leitura ou escrita/escrita é possível.

Controle de Concorrência Multiversão

Uma abordagem alternativa que visa melhorar o desempenho em face ao rigor imposto pela serialização é o controle de concorrência multiversão (ou *multiversion concurrency control* – MVCC)(REED, 1978; BERNSTEIN; GOODMAN, 1983; MURO; KAMEDA; MINOURA, 1984). Diferente do que ocorre com as abordagens vistas até então, a concepção inicial por trás desta abordagem era evitar atrasos na execução de operações das transações, bem como a rejeição de operações ou ainda a anulação não-espontânea de transações. Este feito é realizado a partir da manutenção de múltiplas versões para cada item de dados, de modo que cada operação de escrita sobre um item de dados d_k, em vez de substituir o valor existente de d_k, cria uma nova versão para aquele item de dados, fazendo com que os valores antigos (i.e., versões) sejam mantidos (REED, 1978). Evidentemente que a desvantagem desta abordagem em relação às demais é o custo de armazenamento, para manter as diversas versões de cada item de dados.

Um dos objetivos para a proposição desta abordagem é evitar que uma transação tenha que aguardar para realizar uma operação de leitura sobre um item de dados, bem como evitar que operações de leitura sejam rejeitadas, e assim manter a serialização através de

obtenção de valores antigos de um item de dados. No caso, quando uma operação de leitura é emitida sobre o item de dados d_k, uma versão apropriada daquele dado, de acordo com a execução atual, é selecionada como resultado da operação. Para tanto, um dos aspectos mais cruciais para esta abordagem é que o protocolo deve assegurar que a seleção dos valores proveja a serialização. Outrossim, por razões de desempenho, é importante que uma transação possa determinar de maneira rápida e fácil, qual(is) versão(ões) de item(ns) de dados ela deve ler (SILBERSCHATZ; KORTH; SUDARSHAN, 2011).

É importante salientar que a literatura dispõe de vários esquemas que especificam diferentes maneiras para implementar o controle de concorrência multiversão (ELMASRI; NAVATHE, 2010). Todavia, por razões de coerência, este documento se limita a apresentar apenas os esquemas mais comuns da literatura, que são o MVCC baseado em ordenação por marca de tempo e o MVCC baseado em bloqueios (RAMAKRISHNAN; GEHRKE, 2003; ELMASRI; NAVATHE, 2010; SILBERSCHATZ; KORTH; SUDARSHAN, 2011).

MVCC baseado em ordenação por marca de tempo. Como sugere o nome, o MVCC baseado em ordenação por marca de tempo consiste numa extensão da abordagem de ordenação por marca de tempo já apresentada. Neste esquema, a cada transação T_i criada é associada uma marca de tempo única, denotada por $ts(T_i)$. Além disso, para cada item de dados $d_k \in \mathcal{D}$, uma sequência de versões é mantida (p. ex.: $d_{k1}, d_{k2}, ..., d_{kn}$), onde para cada versão d_{kj} está associado três componentes:

1. **conteúdo**: que consiste no valor da versão d_{kj};
2. $\boldsymbol{wts(d_{kj})}$: marca de tempo da transação que criou a versão d_{kj};
3. $\boldsymbol{rts(d_{kj})}$: a maior marca de tempo de transações que leram, com êxito, a versão d_{kj}.

A partir daí, uma transação T_i que emite uma operação de escrita sobre d_k, cria uma nova versão d_{kj}. No caso, a componente **conteúdo** armazena o valor fornecido por T_i na operação de escrita, enquanto que as componentes $wts(d_{kj})$ e $rts(d_{kj})$ são inicializadas com o valor de $ts(T_i)$. De outro modo, o valor da componente $rts(d_{kj})$ é atualizado sempre que uma transação T_l emitir uma operação leitura sobre a versão d_{kj} e $rts(d_{kj}) < ts(T_l)$. No mesmo ensejo, para que seja possível assegurar a serialização o esquema devem atender as especificações seguintes. Suponha que a transação T_i queira emitir uma operação de

leitura ou uma operação de escrita sobre o item de dados d_k, e que d_{kj} denota a versão de d_k, tal que $wts(d_{kj}) \leq ts(T_i)$. Neste caso:

- **se** T_i executa uma operação de leitura sobre d_k, o valor retornado é aquele da componente **conteúdo** de d_{kj};
- **se** T_i executa uma operação de escrita sobre d_k, e se $ts(T_i) < rts(d_{kj})$, T_i é anulada; do contrário, uma nova versão d_{kl} é criada.

Note que estas especificações devem ser atendidas pelas seguintes razões: (*i*) uma transação sempre deve ler a versão mais recente de um item de dados; (*ii*) uma transação deve ser anulada se ela estiver muito atrasada para realizar uma operação de escrita. A segunda condição pode ocorrer, se T_i tentar escrever uma versão que alguma outra transação T_j pode ter lido, e neste caso, não é correto permitir a execução de tal operação de escrita. Uma das características mais interessantes deste esquema é que uma operação de leitura jamais falhará, tampouco terá de aguardar.

MVCC baseado em bloqueios. Trata-se de um esquema de bloqueio em modo múltiplo, o qual é derivado do 2PL (ESWARAN et al., 1976). Todavia, diferente da especificação original do 2PL, onde há dois tipos de bloqueio, nomeadamente o compartilhado – utilizado para operações de leitura –, e o exclusivo – usado para operações de escrita –, o esquema de bloqueios múltiplos utilizado no MVCC introduz um novo tipo de bloqueio denominado bloqueio de certificação. Neste caso, um bloqueio pode ser adquirido sobre um item de dados para fins de leitura (compartilhado – SL), escrita (exclusivo – XL) ou para certificação (CL).

Na literatura são encontradas duas especificações para o MVCC baseado em bloqueios, são elas: o 2V2PL (*Two Version Two Phase Locking*) (STEARNS; ROSENKRANTZ, 1981; BERNSTEIN; HADZILACOS; GOODMAN, 1987); e o MV2PL (*MultiVersion Two Phase Locking*) (DUBOURDIEUX, 1982; BERNSTEIN; GOODMAN, 1983). Embora as especificações deles sejam bastante similares, já que ambas foram propostas no intuito de reduzir bloqueios em decorrência de conflitos de leitura/escrita, a principal diferença entre eles reside no fato de que o 2V2PL permite reter no máximo duas versões de um item de dados, enquanto que o MV2PL permite reter múltiplas versões de um mesmo item de dados. Note que na especificação padrão do 2PL, não é permitido que nenhuma transação acesse um item de dados que esteja bloqueado por um bloqueio exclusivo. No caso do 2V2PL, estas regras são relaxadas de modo que conflitos de leitura/escrita (i.e., SL/XL) são

eliminados, porém, permanece a regra para conflitos de escrita/escrita (i.e., XL/XL). O Quadro 2 apresenta a matriz de compatibilidade para o 2V2PL.

Quadro 2 – Compatibilidade entre os tipos de bloqueio no 2V2PL.

transação T_i / **transação** T_j	$SL(d_k)$	$XL(d_k)$	$CL(d_k)$
$SL(d_k)$	`sim`	`sim`	`não`
$XL(d_k)$	`sim`	`não`	`não`
$CL(d_k)$	`não`	`não`	`não`

A ideia consiste em permitir que outras transações T_j possam ler um item de dados d_k, que esteja sendo escrito por uma única transação T_i. Isso ocorre através da possibilidade de se manter duas versões para cada item de dados d_k. Para tanto, uma versão d_{k1} sempre é criada por alguma transação já validada, enquanto que a segunda versão d_{k2} é criada quando uma transação T_i adquire um bloqueio exclusivo sobre d_k. Assim, outras transações T_j podem realizar operações de leitura sobre a versão validada e estável de d_k, ou seja, sobre d_{k1}, enquanto T_i mantém um bloqueio exclusivo para uma operação de escrita sobre d_k. Com isso, T_i pode escrever um novo valor que resultara na versão d_{k2}, sem afetar o valor da versão estável e validada de d_{k1}. Todavia, quanto T_i estiver apta a validar, ela terá de obter um bloqueio de certificação para cada item de dados sobre os quais detém a posse de bloqueios exclusivos.

Como o bloqueio de certificação não é compatível com nenhum outro tipo de bloqueio (cfm. Quadro 2), T_i pode ter que aguardar para ser validada até que todos os itens de dados em bloqueio exclusivo sejam liberados por outras transações T_j que detenham a posse de bloqueios compartilhados para os mesmos itens de dados, a fim de que os bloqueios de certificação possam ser obtidos. Uma vez que o bloqueio de certificação é adquirido, o valor escrito em d_{k2} é atribuído como versão validada de d_k e d_{k2} é descartado (i.e., $d_{k1} \coloneqq d_{k2}$ e $d_{k2} \coloneqq \perp$), e em seguida os bloqueios de certificação são liberados. O grande feito deste esquema é que leituras podem ser executadas simultaneamente com uma única operação de escrita, sendo que a única implicação é que uma transação em validação tem de esperar até poder obter os bloqueios de certificação para todos os itens de dados sobre os quais realizou escritas.

Por outro lado, o MV2PL remove todos os conflitos existentes entre operações de leitura e de escrita, isto é, SL/XL e XL/XL. Por

conseguinte, esta característica permite que sejam mantidas múltiplas versões para um mesmo item de dados, a despeito das duas permitidas no 2V2PL. Estas versões, que são escritas pelas transações ativas, são denominadas "versões não-certificadas". No entanto, para preservar a serialização, transações podem ler apenas a versão certificada mais recente de um item de dados. Para tanto, tal como ocorre no 2V2PL, o MV2PL também dispõe de um bloqueio de certificação, o qual é usado para atrasar a validação de uma transação, até que não haja leituras ativas sobre os itens de dados que serão atualizados. O Quadro 3 apresenta a matriz de compatibilidade para os bloqueios disponíveis no MV2PL.

Quadro 3 – Compatibilidade entre os tipos de bloqueio no MV2PL.

transação T_i / **transação** T_j	$SL(d_k)$	$XL(d_k)$	$CL(d_k)$
$SL(d_k)$	`sim`	`sim`	`não`
$XL(d_k)$	`sim`	`sim`	`sim`
$CL(d_k)$	`não`	`sim`	`não`

A especificação do MV2PL visa prover um maior grau de concorrência, pela permissão múltiplas operações de escrita sobre um mesmo item de dados. Neste caso, uma vez que uma operação de escrita sobre um item de dados d_k produz uma nova versão d_{kj} para aquele item de dados, operações de escrita concorrentes sobre d_k não induzem a situação de conflito. Da mesma maneira, como os bloqueios de escrita e de leitura são compatíveis uns com os outros (cfm. Quadro 3), um item de dados pode ser mantido por bloqueios compartilhados e bloqueios exclusivos ao mesmo tempo. Para tanto, o MV2PL diferencia transações de somente-leitura de transações de atualização.

No caso, transações de atualização são conduzidas em consonância com o 2PL rigoroso, de modo que seja possível estabelecer uma ordem de validação e assim prover a serialização. Cada versão de um item de dados d_k tem uma marca de tempo única, que na realidade consiste num contador (p. ex.: *ts_counter*) que é incrementado durante o processamento da validação. Assim, quando uma transação de atualização necessita efetuar uma operação de leitura sobre um item de dados d_k, ela obtém um bloqueio compartilhado sobre aquele item de dados e lê o valor correspondente à versão mais atual do dado. Em se tratando de uma operação de escrita sobre um item d_k, antes de tudo a transação obtém um bloqueio exclusivo sobre d_k e então cria uma nova versão, cuja marca de tempo desta nova versão é inicialmente definida

como ∞ (i.e., $d_{k\infty}$) – que denota o maior valor entre todas as marcas de tempo possíveis. Se a transação de atualização desejar validar suas operações, primeiramente os bloqueios de escrita são convertidos em bloqueios de certificação – note que os bloqueios de certificação não são compatíveis com nenhum tipo de bloqueio. A partir daí, a transação atribui como marca de tempo de cada versão por ela criada como o valor $ts\_counter + 1$, e em seguida incrementa o valor de $ts\_counter$.

Uma vez que o esquema permite que apenas uma única transação seja validada por vez, isso resulta que uma transação sempre lê a versão mais recente criada por uma transação validada. Para tanto, cada transação de somente-leitura, quando iniciada, recebe uma marca de tempo, que é atribuída de acordo com valor atual de $ts\_counter$. No caso, elas seguem a mesma dinâmica de funcionamento do esquema MVCC baseado em ordenação por marca de tempo, no que diz respeito as operações de leitura.

2.1.3.2 Serialização em Bancos de Dados Replicados

Um banco de dados replicado é em sua essência um banco de dados distribuído, no qual diversas cópias dos itens de dados são armazenadas em múltiplos locais (ou sítios) (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Neste sentido, é desejável que o comportamento de um banco de dados replicado seja igual ao de um banco de dados não-replicado, no que concerne ao histórico do processamento de transações. Para tanto, deve-se assegurar a consistência dos dados, de tal maneira que a execução concorrente de um conjunto de transações sobre um ambiente de banco de dados replicado seja equivalente a uma execução serial sobre o mesmo banco de dados não replicado. Portanto, o primeiro aspecto que deve ser considerado em se tratando de serialização em um ambiente replicado é a distinção entre operações lógicas da transação – aquelas realizadas em nível de aplicação –, e operações físicas executadas em cada uma das réplicas (BERNSTEIN; GOODMAN, 1985). Isto implica que, apesar de um protocolo de controle de concorrência assegurar a serialização sobre as operações físicas executadas em cada uma das réplicas, não necessariamente assegura a serialização em nível de operações lógicas, e, portanto, da transação no ambiente replicado (ÖZSU; VALDURIEZ, 2011).

À vista disso, o conceito de serialização visto na Seção 2.1.3.1 deve ser reformulado, a fim de permitir a correção de transações em bancos de dados replicados, no que diz respeito às operações realizadas

sobre itens de dados lógicos. É importante ressaltar que a execução de transações (i.e., o histórico) sobre itens de dados replicados deve ser equivalente a alguma execução sequencial destas transações sobre dados não-replicados. O formalismo que permite estabelecer uma correlação entre a serialização de transações sobre bancos de dados não replicados, com transações em ambientes replicados é denominada *one-copy serializability* (1SR) (ATTAR; BERNSTEIN; GOODMAN, 1984; BERNSTEIN; GOODMAN, 1985). Todavia, esta reformulação de serialização estabelece que, para atender a condição 1SR o histórico de transações realizadas sobre dados replicados $\mathcal{H}$ deve conter apenas as transações que foram validadas, isto é, despreza-se todas aquelas que foram anuladas. Isto quer dizer que, dado um histórico $\mathcal{H}$, se $w_i[d_i, v]$ for o último valor escrito no item de dados d_i em $\mathcal{H}$, então $a_i \notin \mathcal{H}$ e $\forall w_j[d_i, v] \in \mathcal{H}$, ou $a_j \in \mathcal{H}$ ou $w_j[d_i, v] <_H w_i[d_i, v]$.

Nos termos da literatura, para que dois históricos $\mathcal{H}$ e $\mathcal{H}'$, mapeados em função de $\mathcal{T}$ e construídos respectivamente pelos sítios s e s' sejam equivalentes (i.e., $\mathcal{H} \equiv \mathcal{H}'$), as seguintes condições devem ser satisfeitas (ATTAR; BERNSTEIN; GOODMAN, 1984; BERNSTEIN; HADZILACOS; GOODMAN, 1987): (*i*) os valores iniciais lidos em ambos os históricos devem ser os mesmos; (*ii*) $\mathcal{H}$ e $\mathcal{H}'$ devem preservar a mesma relação *reads-from*, e; (*iii*) os valores finais escritos em cada um dos históricos devem ser os mesmos. De um modo geral, o 1SR é o critério de consistência mais rigoroso a ser seguido por um conjunto de transações executadas sobre um banco de dados replicado, tal como ocorre com a serialização em bancos de dados não-replicados. A exemplo da serialização, o 1SR estabelece que a execução de transações simultâneas sobre diferentes réplicas em um ambiente de banco de dados replicado deve ter o mesmo efeito que teria, caso as transações fossem executadas em alguma ordem sequencial em uma única réplica do ambiente. Tipicamente, a serialização em nível de réplica de banco de dados é obtida a partir do uso do protocolo 2PL rigoroso (vide Seção 2.1.3), no qual bloqueios são adquiridos sobre os itens de dados da transação, durante a execução local da mesma na(s) réplica(s).

2.1.4 Problemas Relacionados à Concorrência

Uma das características mais importantes para o processamento de transações é o atendimento às propriedades ACID (vide Seção 2.1.2). Um sistema de processamento de transações que permite que diversas transações sejam executadas simultaneamente, tem que estar prepa-

rado para situações onde o acesso a um item de dados (ou recurso) é requisitado por várias transações ao mesmo tempo. Nestes cenários é comum que operações intercaladas tentem obter acesso concomitante sobre um mesmo item de dados, e, caso alguma destas operações seja uma escrita, elas são caracterizadas como operações conflitantes. Esta seção demonstra os principais problemas relacionados à concorrência, verificados no processamento de transações.

2.1.4.1 Fenômenos e Anomalias de Concorrência

Conforme já mencionado, a execução simultânea de transações em conflito, quando não devidamente coordenada, acarreta em problemas que podem levar o sistema a um resultado inconsistente, e que portanto, deve ser evitado. A execução de operações conflitantes em diferentes transações pode resultar na ocorrência de fenômenos e anomalias de concorrência (ANSI, 1992; BERENSON et al., 1995). Os primeiros fenômenos de concorrência foram inicialmente identificados na literatura pelos proponentes do padrão ANSI SQL (ANSI, 1992). Posteriormente à definição do padrão ANSI SQL, estudos identificaram outros problemas relacionados à concorrência (BERENSON et al., 1995; ADYA, 1999; FEKETE; O'NEIL; O'NEIL, 2004). A partir dos resultados obtidos, os autores distinguiram os problemas de concorrência identificados como fenômenos, os quais podem resultar em erros (denotados por F); e anomalias, as quais sempre resultam em erros (denotadas por A), ambos não cobertos anteriormente pelo padrão ANSI SQL. Uma breve explicação acerca dos fenômenos e anomalias existentes na literatura é realizada no que segue.

F1 – Escrita Suja (*Dirty Write*): ocorre quando duas transações realizam escritas concomitantes sobre um mesmo item de dados. Suponha a existência de duas transações T_1 e T_2. Este fenômeno ocorre quando T_1 escreve em um item de dados x e T_2 também escreve neste mesmo item de dados, mas antes mesmo de T_1 validar ou abortar a transação. Assim, se T_1 ou mesmo T_2 vier a abortar, não é possível determinar qual seria o valor correto a ser preservado para o item de dados x. A Figura 5, adaptada de (BERENSON et al., 1995), ilustra a ocorrência do fenômeno da escrita suja.

Pelo exemplo da Figura 5 é possível verificar que o fenômeno da **escrita suja** pode incorrer na violação da consistência e/ou integridade dos dados. Considere que exista uma restrição de integridade entre os

Tempo	T_1	T_2
1	$begin(T_1)$	
2		$begin(T_2)$
3	$w_1[x, 5]$	
4		$w_2[x, 10]$
5		$w_2[y, 10]$
6		$commit(T_2)$
7	$w_1[y, 5]$	
8	$commit(T_1)$	

Figura 5 – O fenômeno *dirty write*.

itens de dados x e y, de tal maneira que em todos os casos os valores de x e de y devem ser sempre iguais (p. ex.: $x = y$). Evidente que, quando executadas sozinhas, tanto T_1 como T_2 mantém a consistência em face à tal restrição. Todavia, o exemplo da Figura 5 viola a restrição em questão, pois ao término da execução o valor de x será 10, enquanto que o de y será 5, e portanto, sendo $x \neq y$.

F2 – Leitura Suja (*Dirty Read*): ocorre quando há duas ou mais transações em execução, na qual T_1 está escrevendo em um item de dados x e uma segunda transação T_2 realizar uma leitura sobre o mesmo item de dados x. No caso, T_2 lê um valor ainda não válido (*uncommited*) para x, e portanto, é considerado "sujo". Um exemplo clássico deste fenômeno é apresentado na Figura 6.

Tempo	T_1	T_2
1	$begin(T_1)$	
2	$v \leftarrow r_1[x]$	
3		$begin(T_2)$
4	$w_1[x, v + 10]$	
5		$v \leftarrow r_2[x]$
6	$abort(T_1)$	
7		$w_2[x, v * 1.10]$
8		$commit(T_2)$

Figura 6 – O fenômeno *dirty read*.

Suponha no caso ilustrado na Figura 6, que o item de dados x mantém o saldo de uma conta, tendo como valor inicial 100. Na transação T_1 o cliente portador da conta x realiza um depósito de 10

unidades monetárias. Na transação T_2, ao mesmo tempo o gerente da conta realiza a correção do saldo em 10 por cento, devido a uma aplicação financeira contratada pelo cliente. Se porventura a transação T_1 precisar ser desfeita, o saldo final da conta será incorreto, isto é, será 121, quando deveria ser 110.

F3 – Leitura Não-Repetível (*Non-Repeatable Read*): este fenômeno ocorre quando uma transação lê um mesmo item de dados mais de uma vez, e cada operação de leitura retorna valores diferentes. O problema coberto por este fenômeno é, em sua essência, similar ao que ocorre na leitura suja, porém, a diferença é que os valores lidos pelas operações posteriores à primeira leitura são valores válidos (p. ex.: já validados) escritos por outras transações.

Tempo	T_1	T_2
1	$begin(T_1)$	
2		$begin(T_2)$
3	$v \leftarrow r_1[x]$	
4		$w_2[x, 33]$
5		$commit(T_2)$
6	$v \leftarrow r_1[x]$	
7	$commit(T_1)$	

Figura 7 – O fenômeno *non-repeatable read*.

Para uma melhor compreensão acerca do problema ilustrado pela Figura 7, considere 10 como o valor inicial para o item de dados x, isto é, antes do início de T_1 e T_2. No caso, ao T_1 ser iniciada, a operação de leitura realizada no instante 3 retornará o valor 10. Mas neste meio tempo, a transação T_2 escreve neste mesmo item de dados, e atualiza seu valor para 33 e valida a transação (instantes 4 e 5). Porém, em seguida T_1 lê novamente o item de dados x e a operação retorna um valor diferente daquele lido inicialmente, isto é, o 33 escrito por T_2. No caso ilustrado pela figura, o fenômeno não causa nenhum efeito colateral em T_1, porque ela não realiza nenhuma operação a partir dos valores lidos. No entanto, se o fizesse, a consistência dos dados seria violada e o resultado de operações posteriores seria incorreto.

F4 – Atualização Perdida (*Lost Update*): ocorre quando uma transação T_2 escreve (ou atualiza) um valor de um determinado item de dados x, mas esta atualização/escrita é perdida devido à outra operação de escrita/atualização que está a ser efetuada por uma transação T_1,

de maneira intercalada.

Tempo	T_1	T_2
1	$begin(T_1)$	
2		$begin(T_2)$
3	$v \leftarrow r_1[x]$	
4		$v \leftarrow r_2[x]$
5		$w_2[x, v + 50]$
6		$commit(T_2)$
7	$w_1[x, v + 100]$	
8	$commit(T_1)$	

Figura 8 – O fenômeno *lost update*.

O problema ilustrado pela Figura 8 é que, mesmo que a transação T_2 tenha sido validada, as atualizações por ela efetuadas são perdidas. Neste caso, o valor final de x conterá apenas o incremento em 100, o qual foi efetuado pela transação T_1.

F5/A1 – Leitura Fantasma (*Phantom Read*): a leitura fantasma é definida como sendo um fenômeno pelo ANSI SQL (ANSI, 1992), mas Berenson *et al.* (1995) afirma que numa interpretação mais rigorosa ela é considerada uma anomalia. Este fenômeno/anomalia é definido em termos da avaliação de uma condição (ou um predicado), e ocorre quando um item de dados x que atende àquela condição numa transação T_1 é excluído ou incluído por uma transação T_2 concorrente. No primeiro caso, a leitura em T_1 retorna o item de dados x, que satisfaz a uma dada condição, mas x deixa de existir em leituras posteriores baseadas na mesma condição. No segundo caso, como uma operação de inclusão é realizada por T_2, as operações de leitura posteriores baseadas na mesma condição passam a contar com um novo item de dados que não estava presente na primeira operação de leitura, que foi realizada a partir da condição especificada.

No cenário ilustrado pela Figura 9, a transação T_1 recupera da base de dados todos os itens de dados x_i tal que o valor deste item seja igual ou superior a 1000, e soma-os (instante 2). Em seguida, a transação T_2 escreve na base de dados um novo item de dados que atende à mesma condição especificada em T_1 (instante 4). Porém, por alguma razão T_1 re-executa a operação de leitura baseada na condição inicial (instante 6) e, nesta execução o resultado do somatório é diferente da primeira execução, sem que T_1 tenha realizado qualquer alteração na base de dados. Neste caso, é dito que algum item de dados fantasma

Tempo	T_1	T_2
1	$begin(T_1)$	
2	$\forall x_i \in \mathcal{D}, 1 \leq i \leq n \mid r_1[x_i] \geq 1000 : \sum_{i=1}^{n} x_i$	
3		$begin(T_2)$
4		$w_2[x, 1026]$
5		$commit(T_2)$
6	$\forall x_i \in \mathcal{D}, 1 \leq i \leq n \mid r_1[x_i] \geq 1000 : \sum_{i=1}^{n} x_i$	
7	$commit(T_1)$	

Figura 9 – A anomalia/fenômeno *phantom read*.

foi lido por T_1.

A2 – Leitura Oblíqua (*Skewed Read*): esta anomalia é caracterizada quando uma transação T_1 lê algum item de dados x_i e, de maneira intercalada, uma transação concorrente T_2 escreve sobre os itens de dados x_i e x_j (onde $i \neq j$) e valida. Em seguida T_1 lê o item de dados x_j e vê o valor atribuído por T_2. Neste caso, T_1 obteve o valor de x_i antes da execução de T_2 e o valor de x_j após a execução de T_2, conforme é ilustrado na Figura 10.

Tempo	T_1	T_2
1	$begin(T_1)$	
2		$begin(T_2)$
3	$r_1[x_i]$	
4		$v_i \leftarrow r_2[x_i]$
5		$v_j \leftarrow r_2[x_j]$
6		$w_2[x_i, 0]$
7		$w_2[x_j, v_j + v_i]$
8		$commit(T_2)$
9	$r_1[x_j]$	
10	$commit(T_1)$	

Figura 10 – A anomalia *read skew*.

Considere no exemplo da Figura 10, que os valores iniciais para os itens de dados x_i e x_j são 100 e 50, respectivamente. Ambas as transações T_1 e T_2 leem valores corretos para x_i (instantes 3 e 4), isto é, 100. Todavia, em se tratando de x_j apenas T_2 lê o valor correto 50, já que T_1 realiza a leitura de x_j após a validação de T_2, e portanto, obtém o valor 150 (instante 9), o qual reflete a escrita efetuada por T_2 (instante 7) devido à intercalação das operações.

A3 – Escrita Oblíqua (*Skewed Write*): esta anomalia é caracterizada quando há restrições de integridade entre itens de dados de uma base de dados. Tomemos como exemplo uma restrição entre os itens de dados x_i e x_j tal que $i \neq j$, onde tal restrição requer que o valor resultante da soma de ambos deve ser sempre positivo. A anomalia da escrita oblíqua ocorre de tal maneira que a restrição é sempre preservada pelas transações em suas execuções, mas em situação de concorrência ela é violada sem ser percebida pelas transações. Considere o escalonamento da Figura 11.

Tempo	T_1	T_2
1	$begin(T_1)$	
2		$begin(T_2)$
3	$v_i \leftarrow r_1[x_i]$	
4	$v_j \leftarrow r_1[x_j]$	
5		$v_j \leftarrow r_2[x_j]$
6		$v_i \leftarrow r_2[x_i]$
7	$w_1[x_j, v_j - v_i]$	
8		$w_2[x_i, v_i - v_j]$
9		$commit(T_2)$
10	$commit(T_1)$	

Figura 11 – A anomalia *write skew*.

Para o escalonamento ilustrado na Figura 11, suponha que os valores dos itens de dados x_i e x_j representam os saldos das contas corrente e poupança de um mesmo cliente, respectivamente. Considere também, que a política da instituição que mantém as contas é que a soma dos saldos sempre deve ser positivo, e portanto, $x_i + x_j > 0$. Assumindo que os valores iniciais para x_i e x_j sejam 110 e 100, respectivamente. Se transação T_1 obtém os valores de x_i e x_j e subtrai de x_j o valor de x_i, a restrição é mantida, pois $x_j = 100 - 110 = -10$ (instante 7) e $x_i + x_j > 0 \Rightarrow 110 + (-10) = 100$. Da mesma maneira, T_2 simultaneamente obtém os valores de x_j e x_i, mas subtrai de x_i o valor de x_j de modo que a restrição ainda é mantida, já que $x_i = 110 - 100 = 10$ (instante 8) e $x_i + x_j > 0 \Rightarrow 10 + 100 = 110$. Por outro lado, ao término das execuções de T_1 e T_2 a restrição não é mantida, pois $x_i = 10$ e $x_j = -10$, e $x_i + x_j \Rightarrow 10 + (-10) = 0$.

A4 – Anomalia Somente em Leitura (*Read-only Anomaly*): esta anomalia ocorre apenas em transações compostas somente por ope-

rações de leitura. Sejam x_i e x_j (tal que $i \neq j$) dois itens de dados em uma base de dados, cujo valor inicial é 0 para ambos. A anomalia ocorre quando duas transações T_1 e T_2 realizam simultaneamente, operações sobre estes itens de dados, e uma transação T_3 apenas de leitura recupera os valores dos itens de dados x_i e x_j, a exemplo do escalonamento da Figura 12, adaptada de (FEKETE; O'NEIL; O'NEIL, 2004).

Tempo	T_1	T_2	T_3
1	$begin(T_1)$		
2		$begin(T_2)$	
3		$v_i \leftarrow r_2[x_i]$	
4		$v_j \leftarrow r_2[x_j]$	
5	$v_j \leftarrow r_1[x_j]$		
6	$w_1[x_j, 20]$		
7	$commit(T_1)$		
8			$begin(T_3)$
9			$v_i \leftarrow r_2[x_i]$
10			$v_j \leftarrow r_2[x_j]$
11			$commit(T_3)$
12		$w_2[x_i, -11]$	
13		$commit(T_2)$	

Figura 12 – A anomalia *read-only*.

Para verificar a anomalia em questão, considere que existe uma restrição integridade entre os itens de dados x_i e x_j, tal que $x_i + x_j > 0$ e que, caso a restrição não seja cumprida, a operação que a descumprir é penalizada com um decremento em uma unidade no valor do item de dados fornecido naquela operação. No escalonamento da Figura 12, a transação T_1 obtém o valor inicial de x_i (p. ex.: 0) e altera o valor do mesmo para 20 unidades e valida tal alteração (instantes 5, 6 e 7, respectivamente). Por outro lado, a transação T_2 que já havia lido o valor inicial 0 tanto para x_i como para x_j (instantes 3 e 4), faz uma alteração em -10 unidades em x_j e, como a restrição não é cumprida (i.e., $x_i + x_j > 0$) uma unidade é decrementada como penalidade, resultando no valor final de -11 para x_j. E, finalmente, T_3, que é uma transação apenas de leitura, recupera o valor 0 para x_i e 20 para x_j, quando na realidade os valores finais para x_i e x_j são 20 e -11, respectivamente. Além disso, T_3 visualizou uma mistura de dados do passado e do futuro.

2.1.4.2 Impasses (*Deadlocks*)

Um problema não menos importante do que as anomalias e fenômenos apresentadas nesta seção é o impasse, popularmente conhecido como *deadlock* (OBERMARCK, 1982). O impasse consiste em uma situação em particular, na qual duas ou mais transações estão competindo por um bloqueio não compatível (vide Seção 2.1.3.1) sobre um mesmo item de dados, onde alguma(s) delas é(são) bloqueada(s) e deve(m) aguardar pela liberação do(s) bloqueio(s) por parte das demais. Por vezes, pode ocorrer que algumas transações estejam bloqueadas à espera da liberação de um bloqueio, o qual está sendo mantido por uma transação que também está bloqueada, formando assim, um ciclo de espera. A Figura 13 ilustra tal situação num cenário com três transações.

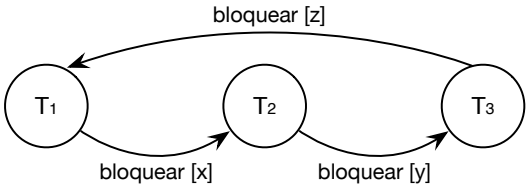


Figura 13 – Cenário com situação de impasse.

No cenário ilustrado pela Figura 13, a transação T_1 está a aguardar a liberação do bloqueio de x por parte de T_2, T_2 está a espera da liberação do bloqueio de y por T_3 e, por fim, T_3 está bloqueada à espera da liberação do bloqueio de z por T_1. Como nenhuma das transações tem condições de fazer progresso, elas estão em situação de impasse e, caso não haja intervenção externa (p. ex.: do sistema), elas ficarão bloqueadas indefinidamente. Não obstante, T_1, T_2 ou T_3 deve ser abortada para que seja eliminado o ciclo de espera.

Uma solução bastante usada para resolver impasses se baseia no algoritmo proposto por Ron Obermarck (OBERMARCK, 1982), cuja implementação inicial se deu no âmbito do sistema R* (MOHAN; LINDSAY; OBERMARCK, 1986). Neste algoritmo, um impasse (*deadlock*) é resolvido por meio da anulação de uma ou mais transações denominadas "vítimas", em que uma "vítima" é escolhida de acordo com critérios que visam minimizar os custos relacionados à anulação, como por exemplo, a transação mais nova dentre as que estão em situação de impasse.

2.1.4.3 Inanição (*Starvation*)

Outro problema característico em sistemas computacionais, e particularmente no âmbito de transações é a inanição, também conhecido por *starvation* (CONNOLLY; BEGG, 2005; SILBERSCHATZ; KORTH; SUDARSHAN, 2011). A inanição ocorre quando alguma transação é postergada indefinidamente, sem obter êxito na aquisição de bloqueios, tampouco em sua terminação. Um cenário de inanição é aquele em que uma mesma transação é sempre escolhida como vítima e nunca consegue ser executada por completo. Além da ocorrência em bancos de dados que implementam o controle de concorrência baseado em bloqueios, a inanição é bastante recorrente em bancos de dados que suportam o controle de concorrência otimista (KUNG; ROBINSON, 1981).

Uma vez que o controle de concorrência otimista recorre à anulação de transações como forma de assegurar a consistência serializável de uma base de dados, transações que não puderem ser validadas em decorrência de conflitos podem ser reiniciadas repetidas vezes. No pior caso, uma mesma transação pode ser "vítima' de sucessivas anulações e nunca consegue ser processada com êxito, o que portanto, deve ser evitado no processamento de transações em ambientes reais.

2.1.5 Isolamento de Transações

O isolamento é um aspecto de particular interesse desta tese, pois ele permite assegurar que transações concorrentes sejam isoladas umas das outras, sempre que necessário, para prover garantias de consistência das ações realizadas pelas transações sobre o banco de dados. Em termos da corretude de transações, o isolamento (**I** do ACID) é a propriedade que define que uma transação em execução deve ser completamente isolada de outras transações em execução simultânea. Este isolamento visa assegurar que o efeito da execução de um conjunto de transações simultâneas seja o mesmo daquela execução onde as mesmas transações são executadas uma após a outra. O isolamento de transações pode ser realizado a partir de diferentes níveis, que definem o grau de consistência permitido durante a execução de transações (GRAY et al., 1976). O grau de consistência mais restritivo assegura a execução serial de transações, enquanto que outros menos restritivos permitem inconsistências. É importante mencionar que a combinação do grau de consistência com o nível de isolamento da transação carateriza o que é conhecido por critério de consistência.

Tipicamente, o isolamento é definido em termos de serialização (vide Seção 2.1.3). Embora a corretude de transações esteja intrinsecamente relacionada à execução sequencial destas, tal condição pode induzir um sistema de processamento de transações a uma perda significativa de desempenho, já que não se pode permitir nenhum tipo de concorrência. Neste sentido, ao longo dos anos, propostas para melhorias em termos de concorrência no processamento de transações foram desenvolvidas. O trabalho seminal de Gray (GRAY; LORIE; PUTZOLU, 1975) foi o ponto de partida, onde o autor especificou "graus de consistência" como suporte para prover o isolamento de dados baseado em bloqueios. Mais tarde, um conjunto de pesquisadores da academia e da indústria estendeu a ideia de Gray (GRAY; LORIE; PUTZOLU, 1975), o que culminou na introdução do conceito de "níveis de isolamento" (ANSI, 1992).

Quadro 4 – Níveis de isolamento baseados no padrão ANSI SQL.

ISOLAMENTO	**FENÔMENO**		
	Leitura Suja	Leitura Repetível	Leitura Fantasma
Read Uncommited	P	P	P
Read Commited	NP	P	P
Repeatable Read	NP	NP	P
Serializable	NP	NP	NP

Os níveis de isolamento do padrão ANSI SQL (ANSI, 1992) são especificados a partir de diferentes semânticas de consistência de dados. Estas semânticas são definidas com base em possíveis fenômenos oriundos da concorrência, os quais caracterizam a execução incorreta de transações como resultado da ausência de isolamento e controle de concorrência apropriados. Não obstante, o ANSI SQL visa estabelecer um padrão para os níveis de isolamento para os SGBDs, a fim de que seja possível fornecer semânticas de consistência que sejam adequadas para diversos tipos aplicações, ao passo de que o projetista possa escolher o isolamento mais apropriado para sua aplicação. Um dos pontos destacados pelo padrão ANSI SQL é que os níveis de isolamento ali especificados são independentes de implementação. No ANSI SQL a caracterização dos níveis de isolamento se dá a partir de alguns dos fenômenos apresentados na Seção 2.1.4.1. Uma visão geral do relacionamento desses fenômenos com os níveis de isolamento é apresentada

no Quadro 4, onde os indicadores NP e P significam **N**ão **P**ossível e **P**ossível, respectivamente.

Posteriormente à definição do ANSI SQL, alguns pesquisadores analisaram cuidadosamente as especificações acerca dos níveis de isolamento definidos pelo padrão ANSI SQL, e na ocasião obtiveram resultados que demonstraram que aquelas definições são ambíguas e somente são implementáveis a partir do uso de técnicas baseadas em bloqueios (BERENSON et al., 1995; ADYA, 1999; FEKETE; O'NEIL; O'NEIL, 2004). E por intermédio da ocasião, Berenson *et al.* (1995) propuseram um novo nível de isolamento denominado *Snapshot Isolation* (SI), que é intermediário entre o rigorosamente serializável e os demais definidos pelo ANSI SQL. O SI é uma abordagem multiversão para o controle de concorrência bastante eficiente; porém, ele é mais fraco quando comparado ao controle serializável e por isso não assegura a serialização de transações concorrentes. A despeito do SI não apresentar as anomalias verificadas pelo ANSI SQL, ele é suscetível a outros comportamentos anômalos (BERENSON et al., 1995; FEKETE; O'NEIL; O'NEIL, 2004). Os níveis de isolamento sugeridos por Berenson *et al.* (1995) são apresentados no Quadro 5.

Na atualidade, os níveis de isolamento descritos no Quadro 5 são suportados pela maioria dos SGBDs comerciais e acadêmicos existentes, embora as implementações destes sejam todas baseadas em bloqueios (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007). A partir do Quadro 5 pode-se verificar quais fenômenos e anomalias são evitado(a)s ou permitido(a)s em cada um dos respectivos níveis de isolamento. No caso, o nível de isolamento mais forte é o serializável, enquanto que o mais fraco é aquele que permite leituras sujas. É evidente que níveis de isolamento mais fracos são menos restritivos e oferecem melhor desempenho, porém, em detrimento do relaxamento da consistência, o que pode incorrer na corrupção e/ou violação da integridade dos dados (WEIKUM; VOSSEN, 2002; RAMAKRISHNAN; GEHRKE, 2003).

Quadro 5 – Níveis de isolamento conforme Berenson *et al.* (1995).

ISOLAMENTO	**ANOMALIA**							
	Escrita Suja	Leitura Suja	Atualização Perdida	Leitura Não-repetível	Leitura Fantasma	Leitura Oblíqua	Escrita Oblíqua	Somente de Leitura
Read Uncommited	NP	P	P	P	P	P	P	P
Read Commited	NP	NP	P	P	P	P	P	P
Repeatable Read	NP	NP	NP	NP	P	NP	NP	NP
Snapshot	NP	NP	NP	NP	P	NP	P	P
Serializable	NP	NP	NP	NP	NP	NP	NP	NP

Em se tratando especificamente do SI, de um modo geral a literatura aponta que ele é suscetível apenas aos fenômenos/anomalias da escrita oblíqua e de somente de leitura (BERENSON et al., 1995; ADYA, 1999; ORACLE, 2014b; MICROSOFT, 2014) (cfm. Quadro 5). Todavia, num passado pouco recente Alan Fekete *et al.* (2005) verificaram que o SI é suscetível também à anomalia da *leitura fantasma*. Os autores demonstraram que apesar de o SI evitar os casos mais tradicionais nos quais ocorrem as leituras fantasmas (ESWARAN et al., 1976), alguns problemas podem ocorrer de forma análoga à anomalia da escrita oblíqua (i.e., *write skew*). Este caso particular é denominado escrita oblíqua baseada em predicados (i.e., *predicate-based write skew*) (FEKETE et al., 2005).

A despeito dos problemas apontados para o SI, este tem ganhado popularidade no âmbito da indústria de software, mais precisamente naquelas empresas que desenvolvem SGBDs. Tal afirmação é justificada pela observação de que muitos SGBDs comerciais têm adotado o SI como nível de isolamento mais forte (ORACLE, 2014b; POSTGRESQL, 2014; MICROSOFT, 2014), o que implica que em determinadas circunstâncias, transações não possam obter uma serialização completa (BERENSON et al., 1995; FEKETE et al., 2005). A literatura demonstra que a única maneira de se obter execuções serializáveis através do SI, é por meio de verificações e ações realizadas em nível de concepção do sistema (FEKETE, 2005), o que pode soar contraproducente em relação ao que é estabelecido pelo modelo de transações.

2.2 CONCEITOS EM SISTEMAS DISTRIBUÍDOS

Esta seção apresenta alguns conceitos básicos relacionados aos sistemas de computação distribuída, ou simplesmente sistemas distribuídos. Os conceitos são apresentados e discutidos a partir da problemática encontrada na concepção destes sistemas, principalmente no que diz respeito ao acordo. Esta breve caracterização se faz necessária, a fim de tornar possível a compreensão em torno dos termos utilizados no âmbito do desenvolvimento desta tese.

2.2.1 Caracterização de Sistemas Distribuídos

A literatura apresenta diversas caracterizações para sistemas distribuídos, as quais são definidas de acordo com algumas perspectivas

em cada uma das obras (JALOTE, 1994; LYNCH, 1996; ATTIYA; WELCH, 2004; KSHEMKALYANI; SINGHAL, 2008; COULOURIS et al., 2012). Da perspectiva de organização, um sistema distribuído pode ser caracterizado como um conjunto de unidades de processamento (ou elementos), independentes e espacialmente dispersas, que têm um objetivo comum de cooperar entre si para a execução de tarefas e resolução de problemas (KSHEMKALYANI; SINGHAL, 2008). Tipicamente, estas unidades de processamento estão interconectadas por meio de uma rede de comunicação e se comunicam através da troca de mensagens por meio de canais que ligam estas unidades umas às outras (COULOURIS et al., 2012). Além disso, cada unidade de processamento geralmente é constituída por elementos de computação, tais como processos, processadores, memória, etc. A Figura 14 ilustra um exemplo típico de sistema distribuído, onde há um servidor de aplicações e um conjunto de clientes executando estas aplicações em diferentes dispositivos.

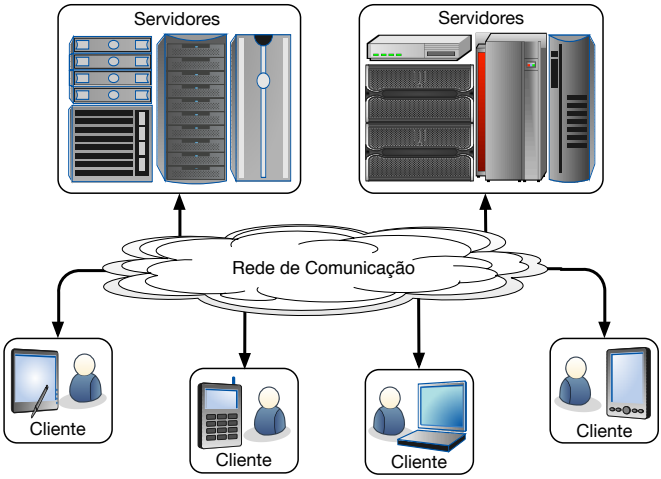


Figura 14 – Exemplo de um sistema distribuído.

Da mesma maneira como ocorre com a caracterização de sistemas distribuídos, a concepção de um sistema distribuído também pode ser realizada sob diferentes perspectivas. Estas perspectivas, por sua vez, estão intrinsecamente relacionadas com as aplicações para as quais se deseja atender os requisitos estabelecidos. Nesta tese, as perspectivas estão atreladas aos aspectos atinentes à computabilidade (KSHEMKALYANI; SINGHAL, 2008), e à arquitetura (TANENBAUM; STEEN, 2006)

de sistemas distribuídos. De computabilidade porque se visa obter modelos e protocolos capazes de resolver problemas de computação distribuída. Do ponto de vista arquitetural porque se pretende especificar uma arquitetura modular, de modo que esta possa ser empregada em nível subjacente, como suporte para a resolução de problemas de computação distribuída.

Um aspecto de grande relevância quanto a concepção de sistemas distribuídos é a interação (ou sincronização) entre os elementos que compõem, ou que irão compor o sistema. A importância da interação decorre do fato de que, como os elementos são independentes e realizam computações locais, ações de sincronização são imperiosas para que as tarefas a serem realizadas pelo sistema distribuído sejam precisamente coordenadas (p. ex. alocação de recursos, validação ou retrocesso de uma ação). Ademais, características comuns em sistemas de computação distribuída são (TANENBAUM; STEEN, 2006; KSHEMKALYANI; SINGHAL, 2008; COULOURIS et al., 2012): (*i*) o não compartilhamento de memória; (*ii*) a ausência de um relógio único de tempo global e; (*iii*) propensão a falhas de alguma natureza. Neste ensejo, alguns dos problemas relacionados aos sistemas distribuídos identificados na literatura são apresentados ao longo desta seção.

2.2.2 Ambiente de Computação Distribuída

Esta seção apresenta alguns dos conceitos elementares em sistemas distribuídos, os quais serão adotados ao longo desta tese. Neste sentido, são apresentadas algumas das abstrações utilizadas no desenvolvimento desta tese, bem como as hipóteses que amparam o modelo de sistema adotado.

2.2.2.1 Modelo de Sistema

Nesta tese, do ponto de vista de algorítmica distribuída o termo **sistema** é utilizado para descrever o ambiente de execução, enquanto que o **modelo** é usado para descrever as propriedades que se pode inferir no sistema. Para tanto, o modelo de sistema deve ser visto como a especificação formal do conjunto de hipóteses admitidas no contexto do sistema de computação distribuída, a fim de que seja possível assegurar que as propriedades inferidas não sejam violadas. Em relação a esta tese, um conjunto de submodelos é encapsulado dentro do modelo

de sistema, de modo que cada submodelo define as peculiaridades para cada um dos componentes que formam o sistema (p. ex. processos, canais de comunicação, tipos de falhas, etc.)

2.2.2.2 Processos

No contexto de sistemas distribuídos, um processo é um elemento lógico que tem por finalidade realizar as computações previstas para aquele sistema (CACHIN; GUERRAOUI; RODRIGUES, 2011), sendo tipicamente modelado a partir de um autômato determinista. Este autômato é composto por um conjunto de variáveis, de ações e de estados, que são especificados de acordo com o algoritmo que o autômato computará (ATTIYA; WELCH, 2004; LYNCH, 1996). As ações são definidas a partir de um conjunto de pré-condições e um conjunto de operações, de modo que uma ação só é executada se todas as pré-condições são verdadeiras.

As transições de estados deste autômato ocorrem a partir de estímulos, tais como a recepção ou o envio de uma mensagem. O estado global do sistema distribuído é composto pelo estado local de cada um dos processos e pelo estado dos canais de comunicação (CHANDY; LAMPORT, 1985), os quais também são conhecidos na literatura como *links* (CACHIN; GUERRAOUI; RODRIGUES, 2011). De outro modo, é dito que a inicialização de um sistema distribuído ocorre quando os processos se encontram em seus estados iniciais (possivelmente arbitrários) e os canais estão vazios (LYNCH, 1996).

Para o desenvolvimento desta tese, admite-se a existência de um universo de processos $\mathcal{U}$, de tal maneira que $\mathcal{U}$ pode ser dividido em alguns subconjuntos, de acordo com a natureza do problema abordado e da solução proposta. Todavia, independentemente desta premissa, a condição abaixo é sempre verificada:

- $\forall \mathcal{S}_i, \mathcal{S}_i \subset \mathcal{U} : |\bigcup_{i=1}^{n} \mathcal{S}_i| = |\mathcal{U}|$

Maiores detalhes acerca dos modelos e conjuntos de processos serão fornecidos nos capítulos subsequentes desta tese, por ocasião da apresentação de cada contribuição, em particular.

2.2.2.3 Canais de Comunicação

Conforme visto na Seção 2.2.1, uma computação distribuída ocorre pela interação entre os processos, o que se dá a partir de trocas de

mensagens entre eles. Não obstante, uma interação requer comunicação e coordenação entre os processos (COULOURIS et al., 2012). A formalização utilizada para denotar os canais de comunicação que ligam os processos entre si são os *links* (CACHIN; GUERRAOUI; RODRIGUES, 2011). Como os *links* são abstrações construídas sobre um meio de comunicação, eles estão sujeitos à ocorrência de eventos transientes e inerentes aos meios de comunicação, e estes por sua vez podem resultar em perdas, duplicação e corrupção das mensagens enviadas/recebidas naqueles *links* (CACHIN; GUERRAOUI; RODRIGUES, 2011). Em face à tal situação, na literatura são especificadas algumas abstrações de comunicação mais fortes, que permitem estabelecer limites nas perdas, ou até mesmo evitam a ocorrência de perdas. Neste sentido, a confiabilidade dos canais de comunicação pode ser inferida a partir de duas propriedades (CHARRON-BOST; DÉFAGO; SCHIPER, 2002):

- **Ausência de Perdas**: se um processo p envia uma mensagem m ao processo q e q é correto, então q finará por receber m;
- **Perda Justa**: se um processo p envia um número infinito de mensagens ao processo q e q é correto, então q receberá um número infinito de mensagens enviadas por p;

A despeito das propriedades citadas, a literatura dispõe de um tipo de canal com propriedades ainda mais rigorosas, as quais estipulam que os canais não criam, não alteram, nem duplicam as mensagens sobre o meio de comunicação. A estes canais que não admitem perdas é dado o nome de canais confiáveis (*reliable channels*) (BASU; CHARRON-BOST; TOUEG, 1996; DÉFAGO; SCHIPER; URBÁN, 2004), porém, restrições ao modelo apontam que este tipo de canal só é factível em sistemas que tenham capacidade infinita para o *buffer* de mensagens. Tal afirmação advém do fato de que *buffers* finitos são suscetíveis a transbordamento (*overflow*), o que, por conseguinte, pode incorrer em perdas. Em face a esta restrição, a literatura define uma abstração para modelar canais que admitem perdas, os *fair-lossy links* (BASU; CHARRON-BOST; TOUEG, 1996). A especificação dos *fair-lossy links* leva em consideração aspectos mais realistas, pois ela admite que falhas temporárias podem causar a perda de mensagens, por exemplo, devido à capacidade finita de armazenamento. Por outro lado, a especificação deste tipo de canal pressupõe que as perdas ocorrem de maneira equitativa.

Um aspecto de grande interesse é que abstrações mais fortes como canais confiáveis podem ser especificadas e implementadas sobre os *fair-lossy links*, por exemplo, a partir do uso de ferramentas

criptográficas para o provimento de autenticação, e mecanismos de reconhecimento e retransmissão, quando apropriado (CHARRON-BOST; DÉFAGO; SCHIPER, 2002; DÉFAGO; SCHIPER; URBÁN, 2004). A partir desta premissa, pode-se especificar canais que satisfaçam propriedades desejáveis, tais como **integridade**, **confiabilidade** e **ordenação**. Para tanto, sejam p e q dois processos, tal que $p \neq q$:

- **Integridade**: se p recebe uma mensagem m de q, então q enviou m a p;
- **Confiabilidade**: se p envia uma mensagem m a q, então q finará por receber m;
- **Ordem FIFO**: se q envia uma mensagem m a p antes de enviar uma mensagem m' também a p, então p recebe m antes de m'.

Canais que satisfazem estas propriedades são conhecidos como canais autenticados com ordem FIFO (DOUDOU; GARBINATO; GUERRAOUI, 2002). Uma aproximação pragmática bastante comum para este tipo de canal é o uso de conexões TCP em conjunto com SSL/TLS (DIERKS, 2008). É digno de nota que as especificações dos protocolos desenvolvidos no âmbito desta tese são baseadas neste tipo de canal.

2.2.2.4 Modelos de Falhas

Conforme visto na Seção 2.2.2.2, a execução de uma computação distribuída ocorre através de processos. Neste ensejo, pode-se dizer que um processo é **correto** se ele executa uma computação (p. ex.: um algoritmo) de acordo com sua especificação. De outro modo, é dito que um processo é **não-correto** (ou **faltoso**), se durante o decurso de sua execução é observado que seu comportamento se tornou diferente daquele especificado para a computação (JALOTE, 1994). Os desvios de especificação a que um processo está sujeito definem o que é conhecido na literatura por tipos de falhas (SCHNEIDER, 1993). Tipicamente, as faltas são especificadas a partir do comportamento para o qual o sistema é conduzido quando de sua ocorrência, e são agrupadas em classes ou modelos de faltas/falhas (JALOTE, 1994; GÄRTNER, 1999).

Na literatura são definidos diversos tipos de faltas (SCHNEIDER, 1993), dentre os quais, os extremos de acordo com o grau de severidade, isto é, a menos severa e a mais severa são as faltas de **parada** (ou *crash*) e as faltas **bizantinas** (ou *byzantine*) (LAMPORT; SHOSTAK; PEASE,

1982). As características dos desvios de acordo com os tipos de falhas citados são (SCHNEIDER, 1993):

- **Falta de Parada**: é caracterizada pela parada prematura e abrupta de um processo, de modo que o serviço fornecido para. É o tipo de falhas mais simples e menos severo, e mais fácil de ser detectado;

- **Falta Bizantina**: é caracterizada pelo comportamento arbitrário exibido por um processo, o qual engloba todos os tipos de falhas menos severas. As faltas bizantinas podem ser de origem benigna, isto é, causada por *bugs* de software, corrupção de dados, de memória, etc., ou de origem maliciosa, onde um processo faltoso pode tentar se passar por outro, enviar mensagens incorretas e espúrias, omitir o envio e/ou recepção das mensagens, produzir intencionalmente respostas incorretas às requisições, etc. É o tipo de falhas mais severo e mais difícil de ser tratada.

Nesta tese o interesse está centrado no desenvolvimento de soluções que toleram faltas Bizantinas, com a finalidade de prover confiabilidade no processamento de transações em bancos de dados. Embora os termos faltas e falhas estejam intimamente relacionados, há uma distinção entre eles. Uma falta (*fault*) é, do ponto de vista fenomenológico, o que causa um erro (*error*) em um nível mais baixo de abstração. E o erro por sua vez, é o que conduz o sistema a um estado de falha (*failure*) (JALOTE, 1994; GÄRTNER, 1999). Portanto, é dito que um sistema tolera **faltas** e não **falhas**. Tal afirmação decorre do fato de que uma vez que ocorre a falha, o sistema é desviado de sua especificação. Por outro lado, como a falta ocorre num estágio anterior à falha, a tolerância a faltas previne a ocorrência da falha e deste modo, previne desvios de especificação no sistema. Para fins de compreensão, uma definição bastante concisa sobre faltas e falhas bizantinas pode ser assumida como a seguinte (DRISCOLL et al., 2003):

- **Falta Bizantina**: uma falta que pode apresentar diferentes sintomas para diferentes observadores (p. ex.: entidades/componentes de um sistema);

- **Falha Bizantina**: a perda de um serviço e/ou corrupção de um sistema, devido à ocorrência de uma falta bizantina não tratada.

2.2.3 Acordo em Sistemas Distribuídos

Em um sistema de computação distribuída as entidades que compõem o sistema cooperam entre si para a execução de tarefas (KSHEMKALYANI; SINGHAL, 2008). No entanto, é imperioso que as entidades que visam objetivos em comum no âmbito do sistema distribuído possam compartilhar uma visão comum do estado do sistema (BARBORAK; DAHBURA; MALEK, 1993; CHARRON-BOST, 2001). Neste caso, a execução de tarefas por parte destas entidades requer algum tipo de coordenação, a fim de que elas sejam capazes de chegar a um denominador comum acerca da(s) ação(ões) a ser(em) realizada(s).

Da perspectiva de algorítmica distribuída, uma classe de problemas que constitui a base fundamental de sistemas distribuídos é o acordo (PEASE; SHOSTAK; LAMPORT, 1980; CHARRON-BOST, 2001). Na literatura são encontrados diversos problemas relacionados ao acordo em sistemas distribuídos (KSHEMKALYANI; SINGHAL, 2008; CACHIN; GUERRAOUI; RODRIGUES, 2011), onde a maioria destes se baseia na mesma premissa, isto é, todos os processos envolvidos naquela computação distribuída devem chegar a uma decisão unânime, cuja decisão dependerá da natureza do problema. De um modo geral, as especificações dos problemas de acordo visam dar condições para que os processos envolvidos numa computação distribuída cheguem a alguma decisão em comum sobre algum valor ou conjunto de valores (CACHIN; GUERRAOUI; RODRIGUES, 2011). Este acordo pode variar desde a simples troca de mensagens entre dois processos, onde ambos devem concordar com a sequência de mensagens que foram emitidas, ou até mesmo um comum acordo quanto à realização de uma tarefa do sistema, por exemplo, a validação ou a anulação de uma transação.

No restante desta seção são apresentados os problemas de acordo que formam a base para o desenvolvimento desta tese. São eles: (*i*) consenso; (*ii*) validação atômica não-bloqueante; (*iii*) difusão confiável; (*iv*) difusão com ordem total, e; (*v*) replicação de máquina de estados.

2.2.3.1 Consenso

O consenso (WENSLEY et al., 1978; PEASE; SHOSTAK; LAMPORT, 1980) é a generalização mais conhecida da classe de problemas de acordo em sistemas distribuídos. O problema do consenso consiste em assegurar que todos os processos corretos envolvidos em uma computação distribuída terminem por obter uma saída comum baseada nas mesmas

entradas, de modo que todos os processos devem decidir por algum valor (o mesmo) que tenha sido previamente proposto por algum destes processos. O consenso pode ser visto como um dos problemas mais estudados no âmbito de sistemas distribuídos, e sua importância é vista tanto em termos teóricos quanto práticos. Teóricas porque ele define os limites para os ambientes computacionais, nos quais o acordo pode ser implementado; e práticas porque constitui a base algorítmica para quase toda a classe de problemas de acordo.

O problema do consenso envolve um conjunto de processos $\mathcal{P} = \{p_1, p_2, ..., p_n\}$, onde cada processo p_i propõe um valor $v_i \in \mathcal{V}$, de modo que ao término da computação todos os processos decidem de forma irrevogável e unânime sobre algum valor contido no conjunto dos valores propostos V. O consenso é formalmente definido por duas primitivas (HADZILACOS; TOUEG, 1994):

- $propose(\mathcal{P}, v)$: denota que o valor v é proposto a um conjunto de processos $\mathcal{P}$;
- $decide(v)$: indica que o valor v é decidido.

Por sua vez, estas primitivas devem satisfazer as seguintes propriedades (RAYNAL; SINGHAL, 2001; CHARRON-BOST, 2001):

- **Acordo**: se um processo correto decide por um valor v, então todos os processos corretos decidem pelo mesmo valor v;
- **Validade**: se um processo decide por um valor $v \in V$, então v foi previamente proposto por algum processo;
- **Integridade**: um processo decide no máximo uma vez;
- **Terminação**: todos os processos corretos chegam ao valor de decisão.

Dentre as propriedades especificadas para o consenso, as três primeiras, isto é, o **acordo**, a **validade** e a **integridade** são concernentes à corretude do problema (*safety*), enquanto que a **terminação** especifica a condição de progressão (*liveness*). Na literatura é demonstrado que as propriedades definidas para o problema do acordo abrem precedentes para que os processos decidam valores diferentes caso um deles venha a falhar (RAYNAL; SINGHAL, 2001; CHARRON-BOST; SCHIPER, 2004). Neste caso, uma generalização denominada **consenso uniforme** (CHARRON-BOST; SCHIPER, 2004) define uma propriedade de **acordo** mais forte para o consenso. Esta propriedade é definida com (RAYNAL; SINGHAL, 2001; CHARRON-BOST; SCHIPER, 2004):

- **Acordo Uniforme**: dois processos, corretos ou faltosos, não decidem por valores diferentes.

Note que, no consenso uniforme a propriedade **acordo uniforme** não permite que dois processos (faltosos ou corretos) decidam de maneira diferente, o que por conseguinte, impõe a mesma decisão para qualquer processo que decide.

Em referência ao problema do consenso, um dos resultados teóricos mais importantes em termos de solubilidade foi apresentado há quase trinta anos por Fischer, Lynch e Paterson (FISCHER; LYNCH; PATERSON, 1985), em que eles demonstraram que não existe solução determinista que resolve o problema de consenso num sistema distribuído assíncrono propenso a uma simples falha. Esta impossibilidade, também conhecida na literatura como "impossibilidade FLP", decorre do fato de um processo falho ser indistinguível de um processo muito lento em um sistema assíncrono. Não obstante, desde sua obtenção, este resultado tem sido extremamente importante, haja vista que o consenso constitui a base para a resolução de muitos problemas práticos, tais como, filiação em grupo, eleição de líder, validação atômica, difusão atômica, entre outros (KSHEMKALYANI; SINGHAL, 2008).

A demonstração da insolubilidade do consenso em ambientes assíncronos (FISCHER; LYNCH; PATERSON, 1985) instigou os pesquisadores a investigar por alternativas que permitissem a construção de algoritmos de consenso para estes ambientes. Os **detectores de falhas não confiáveis** (CHANDRA; TOUEG, 1996) (ou simplesmente detectores de falhas) foi uma das alternativas introduzidas na literatura, com o propósito de resolver o problema do consenso em ambientes assíncronos. Em face deste fato, um detector de falhas torna-se peça fundamental em sistemas tolerantes a faltas, de tal maneira que a capacidade de resolução de problemas de acordo em um sistema assíncrono equipado com um detector de falhas torna-se equivalente à capacidade de resolução dos mesmos problemas em um sistema síncrono (CHANDRA; TOUEG, 1996).

Os blocos básicos de construção (*building blocks*) utilizados nos protocolos desenvolvidos nesta tese, isto é, aqueles que requerem algum tipo de acordo são baseados no algoritmo de consenso conhecido na literatura como "Paxos Bizantino" (ZIELINSKI, 2004; CASTRO; LISKOV, 1999).

2.2.3.2 Validação Atômica Não-Bloqueante

A validação atômica não-bloqueante (do inglês, *Non-Blocking Atomic Commitment*) ou simplesmente NBAC (HADZILACOS, 1990; BABAOGLU; TOUEG, 1993; RAYNAL; SINGHAL, 2001), também é um problema que pertence à classe de problemas de acordo em sistemas distribuídos. Particularmente, o NBAC teve sua origem nos sistemas de bancos de dados distribuídos (BERNSTEIN; NEWCOMER, 2009), cujo propósito era não apenas assegurar a atomicidade, mas também prover garantias de progressão (ou *liveness*) para transações distribuídas.

Tipicamente, uma transação distribuída realiza operações sobre dados residentes em vários locais (p. ex. sítios) e envolve um conjunto de processos – os participantes –, que cooperam entre si para a execução das operações que compõem a unidade lógica de processamento, em cada um dos sítios (BABAOGLU; TOUEG, 1993). Usualmente, um participante desempenha dois papéis em seu local de atuação (ou sítio); o de **gestor de transação** (*Transaction Manager* – TM) e o de **gestor de dados** (*Data Manager* – DM) – por vezes também conhecido por **gestor de recursos** (*Resource Manager* – RM) (BABAOGLU; TOUEG, 1993). O TM tem como atribuições receber a(s) operação(ões) que lhe foi(ram) solicitada(s), e realizar a cooperação com os demais sítios através de seus TMs, no decurso da transação. Já o DM é o responsável por executar a(s) operação(ões) sobre o(s) dado(s) a que esta(s) faz(em) referência naquele sítio (BABAOGLU; TOUEG, 1993).

Com isso, ao término do processamento da transação, a atomicidade global acerca das operações realizadas depende de um acordo entre os participantes engajados naquela transação. Deste modo, é imperioso que eles possam decidir de maneira única e irrevogável sobre o resultado final da transação. Este resultado deve pertencer a um domínio de valores V, tal que $V \supset \{COMMIT, ABORT\}$. Uma decisão pela validação ($COMMIT$) é legítima apenas se tudo ocorreu conforme o esperado durante o processamento da transação; ou do contrário, caso algo anormal tenha ocorrido (p. ex.: *deadlock*, insuficiência de capacidade de armazenamento, conflito de controle de concorrência, etc.) a decisão será pela anulação ($ABORT$). Este acordo é obtido através de um protocolo de validação atômica, que tem por finalidade assegurar a concordância de todos os participantes quanto ao resultado final da transação, o qual será determinado de acordo com as condições locais de cada participante. Com isso, ao concluir o processamento da transação, cada participante emite um voto (SIM ou NÃO), que declara a sua capacidade em garantir a consistência e a durabilidade da sua parte

na transação. E a partir dos votos recebidos é calculado um resultado, que normalmente é *COMMIT* se todos votaram SIM ou, do contrário, o resultado é *ABORT*.

Formalmente, a problema da validação atômica não-bloqueante é definido pelas seguintes propriedades (BABAOGLU; TOUEG, 1993; GUERRAOUI, 1995):

- **Acordo Uniforme**: todos os processos que decidem obtém um mesmo valor de decisão;
- **Validade Uniforme**: se um participante decide *COMMIT*, então todos os participantes votaram SIM;
- **Integridade Uniforme**: um participante decide no máximo uma vez, e não pode reverter sua decisão após ela ter sido tomada;
- **Terminação**: todos os participantes corretos terminam por obter um valor de decisão;
- **Não-Trivialidade**: se todos os participantes votaram SIM e não há falhas, então a decisão é *COMMIT*.

A essência do problema da validação atômica não-bloqueante é capturada pela condição especificada pela propriedade de **Terminação**, isto é, a condição que estabelece a ausência de blocagem no âmbito do problema. Note que esta propriedade especifica a condição de progressão (ou *liveness*) para a resolução do problema, um requisito que deve ser assegurado aos participantes corretos, a despeito de falhas verificadas nos demais. Um aspecto interessante decorre do fato de que tal propriedade é definida em termos de participantes corretos e não operacionais, já que um participante operacional não necessariamente é um participante correto (BABAOGLU; TOUEG, 1993).

Embora o NBAC seja um problema pertencente à classe de problemas de acordo em sistemas distribuídos, os resultados obtidos para o consenso em sistemas assíncronos equipados com detectores de falhas não são aplicáveis a ele. Este importante resultado foi demonstrado por Rachid Guerraoui, o qual demonstrou que não é possível resolver o NBAC em ambientes sujeitos a uma simples falta de parada (GUERRAOUI, 1995). Tal resultado decorre da propriedade **Não-Trivialidade**, que não pode ser plenamente satisfeita na maioria dos sistemas distribuídos que admitem falhas (GUERRAOUI, 1995), já que as falhas não podem ser verificadas de maneira segura com o uso de detectores de falhas não confiáveis (CHANDRA; TOUEG, 1996). A partir daí,

um novo problema de acordo para a terminação de transações foi definido, porém, com exigências mais fracas que o NBAC. O problema da **Validação Atômica Fraca Não-Bloqueante** (*Non-Blocking Weak Atomic Commitment*), ou NB-WAC, é especificado a partir das mesmas propriedades do NBAC, com exceção da **Não-Trivialidade**, que é enfraquecida levando em conta aspectos mais adequados, realistas e suficientes para os sistemas transacionais. No caso, o problema se baseia em "suspeitas de falhas" em vez de "falhas", e a propriedade passa a ser definida como (GUERRAOUI, 1995):

- **Não-Trivialidade**: se todos os participantes votaram SIM e não há suspeitas de falhas, então a decisão é *COMMIT*.

Note que diferente do NBAC (BABAOGLU; TOUEG, 1993), que não pode ser resolvido com detectores de falhas não-confiáveis; o enfraquecimento da propriedade relativa à **Não-Trivialidade** torna o NB-WAC (GUERRAOUI, 1995) redutível ao consenso, pois permite que os resultado obtidos para o consenso com detectores de falhas não-confiáveis sejam estendidos também ao NB-WAC.

2.2.3.3 Difusão de Mensagens

Conforme visto na Seção 2.2.1, de um modo geral, em sistemas distribuídos a cooperação entre os elementos do sistema ocorre por meio de troca de mensagens, e esta troca de mensagens pode ocorrer por meio de canais ponto-a-ponto ou, em alguns casos, por meio de canais de difusão (COULOURIS et al., 2012). A difusão de mensagens é um aspecto de grande importância no contexto de sistemas distribuídos, principalmente naqueles sistemas onde os elementos estão organizados em grupos e devem receber o(s) mesmo(s) conjunto(s) de mensagem(ns), isto é, onde é requerida a comunicação em grupo (BIRMAN, 1993; SCHIPER; RAYNAL, 1996). Um grupo pode ser definido como uma coleção de entidades distribuídas, na qual os membros podem se comunicar uns com os outros e a comunicação ocorre por meio de primitivas de difusão (BIRMAN, 1993).

De um lado, a construção de aplicações distribuídas utilizando apenas a comunicação ponto-a-ponto é por vezes cara e difícil de ser concretizada. Por outro lado, a construção de primitivas de difusão requer algum tipo de coordenação entre os elementos do sistema, a fim de garantir que todos venham a receber informações consistentes. As primitivas de difusão são essenciais à construção de aplicações que

requerem algum grau de confiabilidade (AVIZIENIS et al., 2004), onde geralmente a comunicação é baseada em grupo. Não obstante, em um sistema baseado na comunicação em grupo, as primitivas de difusão fornecem abstrações de comunicação tolerantes a faltas para os elementos do sistema, pois, de um modo geral, os subsistemas de comunicação empregados na construção de sistemas distribuídos são passíveis de falhas (HADZILACOS; TOUEG, 1993; CHANDRA; TOUEG, 1996; DÉFAGO; SCHIPER; URBÁN, 2004).

A literatura apresenta alguns problemas relacionados à difusão de mensagens em sistemas distribuídos, por exemplo, se o emissor sofre uma falha no decurso de uma transmissão ou se o meio de comunicação não é confiável, pode ocorrer que nem todos os receptores recebam a mensagem. Os problemas relacionados à difusão de mensagens em sistemas distribuídos estão relacionados, principalmente, no que diz respeito à provisão de confiabilidade na disseminação de mensagens a um grupo de processos. Uma visão bastante aprofundada dos principais problemas relacionados à difusão em sistemas distribuídos pode ser encontrada em (HADZILACOS; TOUEG, 1993; DÉFAGO; SCHIPER; URBÁN, 2004). Nesta tese nos limitamos a discutir apenas a **difusão confiável** e a **difusão com ordem total**. Um aspecto que também merece destaque, é que tanto as propriedades da **difusão confiável** como da **difusão com ordem total** são definidas em termos de entrega de mensagens ao protocolo de nível superior, e não da recepção de mensagens na máquina.

Difusão Confiável

O problema da **difusão confiável** (do inglês, *reliable multicast*) (BIRMAN; JOSEPH, 1987; HADZILACOS; TOUEG, 1993; CHANDRA; TOUEG, 1996) está relacionado ao provimento de confiabilidade na difusão de uma mensagem, ou de um conjunto de mensagens a um grupo de processos. Neste sentido, um protocolo de difusão confiável deve assegurar que todas as mensagens enviadas a um grupo de processos serão recebidas, e consequentemente entregues por todos os membros corretos daquele grupo. Em sua essência, a difusão confiável garante que (i) todos os processos corretos entregam o mesmo conjunto de mensagens, e (2) se o emissor de uma mensagem é correto, então aquela mensagem será entregue.

Seja $\mathcal{M}$ o conjunto das possíveis mensagens do sistema, $\mathcal{P}$ o conjunto de todos os processos que fazem parte do sistema, e $\mathcal{G}$ um grupo de processos, tal que $\mathcal{G} \subset \mathcal{P}$. Formalmente, a difusão confiável é

definida acerca das seguintes primitivas (CHANDRA; TOUEG, 1996):

- *R-multicast*$(\mathcal{G}, m)$: que denota a disseminação da mensagem m aos processos que pertencem ao grupo $\mathcal{G}$, e;
- *R-deliver*(m): que denota a entrega, isto é, a liberação da mensagem m para a aplicação de nível superior.

Não obstante, a especificação das primitivas *R-multicast* e *R-deliver* devem satisfazer às seguintes propriedades (CHANDRA; TOUEG, 1996):

- **Validade**: se um processo correto dissemina uma mensagem m ao grupo $\mathcal{G}$, então algum processo correto em $\mathcal{G}$ finará por entregar m;
- **Acordo**: se um processo correto em $\mathcal{G}$ entrega uma mensagem m, então todos os processos corretos em $\mathcal{G}$ terminarão por entregar m;
- **Integridade**: para qualquer mensagem m, cada processo correto em $\mathcal{G}$ entrega m no máximo uma vez, e somente se m foi previamente disseminada à $\mathcal{G}$.

Embora esta seção tenha formalizado a difusão confiável considerando cenários de apenas um grupo de processos, a especificação da difusão confiável provê mecanismos que permitem a disseminação de mensagens a múltiplos grupos (FRITZKE JR. et al., 2001). Todavia, o uso de múltiplos grupos está fora do escopo desta tese, razão pela qual nos limitamos em discutir apenas os aspectos mais elementares da difusão confiável.

Difusão com Ordem Total

A difusão com ordem total é definida pelas primitivas *TO-multicast*$(\mathcal{G}, m)$ e *TO-deliver*(m), de modo que elas são especificadas de forma idêntica às da difusão confiável, acrescida pela propriedade de ordenação total (BIRMAN; JOSEPH, 1987; CHANDRA; TOUEG, 1996; DÉFAGO; SCHIPER; URBÁN, 2004).

- **Ordenação Total**: se dois processos corretos p_i e p_j, tal que $i \neq j$, entregam as mensagens m_1 e m_2 disseminadas em $\mathcal{G}$, então p_i entrega m_1 antes de m_2 se, e somente se p_j entregar m_1 antes de m_2; isto é, ambos os processos entregam as duas mensagens na mesma ordem.

Note que a concretização da difusão com ordem total requer a consecução de um acordo entre os membros de $\mathcal{G}$, no intuito de assegurar a propriedade de **Ordenação Total** – um feito que pode ser realizado através de um consenso (vide Seção 2.2.3.1). A ordem total induzida pela primitiva *TO-deliver* é denotada pela relação de ordem/precedência $\prec$. Neste caso, é dito que *TO-deliver*$(m_1) \prec$ *TO-deliver*(m_2), se a mensagem m_1 é entregue antes da mensagem m_2. Além disso, da mesma forma como ocorre com a difusão confiável, a difusão com ordem total também pode operar com múltiplos grupos. Uma discussão mais aprofundada sobre o assunto pode ser encontrada num trabalho de Fritzke Jr. *et al.* (2001).

2.2.3.4 Replicação de Máquina de Estados

A redundância é um dos mecanismos mais usuais e fundamentais para a implementação de sistemas distribuídos tolerantes a faltas. Tipicamente, esta redundância é obtida por meio do emprego de técnicas de replicação, tal como a *Replicação de Máquina de Estados* (RME) (LAMPORT, 1978; SCHNEIDER, 1990). A RME explora o princípio de que um serviço (ou aplicação), pode ser modelado e implementado a partir de um conjunto de réplicas. Por definição, as réplicas devem ser deterministas, devem iniciar em um mesmo estado e serem sujeitas a uma mesma sequência de operações, de modo que seja possível assegurar a evolução destas para um mesmo estado final.

A abordagem de máquina de estados foi introduzida na literatura por Leslie Lamport (LAMPORT, 1978), em que o mesmo a caracterizou como um conjunto de estados possíveis e um conjunto de comandos (ou operações) possíveis. Na definição de Lamport, a evolução da máquina de estados ocorre por meio de uma função determinista, que executa atomicamente um comando sobre o estado atual da máquina e a conduz a um novo estado. Neste sentido, a considerar que a transição de estados na máquina ocorre de maneira determinista, a saída desta máquina de estados é inteiramente determinada pelo seu estado inicial e pela sequência de comandos executados.

Alguns anos mais tarde, a abordagem de máquina de estados foi aprimorada por Fred Schneider (SCHNEIDER, 1990), onde o autor criou réplicas de uma máquina de estados e as organizou em um sistema distribuído. A ideia de Schneider (1990) culminou na especificação da *Replicação de Máquina de Estados*, pela qual fora demonstrada uma implementação prática da abordagem proposta (i.e., a RME) para to-

lerar faltas por parada dos processos. Não obstante, o primeiro trabalho a utilizar a abordagem da RME para tolerar faltas Bizantinas foi o trabalho popularmente conhecido na literatura como PBFT (acrônimo de *Practical Byzantine Fault Tolerance*), dos autores Miguel Castro e Barbara Liskov (CASTRO; LISKOV, 1999).

Uma vez que a caracterização semântica de uma réplica de máquina de estados é tal que, sua saída deve depender unicamente de seu estado inicial e da sequência de comandos executados (SCHNEIDER, 1990), todo e qualquer evento/fator/aspecto que não os comandos de entrada devem ser desconsiderados na RME, a fim de evitar problemas relacionados ao não-determinismo. Esta característica está relacionada ao que é conhecido na literatura por **determinismo de réplicas** (SCHNEIDER, 1990), isto é, o requisito que estipula que réplicas partindo de um mesmo estado inicial e sujeitas à mesma sequência de requisições de entrada submetidas na mesma ordem, devem chegar ao mesmo estado final. E do mesmo modo, para se obter o determinismo de réplicas, a especificação da RME define um requisito denominado coordenação de réplicas, o qual é dividido em dois requisitos particulares (SCHNEIDER, 1990):

1. **Acordo**: todas as réplicas de máquina de estados corretas recebem as mesmas requisições;

2. **Ordem**: todas as réplicas de máquina de estados corretas processam as requisições recebidas em uma mesma ordem.

Estes dois requisitos permitem implementar serviços replicados a partir de uma RME, de modo a satisfazer a propriedade de consistência forte conhecida por **linearização** (do inglês, *linearizability*) (HERLIHY; WING, 1990). A linearização parte do pressuposto de que modificações realizadas por operações concomitantes num objeto concorrente, devem ocorrer de maneira atômica, de modo a evitar que uma sobreposição destas cause a corrupção do estado do objeto. Para tanto, o comportamento quanto à execução das operações no objeto, deve ser equivalente ao de uma execução sequencial, isto é, de uma operação após a outra. Além disso, o efeito produzido no estado do objeto, pela execução de uma dada operação, deve ser imediatamente visível para a operação subsequente, tal que a operação deve produzir efeito antes de seu retorno.

A especificação formal de linearização define um sistema linearizável, como aquele no qual é possível reordenar cada histórico produzido, de modo a respeitar a semântica das operações, tal como definido

nas respectivas especificações sequenciais; além de preservar a relação de ordem de todas as operações executadas sem sobreposição (HERLIHY; WING, 1990). Note que o termo "linearização" refere-se ao fato que, embora haja concorrência entre as operações, tem-se a impressão de que cada operação ocorre em uma ordem linear "bem formada". A despeito da linearização ser o critério de consistência mais empregado em protocolos que implementam RME (CASTRO; LISKOV, 1999; YIN et al., 2003; KOTLA et al., 2007), é importante salientar que a literatura dispõe de critérios de consistência mais fracos que a linearização (p. ex.: causal, eventual, etc. (VOGELS, 2009)). Todavia, tal discussão está fora do escopo deste trabalho, porquanto o critério de consistência requerido, quando empregados os mecanismos pertinentes à RME no âmbito do desenvolvimento desta tese, é baseado na linearização (maiores detalhes no Capítulo 4). Uma argumentação concisa acerca de critérios de consistência mais fracos é realizada pelo teorema CAP (BREWER, 2012).

Por fim, um aspecto não menos importante e que cabe ser elucidado, é que no cerne da RME está a difusão com ordem total, que é um meio factível para prover o requisito de coordenação de réplicas, bem como para satisfazer a linearização. Isto decorre do fato de que as propriedades definidas para a difusão com ordem total (vide Seção 2.2.3.3), são suficientes para assegurar a consistência do estado das réplicas (acordo e ordem). No caso, ao receber uma requisição ordenada, cada réplica executa a operação, atualiza seu estado – quando necessário –, e envia ao cliente o resultado da operação. Um cliente aceita o resultado da operação se recebe $f + 1$ respostas iguais advindas de diferentes réplicas, sendo f o número máximo de réplicas que podem falhar. Com isso, possíveis inconsistências verificadas nas respostas recebidas são mascaradas a partir da aplicação do voto majoritário sobre elas, de modo que aquela que predominar será considerada como o resultado da operação.

2.3 TECNOLOGIA DE VIRTUALIZAÇÃO

Apesar do conceito de virtualização ter sido introduzido na literatura na década de 60 (PARMELEE et al., 1972; POPEK; GOLDBERG, 1974), ele ganhou maior notoriedade na década de 90, com o advento das tecnologias de virtualização capazes de operar sob a arquitetura x86 (ROSENBLUM, 2004). A virtualização (PARMELEE et al., 1972; POPEK; GOLDBERG, 1974; SMITH; NAIR, 2005) pode ser vista como uma

tecnologia inovadora que permite abstrair os recursos de um sistema de computação, de tal maneira que seja possível obter um melhor aproveitamento dos recursos físicos. Um dos propósitos da virtualização é prover um ambiente computacional virtual que permita melhorar a utilização dos recursos, ao mesmo tempo em que ofereça uma plataforma operacional integrada e unificada para usuários e aplicações baseadas na agregação de recursos heterogêneos e autônomos. De um modo geral, a virtualização é provida por meio do que é conhecido por máquina virtual (SMITH; NAIR, 2005).

Da perspectiva de máquina virtual, doravante referenciada apenas pelo acrônimo VM (de *Virtual Machine*), a literatura apresenta diversas definições para o termo "máquina virtual", o que conduz à certa ambiguidade no que concerne à interpretação do mesmo. As primeiras definições para máquina virtuais foram as de Parmelee *et al.* (1972) e de Popek e Goldberg (1974). O trabalho de Parmelee *et al.* (1972) definiu um sistema de máquinas virtuais como um "sistema de computação no qual as instruções emitidas por um programa podem ser diferentes daquelas realmente executadas pelo hardware". Por outro lado, Popek e Goldberg (1974) definiram máquina virtual como uma "cópia isolada e eficiente de uma máquina real".

Embora as definições seminais de Parmelee *et al.* (1972) e de Popek e Goldberg (1974) tenham indicado a existência de único tipo de máquina virtual, com a popularização da tecnologia de virtualização nos anos 2000, novas definições acerca do assunto foram apresentadas na literatura (ROSENBLUM, 2004; SMITH; NAIR, 2005). Uma das definições mais usuais na atualidade é a de Smith e Nair (2005), onde os autores classificaram as VMs em duas categorias, mais precisamente, as máquinas virtuais de processo e as máquinas virtuais de sistema.

- **Máquina Virtual de Processo**: consiste em uma plataforma virtual que executa um processo individual. É um tipo de VM que é criada apenas para dar suporte ao processo virtualizado, de modo que, ao término da execução do processo ela também é terminada. Alguns exemplos deste tipo de VM são a *Java Virtual Machine* (HORSTMANN; CORNELL, 2004) e a *Common Language Runtime* da plataforma .NET (LIMA; REIS, 2002) – ambas implementam aspectos de virtualização em nível de linguagem de programação, como suporte para a execução de software baseado em *bytecodes*;

- **Máquina Virtual de Sistema**: consiste num tipo de máquina que virtualiza o ambiente em um nível mais baixo de abstração

(p. ex.: em nível de hardware ou de arquitetura, tal como o ISA – *Instruction Set Architecture* (SMITH; NAIR, 2005)). Este tipo de VM suporta a virtualização de um sistema operacional completo, onde tipicamente é usada a nomenclatura de convidado (*guest*) para uma instância de VM, e de anfitrião (*host*) para a plataforma sob a qual está sendo executada a VM (HORSTMANN; CORNELL, 2004; SMITH; NAIR, 2005). Este tipo de VM fornece às aplicações os recursos de hardware virtuais necessários a sua execução, tais como: E/S virtual, memória virtual, processador virtual, etc. Alguns exemplos deste tipo de VM são o VMware (VMWARE, 2014), o Xen (XEN, 2014), o VirtualBox (ORACLE, 2014b) e o KVM (KVM, 2014).

Em se tratando das Máquinas Virtuais de Sistema, um componente denominado Monitor de Máquina Virtual (do inglês, *Virtual Machine Monitor* – VMM) (POPEK; GOLDBERG, 1974) – também conhecido como *HyperVisor* (BRESSOUD; SCHNEIDER, 1995) é o responsável por gerenciar a execução das máquinas virtuais, isto é, os sistemas convidados. No entanto, para que um VMM possa ser considerado como um sistema de controle de máquinas virtuais, ele deve atender a algumas propriedades fundamentais (POPEK; GOLDBERG, 1974; GARFINKEL; ROSENBLUM, 2003). São elas:

- **Eficiência**: todas as instruções inofensivas devem ser executadas diretamente pelo hardware, sem intervenção do VMM;

- **Controle de Recursos**: um programa arbitrário executado como convidado não consegue afetar os recursos do sistema, de modo que o VMM possui total controle sobre os recursos que são a ele fornecidos;

- **Equivalência**: para qualquer programa de aplicação, o comportamento exarado pela execução do mesmo sobre um VMM é idêntico ao comportamento verificado pela sua execução em um ambiente não virtualizado;

- **Isolamento**: um programa executado por uma VM em nível de convidado não consegue acessar ou modificar o VMM nem qualquer outra VM em nível de convidado;

- **Inspeção**: todo(s) o(s) estado(s) da(s) VM(s) está(ão) disponível(eis) ao VMM;

- **Interposição**: o VMM deve intervir na execução de determinadas operações realizadas por máquinas virtuais, tais como, instruções privilegiadas.

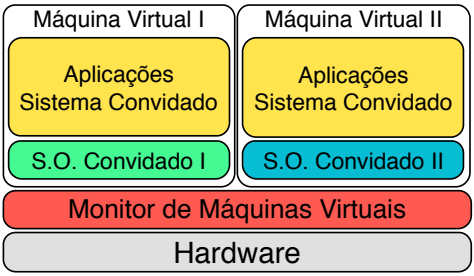


Figura 15 – Monitor de máquinas virtuais Tipo I.

Na literatura, os VMMs estão classificados em dois tipos, nomeadamente como Tipo I e Tipo II (KING; DUNLAP; CHEN, 2003). As Figuras 15 e 16 ilustram os VMMs dos Tipos I e II, respectivamente. Os VMMs do Tipo I são executados diretamente sobre o hardware da máquina física, enquanto que os VMMs do Tipo II são executados como uma aplicação a partir de um sistema operacional anfitrião, que é instalado sobre a máquina física.

Os VMMs do Tipo I, que implementam virtualização em nível de hardware, foram introduzidos desde as primeiras especificações da tecnologia de virtualização (ROSENBLUM, 2004), enquanto que os VMMs do Tipo II, que implementam virtualização em nível de software, foram introduzidos pela VMware (VMWARE, 2014) no ano de 1999 (ROSENBLUM, 2004). É evidente que os VMMs do Tipo I, em relação aos VMMs do Tipo II, apresentam vantagem em termos de desempenho, já que os do Tipo I são implementados diretamente sobre o hardware físico, e os do Tipo II possuem um elemento intermediário entre eles e o hardware – o sistema operacional anfitrião. Por outro lado, as vantagens dos VMMs do Tipo II sobre os VMMs do Tipo I, decorre do fato de que, como o VMM Tipo II é executado como um processo normal no contexto do sistema operacional anfitrião, torna-se possível o uso de ferramentas de monitoramento e depuração do sistema anfitrião, para realizar inspeções tanto no VMM como nas VMs executadas através dele.

No âmbito de sistemas distribuídos, a tecnologia de virtualização tem se mostrado como um mecanismo bastante atrativo para imple-

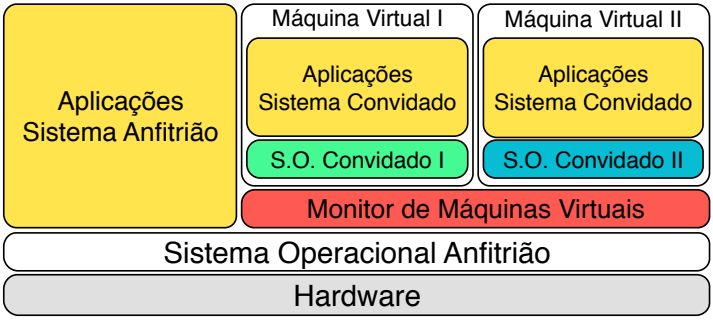


Figura 16 – Monitor de máquinas virtuais Tipo II.

mentar soluções computacionais tolerantes a falhas (BRESSOUD; SCHNEIDER, 1995; REISER; KAPITZA, 2007; STUMM JR. et al., 2010; WOOD et al., 2011; DETTONI et al., 2013a). Esta tese emprega a tecnologia de virtualização como suporte para a especificação de uma arquitetura de sistema distribuído, com o propósito de resolver o problema de acordo da validação atômica não-bloqueante (BABAOGLU; TOUEG, 1993) em ambientes sujeitos à faltas bizantinas.

2.4 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram apresentados alguns conceitos em bancos de dados e transações, bem como alguns modelos fundamentais de sistemas distribuídos, em que ambos serão utilizados nos demais capítulos desta tese. Além disso, foram apresentadas as principais ferramentas de sistemas distribuídos (ou blocos básicos), os quais serão utilizados na especificação e construção das contribuições desenvolvidas nesta tese. Por fim, cabe salientar que grande parte dos conceitos discutidos ao longo do capítulo serão úteis para uma melhor compreensão acerca da especificação dos algoritmos objetos desta tese, bem como para a demonstração das provas de correção – necessárias para atestar que as proposições desenvolvidas nesta tese atendem aos requisitos funcionais estipulados.

3 CONTEXTUALIZAÇÃO

Este capítulo versa sobre a motivação por trás da noção de tolerância a faltas bizantinas (*Byzantine Fault Tolerance*), doravante referenciado pelo acrônimo BFT. Neste sentido, é realizada a apresentação de alguns conceitos introdutórios em BFT e também uma visão geral acerca dos mecanismos empregados para prover a tolerância a faltas bizantinas, com vista para assegurar a consistência das aplicações. Esta apresentação é seguida pela descrição do que é considerado o estado da arte em protocolos tolerantes a faltas bizantinas na atualidade, tanto para serviços não transacionais como para transações e bancos de dados. Além disso, também são apresentados os trabalhos correlacionados às contribuições desta tese, os quais são apresentados de acordo com os respectivos enquadramentos aos trabalhos desenvolvidos.

3.1 CONCEITOS EM TOLERÂNCIA A FALTAS BIZANTINAS

Embora o conceito de faltas bizantinas tenha sido proposto há mais de trinta anos (LAMPORT; SHOSTAK; PEASE, 1982) (vide Seção 3.1.1), ele ganhou notoriedade a partir de um trabalho seminal de Castro e Liskov (CASTRO; LISKOV, 1999), o qual desmistificou diversos aspectos relacionados ao custo em se tolerar faltas bizantinas.

Sistemas tolerantes a faltas bizantinas podem ser entendidos como aqueles sistemas computacionais resilientes a faltas de natureza arbitrária, as quais podem ocorrer durante a execução de um algoritmo em um sistema de computação distribuída (LAMPORT; SHOSTAK; PEASE, 1982). As faltas de natureza arbitrária (ou bizantinas) englobam tanto faltas por omissão, tais como, faltas por parada (*crash*), faltas na recepção de mensagens, etc., bem como faltas por comissão, isto é, processamento incorreto de uma requisição de uma aplicação, envio de respostas inconsistentes, ou corrupção proposital do estado da aplicação. Apesar de as faltas por parada de natureza acidental serem as mais comuns nos sistemas computacionais da atualidade, é lícito salientar que a ocorrência de faltas bizantinas nestes sistemas é uma realidade (SCHROEDER; GIBSON, 2007; SCHROEDER; PINHEIRO; WEBER, 2009; NIGHTINGALE; DOUCEUR; ORGOVAN, 2011). De um modo geral, as faltas de parada apenas comprometem a execução de uma aplicação, enquanto que as faltas bizantinas podem comprometer não apenas a execução, mas também o estado da aplicação. Em alguns casos, a pa-

rada prematura de um sistema pode resultar na corrupção do estado da aplicação. Por outro lado, faltas bizantinas podem corromper o estado sem parar a aplicação, o que pode acarretar na execução de requisições sobre um estado inconsistente, e com isso possibilitar o recebimento de resultados incorretos pelos clientes da aplicação.

A despeito da ocorrência de faltas acidentais ser mais comum no âmbito de sistemas computacionais, faltas oriundas de erros de corrupção em memória e/ou disco são bastante frequentes em sistemas reais. Estudos evidenciaram que, por ano, pelos menos 8% dos módulos de memória DRAM dos servidores em *data centers* da Google sofrem erros, os quais não são mascarados nem por códigos de correção de erros (ECC) presentes nestes bancos de memória (SCHROEDER; PINHEIRO; WEBER, 2009). Alguns destes erros são ocasionados por *bit flips* (i.e., inversão de bits), que muitas vezes ocorrem devido a um fenômeno conhecido por *cosmic ray*, isto é, uma espécie de perturbação gerada por um campo eletromagnético. Além disso, um estudo realizado pela Microsoft em aproximadamente 1 (um) milhão de PCs domésticos, demonstrou que falhas em CPUs e *chipsets* são bastante frequentes (NIGHTINGALE; DOUCEUR; ORGOVAN, 2011).

A tolerância a faltas bizantinas (BFT) consiste em uma técnica usada para prover a resiliência de um sistema computacional contra faltas de natureza arbitrária (i.e., bizantinas). Conforme já dito, faltas bizantinas são aquelas faltas que impõem nos sistemas computacionais um comportamento imprevisível e diferente daquele para o qual o sistema foi projetado, o que inclui, por exemplo: o processamento incorreto de uma solicitação; o envio de respostas inconsistentes; a corrupção intencional do estado da aplicação, dentre outros. Se não tratadas adequadamente, estas faltas podem resultar em consequências danosas ao sistema. Não obstante, sistemas que não toleram faltas bizantinas podem manifestar comportamentos imprevisíveis quando da ocorrência de alguma falta desta natureza, durante o ciclo de vida da aplicação responsável pela execução do sistema em questão.

3.1.1 Acordo Bizantino

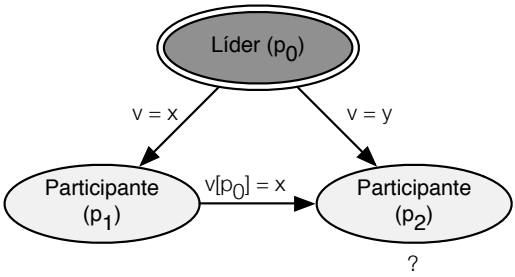
A primeira menção ao termo "falta bizantina" foi introduzida na literatura por Lamport, Shostak e Pease (1982), em que os autores o utilizaram para formalizar uma solução para um problema de acordo entre um grupo de processos em um sistema distribuído, onde alguns destes processos poderiam enviar valores conflitantes entre eles. Nesta

solução, os processos representam os generais bizantinos, onde dentre eles, um representa o general comandante (ou líder) e os demais representam os generais tenentes (ou apenas participantes). Para tanto, o comandante (líder) envia uma ordem aos tenentes (participantes) e todos os tenentes corretos devem executar a mesma ordem. No caso, se o comandante for correto, os tenentes corretos executam a ordem emitida pelo comandante.

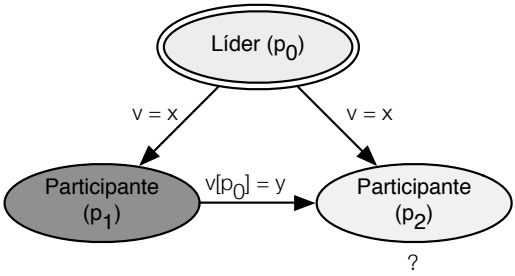
A solução proposta por Lamport *et al.* (1982) é especificada a partir de um ambiente síncrono, para a qual os autores demonstraram que para resolver o problema de acordo distribuído no modelo de falhas por eles introduzido, é requerido que 2/3 dos participantes sejam corretos. Neste caso, é necessário pelos menos $n = 3f+1$ participantes, sendo f o número máximo de participantes faltosos permitido para a solução proposta. Os autores provaram que a presença de 1 (um) participante bizantino, isto é $f = 1$, dentre um grupo formado por $n < 3f + 1$ participantes implica numa impossibilidade de obtenção do acordo. Esta impossibilidade é ilustrada na Figura 17, onde se tem $f = 1$ e $n = 3$, e portanto sendo $n < 3f + 1$.

O problema dos generais bizantinos é definido pelo seguinte, em um exército, que corresponde ao grupo de processos, há um general comandante – o processo p_0 da Figura 17, e diversos generais tenentes – os processos p_1 e p_2 da mesma figura. Como o general comandante lidera os exércitos, ele é o responsável por emitir uma ordem aos generais tenentes, na qual esta ordem expressa os possíveis valores de decisão para o acordo (x ou y no caso da Figura 17). Após receber uma ordem, os generais tenentes trocam informações entre si para validar a ordem emitida pelo general comandante. No primeiro caso ilustrado na Figura 17(a), o general comandante é um traidor e emite ordens diferentes aos generais tenentes, e então, o general tenente p_1 envia uma mensagem ao general tenente p_2 para confirmar a ordem recebida do general comandante, isto é, x. No entanto, ao receber a confirmação de p_1, o general tenente p_2 não sabe qual ação deve ser tomada, pois ele recebeu um y de p_0 e um x de p_1. O mesmo ocorre para o cenário ilustrado pela Figura 17(b), onde o traidor é o general tenente p_1. Neste caso particular, p_1 tenta confundir o general tenente p_2 atestando que a ordem emitida pelo general foi um y. Como p_2 recebeu um x de p_0, ele entra em um estado de indecisão e não sabe qual ação deve ser de fato executada. Em ambos os cenários não é possível obter êxito na execução da ordem emitida, ao mesmo tempo que não se sabe quem é o traidor.

A Figura 18 ilustra um cenário, no qual a condição de possibi-



(a) Cenário onde o líder (p_0) é faltoso.



(b) Cenário onde o participante p_1 é faltoso.

Figura 17 – Impossibilidade do acordo bizantino com $n < 3f + 1$

lidade de solução para o problema dos generais bizantinos se verifica. Tal resultado pode ser observado a partir da segunda fase do algoritmo, onde p_1, p_2 e p_3 trocam mensagens entre si para confirmar a ordem emitida por p_0. Ao término da execução daquela fase, p_1 e p_2 terão recebido os valores x, x, y, enquanto que p_3 terá recebido os valores y, x, x. Deste modo, todos terão condições de decidir pelo valor majoritário da lista de valores, isto é, x, o que por conseguinte, prova que o acordo bizantino só é solúvel com pelo menos $3f + 1$ participantes. Note que esta impossibilidade de acordo bizantino com $n < 3f + 1$ é válida tanto em sistemas síncronos quanto em sistemas assíncronos.

À vista disso, na literatura se encontra soluções alternativas para o problema, que empregam mecanismos criptográficos para prover a autenticação das mensagens (DOLEV; STRONG, 1983). A autenticidade é provida por um esquema de assinaturas, o qual possibilita a verificação da autenticidade das mensagens, e portanto, permite identificar um processo faltoso (i.e., um traidor) no algoritmo. Todavia, mesmo com

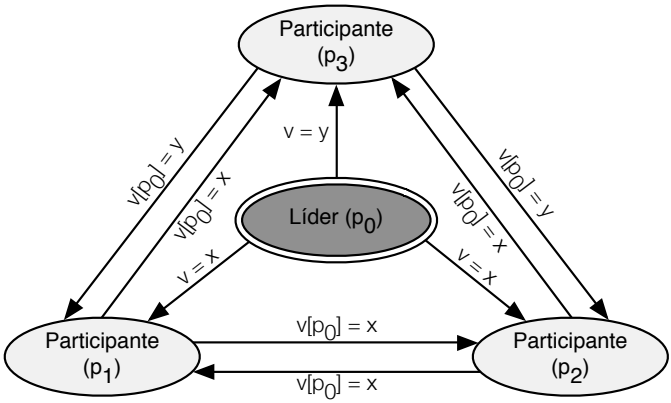


Figura 18 – Acordo bizantino com $n \geq 3f + 1$.

o uso de autenticação, a impossibilidade do acordo bizantino ainda é válida em sistemas assíncronos, se o número de participantes corretos for menor do que 2/3 do total, isto é, $n - f < 2/3$ (TOUEG, 1984). Este resultado se baseia no fato de que mesmo tendo provas quanto à autenticidade da mensagem, um processo bizantino pode se prevalecer do adiamento na entrega das mensagens em um ambiente assíncrono, e assim fazer com que os processos corretos decidam valores diferentes, se $n - f < 2/3$.

3.1.2 Tolerância a Faltas Bizantinas através de Replicação

A tolerância a faltas bizantinas consiste em uma técnica que visa aumentar a disponibilidade e também prover uma melhor confiabilidade acerca de sistemas computacionais, nos quais um número limitado de entidades pode se comportar de maneira arbitrária, e portanto, manifestar faltas bizantinas. De um modo geral, esta tolerância é obtida a partir do uso de técnicas de redundância, que é imposta sobre a aplicação para a qual se deseja prover a capacidade de tolerar as faltas bizantinas (SCHNEIDER; ZHOU, 2005; CACHIN, 2010). No caso, a abordagem de replicação mais comum adotada para tolerância a faltas bizantinas em sistemas computacionais é a **R**eplicação de **M**áquina de **E**stados – RME (SCHNEIDER, 1990), já apresentada na Seção 2.2.3.4.

Tipicamente, os protocolos para RME tolerante a faltas bizanti-

nas (ou RME para BFT) admitem suposições mais realistas, nas quais o ambiente do sistema computacional (p. ex.: sistemas, computadores, redes, etc.) pode apresentar comportamentos inesperados devido à falhas no hardware, problemas transientes de rede, ou mesmo ações maliciosas. Em suma, protocolos RME para BFT são especificados a partir de um conjunto de réplicas de um dado serviço, no qual cada réplica mantém um conjunto de variáveis de estado. Estas variáveis são manuseadas a partir de operações deterministas de leitura e de escrita, as quais são executadas de maneira atômica sobre as variáveis, a fim de que os requisitos básicos da RME sejam atendidos (vide Seção 2.2.3.4). Os requisitos básicos da RME clássica podem ser atendidos pelo uso de duas classes de protocolos utilizadas na replicação BFT, são eles: os protocolos baseados em acordo e os protocolos baseados em quóruns (LYNCH, 1996).

A classe de protocolos baseados em acordo são essencialmente especificados a partir das premissas básicas da RME para BFT (CASTRO; LISKOV, 1999; YIN et al., 2003; KOTLA et al., 2007; GUERRAOUI et al., 2010). Estes protocolos se baseiam no uso de uma réplica primária para estabelecer uma ordem sequencial para a execução das operações oriundas dos clientes de um serviço, o que é realizado a partir de um protocolo de acordo executado entre as réplicas para assegurar a concordância acerca da ordem proposta. Este acordo é possível apenas num ambiente com, pelo menos $3f + 1$ réplicas, e é realizado a partir de um protocolo com múltiplas fases e um padrão de comunicação com complexidade quadrática, isto é, onde todos enviam para todos. Em razão dos padrões de comunicação utilizados em tais protocolos, na medida em que se aumenta o número de faltas toleradas (i.e., o valor de f), o número de mensagens cresce demasiadamente, o que por conseguinte pode propender a problemas de escalabilidade.

Por outro lado, alguns protocolos de replicação para BFT se baseiam no uso de um quórum de réplicas corretas em vez de tentar impor um acordo entre as elas (MALKHI; REITER, 1997). Os sistemas de quóruns como são conhecidos, adotam abordagens otimistas, onde cada cliente contata um quórum de réplicas que ordenam de forma independente cada uma das operações. No caso, o quórum, que representa um conjunto de réplicas, é organizado de tal maneira que quaisquer dois quóruns se intersectam em, pelo menos, uma réplica correta. No padrão de comunicação baseado em quóruns, as operações completam apenas se a ordenação atribuída por cada uma das réplicas naquele quórum é consistente. Se a consistência não puder ser verificada, os sistemas de quóruns iniciam uma fase de reconciliação, que em geral

é custosa, e portanto, afeta o desempenho do protocolo. No entanto, é lícito salientar que este tipo de situação ocorre com maior frequência onde há contenção em operações de escrita, em que vários clientes tentar executar uma operação de escrita simultaneamente. Todavia, esta abordagem de replicação BFT não é adequada para ambientes nos quais é requerido alto desempenho, a despeito de contenção.

No que segue, esta seção apresenta uma explanação acerca de um aspecto de crucial importância no contexto de BFT, que é o limite de faltas em função do número de réplicas necessárias numa abordagem de replicação para BFT. Na sequência são apresentadas algumas das propostas encontradas na literatura, as quais são consideradas como sendo o estado-da-arte em termos de protocolos para replicação BFT.

3.1.2.1 Limites Empregados na Replicação para BFT

De um modo geral, as soluções algorítmicas para replicação BFT admitem que, no máximo, $f \leq \lfloor \frac{n-1}{3} \rfloor$ elementos de um conjunto de réplicas $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ com cardinalidade $n = |\mathcal{R}|$, podem desviar arbitrariamente de suas especificações durante uma janela de vulnerabilidade do sistema (i.e., um período) (CASTRO; LISKOV, 1999). De outro modo, no modelo de falhas por parada, não mais de $f \leq \lfloor \frac{n-1}{2} \rfloor$ podem falhar simultaneamente em uma mesma janela de vulnerabilidade (LAMPORT, 2001). Estes limites decorrem dos valores mínimos requeridos para se implementar os requisitos básicos da RME (i.e., acordo e ordem – vide Seção 2.2.3.4), de modo que seja possível tolerar cada um destes tipos de falhas.

Por outro lado, se levados em consideração os requisitos básicos da RME, o processamento de uma operação em um serviço implementado a partir desta abordagem é realizado em duas etapas. A primeira etapa consiste na ordenação das requisições que contém as operações, e a etapa seguinte é a execução das operações após a respectiva ordenação. Neste caso, para cada uma das etapas de processamento das operações é requerido um número diferente de réplicas para tolerar as faltas desejadas. A Tabela 2 adaptada de (BESSANI; ALCHIERI, 2014), enumera os resultados ótimos obtidos para cada uma das etapas, levando em conta o tipo de faltas que se deseja tolerar.

Note que, para a ordenação são requeridas $2f + 1$ e $3f + 1$ réplicas para se tolerar faltas de parada e faltas bizantinas, respectivamente. Estes resultados decorrem do número necessário de réplicas para executar um protocolo de acordo para a ordenação de mensagens, como

Tabela 2 – Limiar de faltas toleradas em um protocolo de replicação.

Tipo de faltas	**Acordo/Ordenação**	**Execução/Mascaramento**
Parada (*crash*)	$2f+1$	$f+1$
Bizantina	$3f+1$	$2f+1$

por exemplo, o Paxos (LAMPORT, 1978; ZIELINSKI, 2004). De outro modo, uma vez que a ordem é estabelecida, para o mascaramento das faltas ocorridas durante a execução da operação são requeridas $f+1$ e $2f+1$ réplicas para faltas de parada e bizantina, respectivamente (SCHNEIDER, 1990) – números suficientes para se estabelecer uma maioria e determinar a validade do resultado de uma operação (p. ex.: $n-f > f$). No entanto, embora seja necessário apenas $2f+1$ réplicas para a execução de operações em uma RME para BFT, este número pode resultar em problemas acerca das propriedades de um serviço replicado (p. ex.: *safety* e *liveness*). Na sequência é dada uma breve explicação sobre este fato. Para um melhor entendimento acerca desta explicação, considere $\mathcal{R}$ como o conjunto de todas as réplicas presentes no sistema, $n = |\mathcal{R}|$ como o número total de réplicas do sistema e f como o número máximo de faltas permitidas durante uma janela de vulnerabilidade do sistema.

No primeiro caso, suponha a existência de um serviço especificado por uma RME tolerante a faltas de parada, tal que este serviço implementa a exclusão mútua entre as operações de leitura e de escrita, para fins de manutenção da consistência. Quando um cliente c_i emite uma operação de escrita sobre este serviço – de acordo com as propriedades da RME (vide Seção 2.2.3.4) –, ele deve aguardar o recebimento de $n-f$ respostas mutuamente consistentes de diferentes réplicas, para completar a operação. Note que tal aspecto também assegura a progressão do sistema, já que f réplicas podem não responder a todas as operações. De outro modo, quando um cliente c_j (tal que $i \neq j$) emite uma operação de leitura, ele também deve aguardar pelas $n-f$ respostas mutuamente consistentes de diferentes réplicas para completar sua operação. No entanto, estas respostas podem ser enviadas por outro subconjunto de réplicas em $\mathcal{R}$. Neste caso, o conjunto de réplicas comuns existentes entre os quóruns de leitura e de escrita é de tamanho $n-2f$[1]. E para que o serviço seja considerado como correto, os quóruns de leitura e de escrita devem se intersectar em, pelo menos, uma

[1] O quórum denota um conjunto de réplicas que atende aquela requisição

réplica, réplica esta que por construção não é faltosa, já que no modelo de falhas por parada ela teria parado de funcionar prematuramente, se não fosse correta. Em face desta asserção, tem-se que $n - 2f > 0$ e que para tolerar f faltas é necessário, pelo menos, $2f + 1$ réplicas para manter as propriedades da RME no modelo de faltas por parada.

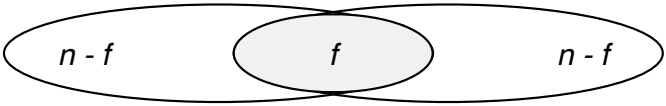


Figura 19 – Interseção de quóruns de leitura e de escrita em RME.

Ao contrário, se considerado o mesmo serviço especificado para o cenário já descrito, porém, implementado por meio de uma RME para BFT, o cenário se modifica completamente. O principal aspecto a ser considerado decorre do comportamento arbitrário manifestado por uma réplica bizantina (LAMPORT; SHOSTAK; PEASE, 1982), o qual impõe uma certa dificuldade em determinar, de maneira convicta, se uma ação é correta ou não. No caso do mesmo serviço descrito pelo parágrafo anterior, torna-se difícil realizar uma verificação sobre as f réplicas que deixam de responder, no sentido de estimar se o comportamento manifestado é oriundo de atrasos reais, ou é faltoso (i.e., se elas são bizantinas ou estão de fato, atrasadas/lentas). Se elas estivessem atrasadas/lentas, então poderia haver f réplicas bizantinas entre o quórum da primeira operação emitida por um cliente (leitura ou escrita). Neste caso, entre a intersecção $n - 2f$ poderia haver f réplicas bizantinas que apenas responderiam ou na operação de leitura, ou na operação de escrita – réplicas bizantinas podem se portar de maneira arbitrária. E para que o serviço seja considerado como correto, a intersecção dos quóruns das diferentes operações deve conter, pelo menos, uma réplica correta, além das f réplicas para as quais não se tem a certeza de que são faltosas ou estão atrasadas/lentas. Logo, para faltas bizantinas tem-se que $n - 2f > f$ ou $n > 3f$, razão pela qual são necessárias pelo menos $3f + 1$ réplicas para assegurar as propriedades da RME, em face deste tipo de faltas.

3.1.2.2 Estado-da-arte em Protocolos de Replicação para BFT

Nesta seção se apresenta uma breve visão de alguns protocolos para BFT verificados na literatura, com destaque para aqueles que po-

dem ser considerados como o estado-da-arte no tema em lide. Neste sentido, são apresentados tanto protocolos baseados em acordo como protocolos baseados em quóruns.

PBFT: Tolerância a Faltas Bizantinas Prática

Protocolo introduzido por Castro e Liskov (CASTRO; LISKOV, 1999), foi um dos projetos pioneiros com foco à viabilidade prática de implementação de sistemas tolerantes a faltas Bizantinas, sendo considerado por muitos o *baseline* da RME para BFT. O PBFT (acrônimo para *Practical Byzantine Fault Tolerance*) encapsula um conjunto de protocolos e mecanismos para a construção de sistemas tolerantes a faltas Bizantinas, a partir de uma RME (SCHNEIDER, 1990), tendo sido concebido com o propósito de ser um protocolo robusto e quando apropriado (e necessário), admite a degradação do desempenho no intuito de manter o serviço em funcionamento mesmo quando há falhas, ataques e/ou intrusões. Além disso, os algoritmos do PBFT admitem uma série de otimizações para prover eficiência ao sistema na ausência de falhas.

Em suma, o protocolo do PBFT opera a partir de um grupo formado por $n \geq 3f + 1$ réplicas, as quais são responsáveis por manter o estado da aplicação/serviço e também por executar o protocolo de acordo/ordenação. O algoritmo de base do PBFT utiliza de uma réplica, denominada primária, para ordenar as requisições que chegam dos clientes ao serviço replicado. As réplicas se comunicam umas com as outras no intuito de assegurar que todas as réplicas corretas executem a mesma sequência de operações, e também para permitir o mascaramento das faltas ocorridas num subconjunto de réplicas. Para garantir as propriedades da RME, o PBFT admite um sistema assíncrono com premissa de sincronismo terminal (DWORK; LYNCH; STOCKMEYER, 1988) no sistema subjacente de comunicação.

O algoritmo é iniciado quando um cliente envia uma requisição às réplicas, e a partir daí são executadas três rodadas de comunicação entre a réplica primária e as réplicas secundárias, as quais são ilustradas na Figura 20 pelas fases `PRE-PREPARE`, `PREPARE` e `COMMIT`. Quando a réplica primária recebe uma requisição de um cliente, ela acrescenta um número de sequência nesta requisição – que indica a ordem na qual aquela requisição deve ser executada pelas réplicas, e envia uma mensagem `PRE-PREPARE` para todas as réplicas contendo a ordem proposta para a requisição. Quando uma réplica secundária recebe a mensagem `PRE-PREPARE`, ela envia como confirmação de recebimento uma nova

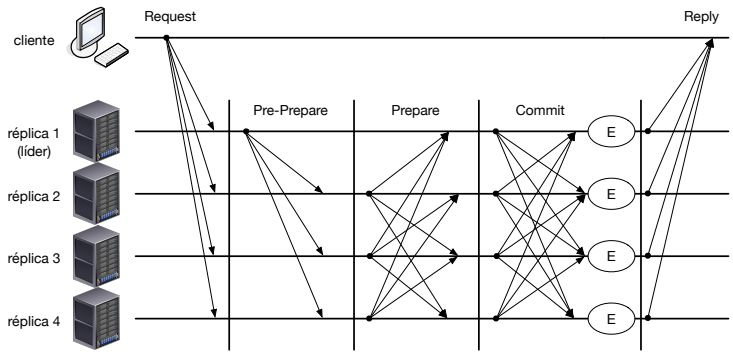


Figura 20 – O protocolo PBFT em execução normal.

mensagem `PREPARE` para todas as demais réplicas. Ao receber um quórum de $2f + 1$ mensagens `PREPARE`, cada réplica assume o compromisso de executar a requisição na ordem proposta pela réplica primária, o que é feito a partir do envio da mensagem `COMMIT` a todas as réplicas. Assim, quando uma réplica recebe um quórum de $2f + 1$ mensagens `COMMIT` ela executa a requisição – ação representada pela elipse com a letra **E** na Figura 20, e responde ao cliente. O cliente aceita o resultado da operação se receber $f + 1$ respostas mutuamente consistentes de diferentes réplicas; do contrário, o cliente retransmite a requisição.

É evidente que, como o protocolo assume sincronia fraca terminal, ele se baseia no uso de temporizadores para verificar possíveis falhas ou atrasos. Por esta razão, um temporizador pode vir a se esgotar em alguma réplica secundária enquanto ela aguarda resposta da primária em alguma rodada de comunicação. Neste caso, para assegurar a progressão do protocolo no caso da suspeita de falhas, o PBFT usa um mecanismo de troca de visão para realizar duas tarefas básicas: a primeira serve para assegurar a evolução do sistema, e a segunda serve para realizar um acordo acerca da ordem das mensagens que ficaram pendentes na visão anterior. Para tanto, o algoritmo de troca de visão admite dois conjuntos de mensagens $\mathcal{P}$ e $\mathcal{Q}$, que contém as mensagens `PREPARE` e `PRE-PREPARE` de visões anteriores, respectivamente.

A troca de visão ocorre quando uma das réplicas secundárias suspeita que na visão v a réplica primária é faltosa, então transpõe sua visão para $v + 1$ e envia uma mensagem `VIEW-CHANGE` para todas as réplicas do conjunto, informando sobre sua troca de visão acompanhada

de seus conjuntos $\mathcal{P}$ e $\mathcal{Q}$. As demais réplicas aceitam a mensagem *VIEW-CHANGE* se e somente se as mensagens contidas em $\mathcal{P}$ e $\mathcal{Q}$ pertencem a visões anteriores a $v+1$, onde atestam tal aceite pelo envio de uma mensagem de notificação `VIEW-CHANGE-ACK` à réplica que solicitou a troca de visão – no caso, a réplica cujo identificador seja $p_i = (v+1)$ **mod** n. Quando a nova réplica primária recebe $2f$ mensagens `VIEW-CHANGE-ACK` válidas de diferentes réplicas secundárias, ela envia uma mensagem `NEW-VIEW` a todas as réplicas secundárias, acompanhada do conjunto das mensagens até então pendentes de ordenação. As réplicas secundárias, por sua vez, instalam a nova visão após verificar a mensagem `NEW-VIEW` da nova réplica primária; e a partir daí o protocolo retoma sua operação normal.

Zyzzyva: Tolerância a Faltas Bizantinas Especulativa

O PBFT pode ser considerado como trabalho propulsor em termos de investigação em BFT, o que é evidenciado pela literatura através dos diversos trabalhos desenvolvidos pós-PBFT (KOTLA et al., 2007; ABD-EL-MALEK et al., 2005; COWLING et al., 2006; GUERRAOUI et al., 2010). A maioria destes trabalhos têm o objetivo centrado em torno de otimizações, sob certas condições, no que diz respeito à replicação BFT. O Zyzzyva (KOTLA et al., 2007) foi uma das primeiras propostas com foco à eficiência na implementação de RME para BFT, ao mesmo tempo que objetiva a confiabilidade e o alto-desempenho. O Zyzzyva consiste em um protocolo de RME otimista, que adota uma abordagem que os autores denominam de execução especulativa, na qual a ordenação das requisições é realizada em apenas três passos de comunicação.

Por ser um protocolo otimista, o Zyzzyva adota a noção de execução especulativa (LAMPSON, 2006) a fim de obter um alto desempenho no processamento de requisições em uma RME para BFT. No protocolo do Zyzzyva, o termo especulação se refere ao fato de que, em casos normais onde não há faltas, todas as réplicas corretas executarão as requisições na ordem imposta pela réplica primária. Nesta abordagem, igualmente como ocorre no PBFT, uma réplica exerce o papel de primária e será a responsável por determinar a ordem de execução das requisições dos clientes no grupo de réplicas. No entanto, devido ao uso da abordagem especulativa, não há a necessidade das réplicas chegarem a um acordo acerca da ordem de execução das requisições, sendo necessário apenas delegar a uma réplica (p. ex.: a primária) a imposição da ordem de execução das requisições, e das demais confiarem nela.

Numa execução do Zyzzyva para o caso livre de falhas, conforme

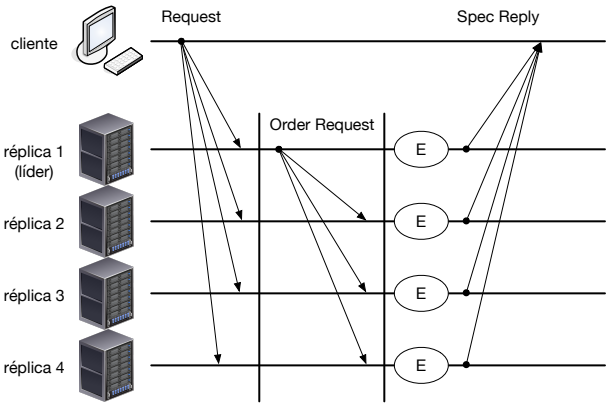


Figura 21 – O protocolo Zyzzyva em execução livre de falhas.

ilustrado pela Figura 21, o cliente envia uma requisição para todas as réplicas do sistema, porém, a réplica primária atribui à requisição um número de sequência e o encaminha para as demais réplicas. As réplicas secundárias, por sua vez, ao receber a ordem indicada pela réplica primária executam a requisição naquela ordem e enviam ao cliente a respectiva resposta. Com isso, o protocolo evita todas as fases de acordo necessárias para determinar a ordem de execução da requisição, tal como ocorre no PBFT. Neste caso, o cliente que opera como uma espécie de aprendiz, dá por completa a execução da requisição em dois casos: (i) se receber $3f + 1$ respostas, ou; (ii) se receber entre $2f + 1$ e $3f$ respostas – em ambos os casos são consideradas apenas respostas mutuamente consistentes. Se nenhuma das condições especificadas nos casos (i) e (ii) forem satisfeitas, o cliente orienta as réplicas a iniciarem um procedimento de recuperação, o que pode induzir o sistema a uma troca de visão.

Por outro lado, o fato de o protocolo ser otimista abre precedentes para que as réplicas estejam temporariamente inconsistentes entre si. Neste caso, o cliente é quem observa tais inconsistências e auxilia as réplicas a convergir para uma ordem atômica acerca da execução das requisições. Isto só é possível porque as réplicas anexam informações do histórico do processamento de requisições às respostas, e deste modo o cliente tem subsídios para determinar se as respostas e o histórico são estáveis e podem ser validados. A partir das respostas recebidas

das réplicas, o cliente tem condições de determinar se um resultado é correto, porém, uma réplica não sabe se as atualizações efetuadas sobre seu estado estão de acordo com o das outras réplicas até que uma validação explícita seja recebida. Esta validação é realizada periodicamente pelas réplicas. Além disso, outro aspecto relacionado à execução especulativa é que as réplicas não sabem se as outras estão executando as requisições na mesma ordem. Por conseguinte, o estado que também é especulativo deve manter um histórico de todos os estados anteriores, a fim de possibilitar uma operação de reversão de estado quando são verificadas inconsistências durante o processo de execução especulativa.

Chain: Protocolo BFT Anulável

A noção de anulabilidade em protocolos de RME para BFT foi introduzida por Guerraoui *et al.* (2010), no intuito de reduzir a complexidade de especificação e implementação de protocolos BFT por meio de uma RME para BFT modular. O *Chain* (GUERRAOUI et al., 2010) é um dos subprotocolos que faz parte de um protocolo modular de RME para BFT denominado **Aliph** (GUERRAOUI et al., 2010). Similar ao que ocorre no Zyzzyva, ele também adota uma abordagem especulativa (ou otimista) para a execução de operações em uma RME para BFT, mas com algumas especificidades. No *Chain* as réplicas são organizadas como uma cadeia, de modo a implementar um *pipeline* eficiente que visa a obtenção de alto desempenho em execuções livres de falhas. Como as réplicas são organizadas como uma cadeia, esta cadeia depende de duas réplicas distintas, as quais são denominadas **cabeça** (*head*) e **cauda** (*tail*), respectivamente.

O *pipeline* de execução do *Chain* é ilustrado pela Figura 22. Como mencionado, trata-se de um protocolo anulável, isto é, onde é possível que operações não sejam efetivadas em casos específicos (p. ex.: com faltas). Neste caso, o *Chain* depende de um protocolo de backup similar ao PBFT, para lidar com situações nas quais o *pipeline* não consegue obter êxito na execução de operações. O protocolo funciona da seguinte maneira, o cliente envia uma requisição apenas para a réplica que exerce o papel de **cabeça** na cadeia, e esta réplica por sua vez, tem a prerrogativa de ordenar a requisição do cliente, e portanto, de atribuir o número de sequência para esta. Em seguida, a réplica **cabeça** envia a requisição ordenada para a próxima réplica da cadeia, que, após executar a requisição, encaminha a mesma para a próxima réplica até a requisição chegar à réplica que exerce o papel de **cauda** na cadeia.

Mais precisamente, ao receber a requisição da réplica predeces-

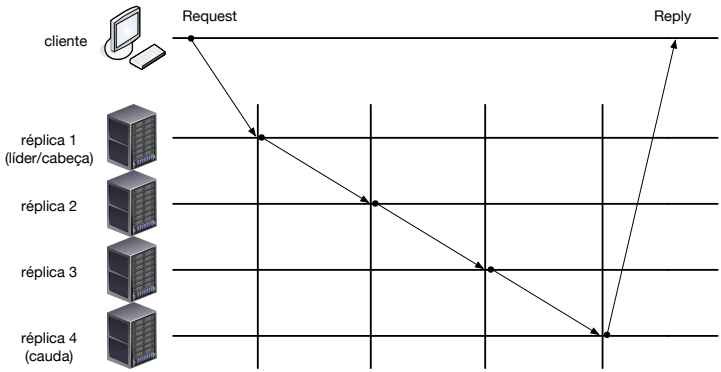


Figura 22 – O *pipeline* de execução do protocolo Chain.

sora na cadeia, a réplica em questão executa a requisição, a acrescenta em seu histórico local e encaminha a requisição para a próxima réplica na cadeia até chegar ao final da cadeia, e por conseguinte completar o *pipeline*. Por fim, a réplica que exerce o papel de **cauda** envia a resposta ao cliente. Importante salientar que as últimas $f + 1$ réplicas incluem o resumo criptográfico (i.e., um *hash*) de seus históricos junto as respectivas requisições enviadas para as réplicas sucessoras na cadeia, e estes resumos são incluídos junto à resposta enviada ao cliente pela réplica que exerce o papel de **cauda**. Com isso, se os resumos são consistentes entre si, o cliente valida a requisição e conclui o protocolo. De outro modo, se os resumos não são consistentes, o cliente recorre a um protocolo de backup não anulável, tal como o PBFT (CASTRO; LISKOV, 1999) para validar a requisição.

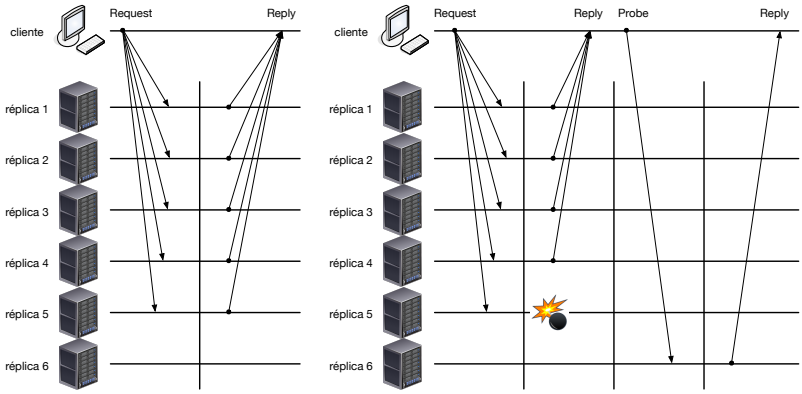
O *Chain* é um protocolo que apresenta o melhor desempenho dentre os protocolos de RME para BFT da literatura, desde que a execução seja síncrona e não ocorram falhas no decurso do protocolo.

Protocolo *Query-Update*

O protocolo *Query-Update* (ou simplesmente Q/U) (ABD-EL-MALEK et al., 2005) é uma alternativa em relação às soluções de RME para BFT baseadas em protocolos de acordo, já que ele consiste em uma solução de RME para BFT baseada em quórum, e portanto, diferente dos protocolos PBFT e Zyzzyva, que são baseados em acordo. O Q/U é um protocolo no qual a execução das requisições é realizada em apenas

uma rodada de comunicação. Por se tratar de um protocolo baseado em quóruns, um dos propósitos do Q/U é permitir a construção de serviços tolerantes a faltas Bizantinas de maneira eficiente e otimista. De outro modo, o Q/U tolera um máximo de f réplicas faltosas em um conjunto de pelo menos $5f+1$ réplicas, sendo os quóruns formados por $4f+1$ réplicas.

A essência do protocolo Q/U é, na realidade, uma terminologia aplicada aos objetos que são replicados em cada uma das réplicas do sistema (nos quóruns). Assim, os objetos que são replicados são acessados por meio de interfaces, que são compostas por métodos deterministas. Os métodos de acessos aos objetos em modo de leitura são denominados *Queries*, enquanto que os métodos que permitem a modificação de objetos são denominados *Updates*, daí a origem do nome *Query/Update*. Estes métodos disponibilizados pelos objetos Q/U apenas recebem argumentos e retornam respostas, onde o protocolo assegura que todas as operações emitidas ao sistema sejam serializadas. Os padrões de comunicação do protocolo Q/U tanto em cenários normais como em cenários com falhas são ilustrados pela Figura 23.



(a) Execução normal. (b) Execução com falhas.

Figura 23 – Cenários de execução do protocolo Q/U.

Em suma, no Q/U as réplicas corretas executam localmente as requisições dos clientes de maneira otimista, isto é, sem a realização de um acordo explícito entre elas acerca de uma ordem. Os clientes têm participação ativa na execução do protocolo, já que eles fazem um *cache* do histórico contido nas respostas enviadas pelas réplicas,

para anexá-las nas requisições posteriores. A operação do Q/U se dá da seguinte maneira: os clientes realizam operações em um objeto por meio do envio de requisições a um quórum de réplicas. Uma réplica que recebe a requisição e a aceita, invoca um método em sua versão local do objeto solicitado na requisição. Note que o Q/U se baseia no uso de diversas versões de objetos para implementar a característica que o torna um protocolo otimista. Para tanto, os objetos Q/U são replicados em todos as réplicas do sistema, onde cada réplica mantém as versões dos objetos associadas a uma estampilha lógica, constituindo assim o histórico daquela réplica.

Em casos normais, onde não há contenção nem erros, as operações são executadas em uma única rodada e o cliente entrega o resultado à aplicação após receber $4f + 1$ respostas de diferentes réplicas. Por outro lado, quando dois ou mais clientes acessam as réplicas simultaneamente, pode ocorrer que nenhum deles consiga completar um quórum, e para tanto o protocolo emprega o uso da técnica de *backoff* exponencial (CHOCKLER; MALKHI; REITER, 2001), que é semelhante à técnica utilizada para resolver problemas de colisão em canais multiacesso. Contudo, esta aproximação permite que as réplicas evoluam para estados incorretos, uma vez que diferentes réplicas aceitam diferentes operações sem que sejam formados quóruns. Como no Q/U as réplicas nunca se comunicam, a resolução de eventuais divergências verificadas nos estados dos objetos replicados é delegada aos clientes.

É digno de nota que embora o Q/U necessite de um número significativamente menor de mensagens e também de rodadas de comunicação em relação ao BFT, em situações onde há grande concorrência devido à emissão de múltiplas requisições correntes ele é suscetível à *livelocks*. Ademais, pelas características inerentes ao protocolo, nota-se que o Q/U opera com dificuldade em situações onde há muita contenção, já que os acessos concorrentes iniciam operações de reparação e estas por sua vez, bloqueiam o acesso às réplicas.

Protocolo Quórum Híbrido

O *Hybrid Quorum Protocol* (ou simplesmente HQ) (COWLING et al., 2006) consiste em uma solução que combina as vantagens das soluções de RME para BFT baseadas em acordo, mais precisamente no PBFT, e nas baseadas em quóruns, nomeadamente no Q/U. A combinação engenhosa realizada pelo HQ visa, principalmente, lidar com algumas limitações de desempenho verificadas tanto no PBFT como no Q/U. O HQ é fundamentado em dois princípios básicos, dos quais o

primeiro está ligado à redução do número de réplicas necessárias para formar os quóruns, a despeito do custo verificado para o protocolo Q/U (i.e., $5f + 1$); e o segundo reside no fato de o mesmo prover ganhos de escalabilidade, a despeito das necessidades de comunicação entre todos os processos nas abordagens baseadas em acordo. Neste caso, em situações onde não há contenção, o HQ utiliza um protocolo leve de quóruns, em que as leituras necessitam de apenas uma rodada de comunicação (entre cliente e réplicas) e as escritas necessitam de duas rodadas. Para as operações de escrita, o HQ adota certificados para garantir que as operações serão ordenadas de forma apropriada. De outro modo, quando há contenção o HQ usa o PBFT (CASTRO; LISKOV, 1999) para ordenar as requisições concorrentes de maneira eficiente.

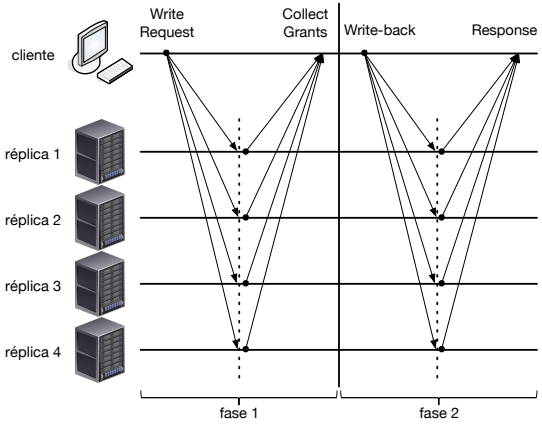


Figura 24 – Execução de escritas no protocolo HQ.

O HQ, em execuções normais e na ausência de escritas conflitantes é executado por meio de duas fases em um protocolo de quórum, a partir de $3f + 1$ réplicas (vide Figura 24). Ao receberem uma requisição de um cliente, as réplicas escolhem de maneira otimista uma ordem para aquela requisição (i.e., um *grant*) e notificam a situação ao cliente. O cliente por sua vez, coleta um quórum de pelo menos $2f + 1$ *grants* e inicia a segunda rodada do protocolo, onde ele envia os *grants* recebidos e coletados em uma mensagem de *write-back*. No caso, ao receber a mensagem de *write-back* as réplicas têm condições de detectar possíveis contenções a partir da observação do conjunto de *grants* contidos na mensagem de *write-back*. Se as réplicas não verificaram situação de

contenção, elas executam a operação do cliente e o enviam a resposta da operação.

No caso tenha havido contenção, o protocolo de ordenação do PBFT é acionado por meio de um *proxy*, que na realidade corresponde à réplica primária do protocolo PBFT. O *proxy* tem a finalidade de coletar de maneira oportunista as requisições em conflito, combiná-las em um lote de resolução de conflitos, e enviá-las para o módulo responsável pela execução do PBFT para ordenação, e por conseguinte, para linearização. Após a ordenação das requisições conflitantes pelo protocolo de acordo do PBFT, as réplicas validam o lote de conflito da seguinte maneira. Primeiramente elas verificam se a resolução de conflitos foi, de fato, necessária, e se todas as requisições conflitantes são válidas. Segundo, se as condições anteriores foram satisfeitas, as réplicas executam o lote válido de requisições em conflito em uma ordem consistente; do contrário, o lote de requisições é rejeitado e elas passam a suspeitar de falha no *proxy* (i.e., a primária do PBFT), em razão de ter gerado um lote não válido, e então iniciam o processo de troca de visão do PBFT para substituir a réplica primária do PBFT, a qual fará o papel do novo *proxy*.

Para o caso de operações de leitura, estas são executadas em apenas uma rodada de comunicação, onde o cliente envia a requisição às réplicas, e elas respondem com o resultado para o respectivo cliente. Caso os resultados sejam coerentes e consistentes, o cliente entrega o resultado para a aplicação; do contrário, um processo de recuperação é iniciado – maiores detalhes acerca deste processo podem ser vistos em (COWLING et al., 2006). Em comparação ao Q/U, o HQ utiliza o acordo bizantino do PBFT ao invés do *backoff* exponencial adotado pelo Q/U. Os resultados obtidos pelos autores mostram que o HQ melhora significativamente o desempenho em relação ao PBFT em cenários de baixa concorrência, enquanto a resolução de conflitos demonstra resultados piores em termos de latência, se comparado ao Q/U.

3.1.3 Comparação dos Protocolos de Replicação BFT

Na seção anterior foram apresentadas as principais soluções de RME para BFT encontradas, as quais podem ser destacadas como o estado-da-arte no tema em lide. Tais soluções são desenvolvidas tomando como base as duas abordagens mais usuais para a construção de protocolo de RME, que são a abordagem baseada em quóruns e a abordagem baseada em acordo. E para concluir a discussão e permitir

uma maior reflexão acerca do estado-da-arte em RME para BFT, a tabela 3 enumera alguns resultados acerca de alguns aspectos analisados em cada um dos protocolos apresentados ao longo da seção.

De um lado, o PBFT (CASTRO; LISKOV, 1999), o Zyzzyva (KOTLA et al., 2007) e o Chain (GUERRAOUI et al., 2010) são conhecidos por serem os protocolos de RME para BFT mais eficientes em termos de vazão (i.e., *throughput*) sob condições de carga elevada. De outro lado, os protocolos baseados em quórum como o HQ (COWLING et al., 2006) e Q/U (ABD-EL-MALEK et al., 2005) são os que apresentam a menor latência sob condições de carga mais baixas. Porém, quando a condição de carga aumenta, estes protocolos baseados em quórum não conseguem obter um bom desempenho, pois muitas vezes exigem que as réplicas sejam reconciliadas, o que resulta em demasiada degradação de desempenho. Um feito interessante e que merece destaque é o fato de o HQ otimizar a resiliência de $3f + 1$ em um ambiente baseado em quórum.

Tabela 3 – Comparação analítica do estado-da-arte em BFT

Protocolo RME/BFT	**Aspectos analisados**				
	Ordenação	**# Réplicas**	**# Respostas**	**Latência**	**Mensagens**
PBFT	Primária	$3f + 1$	$2f + 1$	4	$O(n^2)$
Zyzzyva	Primária	$3f + 1$	$3f + 1$	3	$O(n)$
Chain	Otimista (*pipeline*)	$3f + 1$	$f + 1$	4	$O(1)$
Q/U	Quórum	$5f + 1$	$4f + 1$	2*	$O(n)$
HQ	Primária + Quórum	$3f + 1$	$2f + 1$	4	$O(n)$

Além disso, os protocolos baseados em quórum fornecem complexidade inferior aos protocolos baseados em acordo, isto é, $O(n)$ em vez de $O(n^2)$, com exceção do protocolo *Chain* que é baseado em acordo, mas com algumas peculiaridades em relação à abordagem clássica. No caso do Zyzzyva, embora seja baseado em acordo, o padrão de comunicação por ele adotado evita comunicações todos-para-todos e por esta razão ele tem melhor desempenho em relação H/Q em situações de contenção – o H/Q usa o protocolo do PBFT para este caso. De outro modo, o Zyzzyva é o único que requer a resposta de todas as réplicas para considerar uma requisição estável e correta.

É digno de nota que os protocolos apresentados nesta seção formam um arcabouço teórico em termos de embasamento necessário para a construção dos protocolos desenvolvidos no âmbito desta tese. Na próxima seção serão apresentados os trabalhos encontrados na litera-

tura que tratam de aspectos similares aos investigados por esta tese.

3.2 TRABALHOS RELACIONADOS

Nesta seção são apresentados alguns dos trabalhos correlacionados a esta tese, encontrados na literatura. Como esta tese visa apresentar soluções para alguns problemas relacionados ao processamento e terminação de transações sob a sujeição de faltas bizantinas, os trabalhos relacionados estão divididos em duas categorias. A Subseção 3.2.1 apresenta as soluções encontradas na literatura que tratam do processamento e terminação de transações a partir de protocolos baseados em replicação, enquanto que a Subseção 3.2.2 apresenta alguns trabalhos que tratam da terminação de transações através de protocolos de validação atômica.

3.2.1 Soluções Baseadas na Replicação de Bancos de Dados

A replicação de dados pode ser vista como uma estratégia chave para prover tolerância a faltas e eficiência, em ambientes onde as aplicações requerem um alto grau de disponibilidade e outros atributos relacionados à confiabilidade. É notório que a replicação de dados em múltiplos locais (i.e., réplicas) amplia a resiliência do ambiente computacional, ao passo que as réplicas disponíveis podem assumir a carga de trabalho daquelas que porventura vierem a falhar. Por outro lado, a replicação de dados tem um preço considerável, que consiste principalmente no custo associado à manutenção da consistência do estado dos bancos de dados locais das réplicas.

Não obstante, se levado em consideração os problemas reais causados pela ocorrência de faltas bizantinas, tal como visto na Seção 3.1, é imperioso que os bancos de dados sejam capazes de efetuar o processamento de transações de maneira confiável, provendo também, disponibilidade, corretude e desempenho aceitáveis. A replicação de dados é o meio mais usual e factível para se obter estes requisitos, de maneira análoga ao que ocorre com os sistemas não transacionais tolerantes a faltas bizantinas (CASTRO; LISKOV, 1999; KOTLA et al., 2007; COWLING et al., 2006). Todavia, diferente do que ocorre nos sistemas não transacionais, a replicação de bancos de dados ainda é um problema complexo e desafiador, quando se trata da manutenção da consistência dos dados replicados (GRAY et al., 1996), em face de faltas bizantinas.

É digno de nota, que soluções para tolerar faltas de qualquer natureza em bancos de dados também adotam a replicação como ponto de partida (GASHI; POPOV; STRIGINI, 2007; VANDIVER et al., 2007; PREGUIÇA et al., 2008; GARCIA; RODRIGUES; PREGUIÇA, 2011; PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO, 2011). Porém, a despeito da replicação ser algo bem estabelecido na literatura de banco de dados, dependendo da(s) classe(s) de faltas tolerada(s), outros problemas surgem no que se refere à especificação da solução baseada em replicação. Em se tratando de faltas bizantinas, embora pesquisas envolvendo tolerância a faltas bizantinas em sistemas computacionais tenham tido grande ascendência na última década, poucas iniciativas foram desenvolvidas para lidar com esta classe de faltas em bancos de dados. Nesta seção são apresentados os trabalhos que foram desenvolvidos neste sentido.

3.2.1.1 Replicação de Máquina de Estados para Bancos de Dados

O protocolo ***D**ata**B**ase **S**tate **M**achine Replication*, doravante referenciada apenas por DBSM, introduzido na literatura por Pedone, Guerraoui e Schiper (2003) pode ser visto como um importante trabalho desenvolvido no âmbito de replicação de banco de dados nos últimos anos, visto que ele introduziu uma nova abordagem para prover confiabilidade e tolerância a faltas em bancos de dados. A DBMS visa obter bom desempenho no processamento de transações, ao mesmo tempo em que provê garantias fortes de consistência para o banco de dados. Em suma, a DBMS consiste na união da abordagem de Replicação de Máquina de Estados (RME) (SCHNEIDER, 1990) com a estratégia de replicação de bancos de dados *Deferred Write/Update* (BERNSTEIN; HADZILACOS; GOODMAN, 1987). A grande contribuição deste trabalho para a literatura está na demonstração de que, para obter a serialização das transações executadas não é requerida uma ordem total para a execução, mas apenas no que diz respeito à terminação das transações. Destarte, os autores especificaram um protocolo de terminação baseado em uma máquina de estados, e provaram que é possível serializar as transações a partir dele.

A estratégia introduzida pela DBSM consiste em um ambiente de banco de dados replicado a partir de um *cluster* de servidores, onde não há nenhum tipo de sincronização, tampouco mecanismos de transações distribuídas (p. ex.: bloqueios distribuídos). No caso, a abordagem se apoia na estratégia *deferred update*, que permite processar de maneira otimista as transações com o uso de um protocolo de terminação ba-

seado em uma máquina de estados. O modelo assume que cada réplica mantém uma cópia completa da base de dados e que as transações são executadas localmente nas réplicas, de acordo com esquema de bloqueio em duas fases rigoroso (vide Seção 2.1.3.1). As transações de somente-leitura são executadas localmente e as transações de atualização não requerem nenhuma sincronização entre réplicas até a validação das mesmas. Assim, durante a execução de uma transação o cliente interage apenas com uma réplica, a primária. Quando o cliente solicita a validação da transação, as atualizações realizadas apenas na primária são propagadas para as demais réplicas, e a transação é certificada para verificar se ela cumpre os requisitos para sua serialização, e então validada. Se a certificação determinar que transação não pode ser serializada, ela é imediatamente anulada.

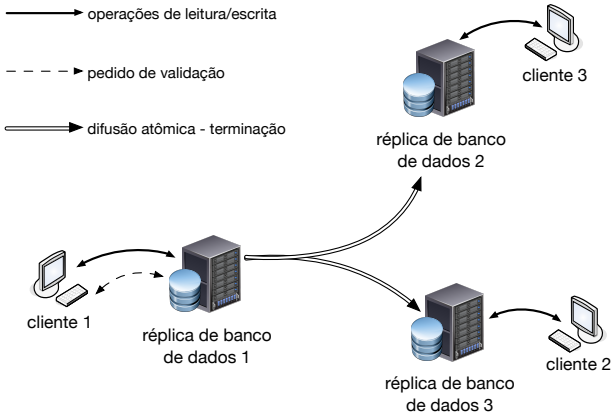


Figura 25 – Dinâmica de funcionamento do protocolo DBSM.

Um dos aspectos mais importantes introduzidos pela DBSM foi a substituição do tradicional protocolo de validação atômica pelo protocolo de difusão atômica (DÉFAGO; SCHIPER; URBÁN, 2004). No caso, como cada réplica se comporta como uma máquina de estados, com o uso do protocolo de difusão atômica, a entrega das mensagens de terminação da transação ocorrem na mesma ordem em todas as réplicas. Por conseguinte elas processam a certificação e a validação na mesma ordem, e com isso todas as réplicas alcançam o mesmo estado final pós-validação. No entanto, para se obter este feito, todas as réplicas aplicam as atualizações da transação na ordem em que a mensagem

foi entregue – exceto a primária que já executou de maneira otimista mas ainda não as validou, e a concessão dos bloqueios segue a ordem de entrega da mensagem de certificação das transações. A certificação das transações entregues segue o mesmo princípio do teste de Kung e Robinson (1981), onde é verificado se as atualizações efetuadas por transações concorrentes mas já validadas não se intersectam com as leituras realizadas pela transação em processo de certificação/validação. O propósito da certificação é evitar que transações cujas operações são conflitantes violem a consistência do banco de dados.

Embora na DBSM o processamento de transações ocorra de maneira otimista, o protocolo procura reduzir a taxa de anulação de transações por meio de uma versão melhorada do processo de certificação, cuja ideia é tentar reordenar as transações que solicitam a validação, de modo a aumentar a possibilidades de validação através da construção de uma ordem abstrata diferente da ordem real de entrega. Esta ordem é disposta em uma lista de serialização abstrata, que é construída a partir da observação dos possíveis conflitos sobre as transações já certificadas. No caso, o algoritmo tenta inserir a transação que está em processo de certificação em uma posição desta lista, de modo que seja possível serializá-la antes de decidir por sua anulação. É digno de nota que a DBSM foi especificada para tolerar apenas faltas por parada, de modo que faltas bizantinas não são consideradas, tampouco suportadas.

3.2.1.2 Aplicação do Acordo Bizantino em Bancos de Dados

O estudo pioneiro no âmbito de faltas bizantinas e bancos de dados foi realizado há várias décadas (GARCIA-MOLINA; PITTELLI; DAVIDSON, 1984). Este trabalho pode ser considerado como uma espécie de *position paper*, já que o mesmo não apresentou nenhuma proposição e os autores apenas levantaram algumas hipóteses acerca da aplicação de um algoritmo de acordo bizantino (LAMPORT; SHOSTAK; PEASE, 1982) genérico em sistemas de banco de dados. Pouco mais tarde, os mesmos autores evoluíram este estudo e publicaram um novo trabalho com mais detalhes acerca do problema relacionado à faltas bizantinas em bancos de dados (GARCIA-MOLINA; PITTELLI; DAVIDSON, 1986). Este último detalhou um esquema de replicação para tolerar faltas bizantinas em bases de dados transacionais, a partir de uma contextualização do problema do consenso (PEASE; SHOSTAK; LAMPORT, 1980) com faltas bizantinas em ambientes síncronos. Nesta contextualização os autores

estenderam a discussão e apresentaram a proposição de uma solução para prover a confiabilidade no processamento de dados sobre uma base de dados. Para tanto, algumas hipóteses foram adotadas para a proposição daquela solução: os autores definem que os usuários têm de lidar com o fato de que as transações nem sempre serão executadas, porém, a solução garante que tudo o que for executado, o será de maneira correta.

Para a solução proposta, os autores estabeleceram algumas condições elementares de correção (*safety*), para que se possa assegurar as propriedades definidas para o sistema. Tais condições são: (*i*) os usuários do sistema devem obter os mesmos resultados que obteriam em um sistema ideal onde não ocorrem falhas; (*ii*) se uma transação é corretamente submetida, então ela fará parte do escalonamento final correto; (*iii*) o tempo para realizar a validação de uma transação é limitado, a fim de que o sistema seja forçado a decidir acerca da execução ou não de uma determinada transação, num período de tempo definido. Não obstante, se uma decisão pela validação foi obtida, ela será irreversível. Além das condições, os autores especificaram algumas premissas para reforçar a obtenção da solução, são elas: (*i*) todas as transações contém informações de autenticação dos usuários, de modo que transações submetidas por usuários não autorizados são descartadas; (*ii*) cada transação é originária de um único usuário; (*iii*) as funções de entrada e saída do sistema são realizadas por sítios (ou nós) específicos de entrada e saída, que não são os mesmos sítios que efetuam o processamento das transações (e dos dados). Esta última premissa é definida especificamente para estabelecer que os sítios (ou réplicas) responsáveis pelo processamento da transação nunca emitirão transações no sistema.

A partir das condições e premissas especificadas, os autores discutiram e apresentaram as características da solução proposta, a qual trata de um esquema de replicação de bancos de dados. Para esta replicação é assumido que são mantidas pelos menos $2f+1$ cópias completas da base de dados, sendo cada uma em uma réplica distinta, de modo que seja possível identificar a corretude dos resultados enviados pelo sistema a partir de um mecanismo de voto majoritário – a(s) resposta(s) proveniente(s) de uma maioria de réplicas. Neste caso, é assumido que todas as réplicas não faltosas executam exatamente o mesmo conjunto de transações, e numa mesma ordem. Do contrário, o estado das réplicas poderia vir a divergir e por isso não seria possível determinar os resultados pelo voto majoritário. E é aí que entra o **acordo bizantino** utilizado na solução proposta.

Uma observação bastante peculiar sobre a solução proposta de-

corre do fato que o trabalho/artigo não apresenta nenhuma consideração a respeito da execução concorrente de transações, o que implica que a solução não é – ou seria à época em que foi desenvolvida – factível de uso em sistemas de bancos de dados reais. Além do mais, à época em que o trabalho foi proposto, o conceito de tolerância a faltas bizantinas não era muito bem estabelecido e compreendido, principalmente em se tratando do limite estabelecido em $3f + 1$ réplicas necessárias para realizar um acordo bizantino, das quais até f podem falhar (vide Seção 3.1.1). E tomando em conta estes detalhes, há um forte indício de que a solução nunca foi empregada na prática, já que usava apenas $2f + 1$ réplicas para obter o acordo bizantino, e também por não ter apresentado nenhum indício quanto à sua implementação. Ademais, o trabalho tem caráter bastante seminal, e discorreu apenas sobre a integridade e consistência das respostas de transações em um banco de dados, a despeito de faltas bizantinas. Não obstante, nenhuma discussão acerca do processamento concorrente de transações foi elucidada pelo trabalho.

3.2.1.3 Uso de Diversidade como Suporte à Replicação de Bancos de Dados

Um dos primeiros trabalhos a reportar a caracterização de faltas bizantinas em bancos de dados relacionais foi o estudo realizado por Gashi *et al.* (2004, 2007). Neste trabalho, os autores discutiram o problema de se tolerar faltas bizantinas em bancos de dados relacionais, além de terem investigado e documentado uma séria de *bugs* presentes em SGBDs comerciais, os quais manifestaram faltas bizantinas. O objetivo deste trabalho não estava centrado na resolução do problema relacionado ao processamento de transações a despeito de faltas bizantinas em bancos de dados, mas em demonstrar a importância do uso de diversidade (RANDELL, 1975; OBELHEIRO; BESSANI; LUNG, 2005) para a replicação de dados em SGBDs.

A partir da caracterização de faltas apresentada, os autores estimularam o uso de diversidade na replicação de dados em SGBDs, por meio do uso de implementações distintas de uma mesma especificação, com o objetivo de reduzir a correlação entre os erros verificados e reportados em seu estudo. Não obstante, os autores demonstraram que, embora a diversidade seja um tema discutido há, pelo menos, trinta anos, ela é uma técnica pouco utilizada devido aos custos associados. Todavia, o custo associado ao desenvolvimento de versões distintas vem caindo progressivamente, em decorrência do sucesso de diversos produ-

tos em várias indústrias, além do crescimento do mercado para componentes *off-the-shelf*. A utilização destes componentes constitui uma alternativa viável e de baixo custo para os integradores de sistemas que necessitam aumentar a confiabilidade de seus produtos e serviços.

Não obstante, o trabalho também discorre sobre diversas limitações verificadas pelos autores quanto à replicação de dados em SGBDs comerciais. A partir daí, os autores propõem uma solução baseada em *middleware* para prover algumas funcionalidades básicas de tolerância a faltas em SGBDs. O *middleware* é especificado como um nó de tolerância a faltas, e é constituído por pelo menos 2 (duas) réplicas de bancos de dados. Assim, a conectividade entre os clientes de uma aplicação com o *middleware* é implementada a partir da API padrão JDBC/ODBC, e a conectividade do *middleware* com as réplicas de bancos de dados se dá através dos protocolos nativos disponibilizados pelos SGBDs escolhidos.

Por outro lado, embora o estudo aponte aspectos de grande relevância para o contexto de implementação de serviços confiáveis de bancos de dados, por meio do uso da diversidade, o mesmo não apresenta soluções para todos os problemas verificados pelos autores. Por exemplo, o *middleware* trata alguns problemas de consistência oriundos de possíveis falhas no processamento de transações, por meio da aplicação do voto majoritário das respostas recebidas das réplicas. E é com base neste voto que a solução detecta se alguma réplica falhou durante o processamento de uma transação. Do mesmo modo, o trabalho não discute aspectos relacionados à execução concorrente de transações sobre o *middleware* proposto, tampouco se ele permite que transações concorrentes preservem escalonamentos serializáveis sobre as réplicas.

3.2.1.4 Escalonamento Baseado em Barreira de Validação para Replicação Tolerante a Falta Bizantinas de Bancos de Dados

O protocolo de escalonamento baseado em uma barreira de validação (do inglês, *Commit Barrier Scheduling* – CBS) (VANDIVER et al., 2007) foi o primeiro protocolo BFT para o processamento de transações concorrentes em bancos de dados relacionais, tendo sido pioneiro a tratar dos aspectos de concorrência e consistência para transações, em ambiente de bancos de dados sujeitos a faltas bizantinas. O CBS é um protocolo que faz parte de um sistema denominado HRDB (*Heterogeneous Replicated Database*), que implementa uma solução bastante completa para a replicação de bases de dados tolerantes a faltas bi-

zantinas. O protocolo CBS visa assegurar a corretude de transações concorrentes executadas por múltiplos clientes. Para isso, o CBS restringe a ordem na qual as operações são enviadas para as réplicas, a fim de evitar escalonamentos conflitantes e preservar um nível de concorrência satisfatório, sob diversas condições de carga.

No HRDB, os clientes não interagem diretamente com as réplicas de banco de dados, em vez disso, eles se comunicam com uma entidade centralizada denominada *Shepherd*, que atua como uma espécie de *front-end* para as réplicas e tem as tarefas de coordená-las e de conduzir o protocolo CBS. Este é o principal ponto de fraco da proposta, pois para prover a corretude das transações, o HRDB assume que o *Shepherd* é confiável, e portanto, não pode falhar de maneira bizantina – mas apenas por parada. Os autores afirmam que a hipótese é bastante razoável, tendo em vista que a complexidade e quantidade de código existentes no *Shepherd* são de ordens de magnitude muito inferiores às das réplicas. Contudo, considerando que o *Shepherd* pode falhar por parada, para que o sistema se mantenha em operação é necessário que esta entidade seja também replicada, para permitir que quando a primária vier a falhar um dos *backups* assumirá o seu lugar. O HRDB se baseia na abordagem proposta no trabalho de Yin *et al.* (2003), que utiliza apenas $2f + 1$ réplicas de bancos de dados para executar as operações das transações. Isto só é possível pelo fato do HRDB não executar um protocolo de acordo bizantino, já que depende do *Shepherd* e a réplica primária é quem determina a ordem de validação das transações.

Em se tratando da base do HRDB, isto é, do CBS, uma das réplicas é designada como primária e tem a função de executar as operações antes enviá-las às réplicas secundárias. A ordem na qual as transações são validadas na réplica primária determina a ordem de referência utilizada para a validação nas demais réplicas. Portanto, cabe ao CBS fazer com que a ordem de validação a ser efetuada pelas réplicas secundárias seja equivalente à ordem de referência. Quando o coordenador recebe uma operação de um cliente, ele imediatamente envia-a ao gestor da réplica primária, e a réplica primária por sua vez, executa as operações das transações utilizando o seu mecanismo de controle de concorrência interno, mais precisamente o esquema de bloqueio em duas fases rigoroso, uma condição necessária para assegurar as propriedades de correção do protocolo CBS.

Quando uma resposta é enviada ao coordenador, a operação correspondente é submetida imediatamente aos gestores das réplicas secundárias, aos quais cabe a manutenção de um conjunto que contém as

operações a serem executadas nas réplicas. Também cabe aos gestores das réplicas secundárias submeter as operações aos respectivos bancos de dados em condições que permitam obter o máximo de concorrência possível, com observância para algumas regras: (*i*) uma transação deve ser processada sequencialmente; (*ii*) as validações das transações devem ser processadas em uma mesma ordem; (*iii*) se uma operação O de uma transação T_i for executada após a validação de uma transação T_j na réplica primária, ela só será submetida para as réplicas secundárias após elas terem validado a transação T_j. Por outro lado, para aumentar a eficiência do sistema, as respostas são enviadas aos clientes pelo coordenador, tão logo elas sejam recebidas da réplica primária, a despeito da necessidade de, pelo menos $f+1$ respostas para a validação da transação. No caso de uma falha da réplica primária, esta é substituída por uma das réplicas secundárias a partir de um protocolo de mudança de visão, visando assegurar a progressão (*liveness*) do sistema.

3.2.1.5 Replicação de Bancos de Dados Tolerante a Faltas Bizantinas com Suporte ao *Snapshot Isolation*

O sistema Byzantium (PREGUIÇA et al., 2008; GARCIA; RODRIGUES; PREGUIÇA, 2011) foi o segundo trabalho da literatura a tratar de replicação de bancos de dados relacionais em ambientes sujeitos a faltas bizantinas. Ele consiste em um *middleware* para replicação de bancos de dados tolerante a faltas bizantinas, que se baseia e adota o critério de consistência *Snapshot Isolation* (BERENSON et al., 1995), para o processamento de transações. Neste caso, é uma solução que não satisfaz completamente ao critério de consistência baseado em serialização (vide Seção 2.1.5). Uma primeira versão do sistema (PREGUIÇA et al., 2008) foi desenvolvida no intuito de avaliar a viabilidade da implementação de um esquema de replicação de bancos de dados BFT, a partir do PBFT (CASTRO; LISKOV, 1999) como protocolo de nível subjacente. Desde a sua primeira versão, o propósito do Byzantium era explorar os aspectos de concorrência providos pelo *Snapshot Isolation* para a replicação de bancos de dados em face da ocorrência de faltas bizantinas. E por esta razão, o protocolo de replicação deste sistema adota duas premissas básicas para as réplicas de bancos de dados, elas devem prover suporte nativo tanto ao *Snapshot Isolation* como aos *savepoints* – pontos de sincronização em uma transação.

Em sua versão inicial, o Byzantium fora apenas especificado, mas não implementado. Um trabalho mais recente apresentou diversas

melhorias na especificação do Byzantium (GARCIA; RODRIGUES; PREGUIÇA, 2011) e também sua implementação. A arquitetura especificada para o *middleware* é apresentada na Figura 26, adaptada de (GARCIA; RODRIGUES; PREGUIÇA, 2011). O protocolo é composto por um conjunto de $n = 3f + 1$ réplicas do banco de dados. No lado cliente, à frente da aplicação há um componente responsável pelo tratamento das requisições e respostas, que provê transparência de replicação e é acessado pela aplicação por meio de uma interface JDBC. Ainda no lado cliente, há um módulo de comunicação FIFO com suporte à comunicação em grupo, cuja implementação se baseia no *multicast* convencional, que é incrementado para prover a difusão confiável com a semântica FIFO, a fim de garantir a entrega das mensagens quando o emissor não falha. O módulo de comunicação também permite o envio de mensagens ponto-a-ponto com garantias de confiabilidade e ordenação.

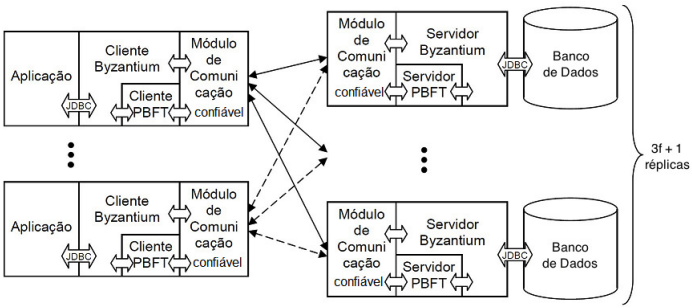


Figura 26 – Arquitetura de *middleware* do Byzantium.

No lado servidor, além do módulo de comunicação confiável há um componente que recebe as requisições, de acordo com o tipo de operação recebida (leitura/escrita), e encaminha para a execução apropriada no servidor de banco de dados. A execução das transações é realizada com o apoio da biblioteca que implementa o PBFT. De forma resumida, o processamento de transações é realizado a partir da seguinte dinâmica: as operações de início e de término de cada transação (i.e., *begin_trx*, *commit* ou *abort*) são enviadas pelo cliente às réplicas através do PBFT para linearizar o estado das réplicas e obter um mesmo *snapshot* para a transação. Ao receber um pedido de início de transação, o cliente seleciona aleatoriamente uma réplica para executar de maneira otimista as operações da transação. A partir daí, o cliente submete a sequência de operações de leitura e escrita que compõem a

transação diretamente a esta réplica primária, a qual por sua vez, imediatamente enviará os resultados para cada um dos comandos. Note que o algoritmo é baseado no mesmo princípio da DBSM (PEDONE; GUERRAOUI; SCHIPER, 2003) e do *deferred write/update* (BERNSTEIN; HADZILACOS; GOODMAN, 1987).

Ao concluir a transação, o cliente envia a operação de validação (*commit*) ou anulação (*rollback*) também por meio do PBFT, para assegurar que todas as réplicas finalizarão a transação na mesma ordem. Para tanto, o cliente envia junto ao pedido de validação as operações e os respectivos resultados recebidos, a todas as réplicas, que ao entregar a mensagem, executam as operações e verificam a consistência dos resultados executados inicialmente pela réplica primária, junto aos resultados obtidos na execução local. Se for verificada a consistência dos resultados, as réplicas decidem pela validação da transação, do contrário, ela é anulada. O sistema emprega algumas otimizações no processamento de transações, uma delas é quanto à propagação das operações da transação, onde o cliente pode optar por enviar as operações imediatamente a todas as réplicas, por meio de uma primitiva de comunicação em grupo. Deste modo, no momento da validação todas as réplicas já terão as operações que foram executadas pela transação. Não obstante, as operações de leitura que outrora eram submetidas a todas as réplicas, agora passam a ser submetidas à apenas $f + 1$ réplicas consideradas corretas no sistema, uma vez que estas $f + 1$ respostas iguais provenientes de diferentes réplicas são o suficiente para verificar a correção/consistência do resultado obtido.

Como o critério de consistência é baseado no *Snapshot Isolation*, nenhuma operação de leitura é bloqueada por operações de escritas concorrentes. Esta característica garante que validação de uma transação de somente-leitura termine imediatamente sem troca mensagens, sendo apenas necessário verificar se os resultados das operações foram enviados corretamente por $f+1$ réplicas. Uma observação importante é que, apesar dos autores discutirem a possibilidade de empregar o *middleware* em um ambiente com réplicas heterogêneas, eles afirmam que as versões existentes do Byzantium são totalmente funcionais apenas no PostgreSQL3. Por fim, cabe ressaltar que a consistência provida pelo *Snapshot Isolation* é estritamente mais fraca que aquela provida pela serialização, e é suscetível a algumas anomalias de concorrência (vide Seção 2.1.5). Além disso, no *Snapshot Isolation* as transações apenas enxergam os efeitos de transações passadas e as escritas são gerenciadas pela regra *first-updater-wins* (BERENSON et al., 1995), o que significa que, se duas transações concorrentes T_i e T_j escrevem sobre

um mesmo item de dados, apenas a primeira que solicitar a validação terá êxito, enquanto que a seguinte será anulada.

3.2.1.6 Tolerância a Faltas Bizantinas no Processamento de Transações em Sistemas Baseados em Armazenamento Chave-Valor

O *Byzantine Fault-Tolerant Deferred Update Replication*, ou apenas BFT-DUR (PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO, 2011; PEDONE; SCHIPER, 2012) consiste em um protocolo especificado para tolerar faltas bizantinas no processamento de transações sobre dados armazenados em ambiente replicado do tipo chave-valor (POKORNY, 2011), sendo portanto, diferente dos trabalhos já apresentados e que foram desenvolvidos para bancos de dados relacionais. Na realidade, a solução consiste em um protocolo baseado nos mesmos princípios da DBSM (PEDONE; GUERRAOUI; SCHIPER, 2003), mas para ambientes chave-valor tolerantes a faltas bizantinas, com algumas peculiaridades em relação à especificação original da DBSM, as quais são: (*i*) a execução de uma transação não requer nenhum tipo de comunicação entre os servidores, mas apenas a terminação desta; (*ii*) apenas um servidor executa os comandos da transação, mas todos os servidores corretos aplicam as atualizações de uma transação validada através do protocolo. Um dos resultados mais importantes apresentados por este trabalho é a definição de um mecanismo para executar transações somente-leitura em um único servidor, mesmo em circunstâncias onde é observado o comportamento bizantino por parte de alguma(s) réplica(s).

O protocolo faz o uso de um algoritmo de difusão com ordem total tolerante a faltas bizantinas, tal como o PBFT (CASTRO; LISKOV, 1999). Como uma réplica bizantina pode comprometer o estado da base de dados, o protocolo define dois tipos de resultados incorretos para operações executadas por réplicas faltosas, o dado-inválido e o dado-antigo. Um dado-inválido consiste em um dado fabricado por uma réplica faltosa, enquanto um dado-antigo pode ser válido, mas já defasado/obsoleto, já que o protocolo se apoia sobre um controle de concorrência multi-versão (BERNSTEIN; GOODMAN, 1981). O problema do dado-antigo é resolvido no protocolo por meio da inclusão de uma fase adicional no teste de certificação, que precede a validação da transação. Nesta fase é checado se os valores lidos pela transação não estão obsoletos. Desta forma, o teste de certificação é decomposto em duas partes, uma que realiza a verificação da validade dos dados, e a outra que checa a obsolescência dos mesmos. Além disso, para assegurar que

a transação foi, de fato, validada pelos servidores corretos, o cliente deve aguardar $f + 1$ respostas para o pedido de validação, vindos de diferentes réplicas.

No BFT-DUR, as operações de leitura de uma transação são executadas por apenas uma réplica, sem necessidade de comunicação com as demais réplicas, tampouco de o cliente enviar as operações por meio do algoritmo de difusão com ordem total. De outro modo, em uma transação de escrita as operações são enviadas apenas a uma réplica do sistema, onde as demais réplicas tomam conhecimento da transação somente no momento de sua validação – onde ocorre a propagação dos conjuntos de leitura e de escrita (*Read-Set* e *Write-Set*) para as réplicas. Algumas otimizações são apresentadas no intuito de aumentar os ganhos de escala e reduzir o impacto do uso de mecanismos criptográficos no protocolo. A primeira otimização é a criação de um *cache* no cliente para armazenar os itens de dados, valores, versões, resumos e elementos que atestem que os valores pertencem a uma visão consistente da base de dados, visando reduzir o número de acessos aos servidores. Assim, em algumas condições, os clientes podem executar operações de leitura sem a necessidade de contatar qualquer servidor, desde que os itens solicitados estejam em seu *cache*. Da mesma forma, as operações de escrita não são realizadas imediatamente nos servidores, mas em um *buffer* local do cliente. Neste caso, até mesmo a réplica primária toma conhecimento das operações de escrita somente quando o cliente solicita a validação para a transação.

Outra otimização que afeta em potencial o desempenho é a substituição das assinaturas de chave-pública nas mensagens de confirmação, e nos conjuntos de elementos que são usados para atestar a consistência e validade dos valores lidos por uma transação (visão consistente da base de dados), por vetores de códigos de autenticação de mensagens (MACs). Um aspecto interessante do protocolo é que o fato dele deixar a cargo do cliente a escolha da réplica primária para sua transação pode incorrer numa perda de desempenho, se todos os clientes optarem por uma mesma réplica primária. Por conseguinte, se uma única réplica é sempre escolhida como primária, em caso de uma condição elevada de carga ela pode vir a sofrer uma inundação de requisições, o que pode induzi-la a situação de negação de serviço.

3.2.1.7 Avaliação Analítica dos Protocolos BFT para Replicação de Bancos de Dados

Para finalizar a discussão acerca dos trabalhos correlacionados a esta tese no que diz respeito à replicação de bancos de dados, esta seção apresenta uma avaliação analítica preliminar para demonstrar a eficiência dos protocolos apresentados. Esta análise se dá através da verificação de algumas propriedades dos algoritmos apresentados, tais como o número de réplicas necessárias; a complexidade de mensagens verificada; a latência de comunicação – que representa o custo associado ao processamento de uma transação pelo respectivo protocolo; os tipos de consistência, de bancos de dado e de faltas suportadas. Os resultados verificados são apresentados na Tabela 4, e especificamente no caso da latência, as equações se referem ao número de mensagens requerido para todas as fases da transação, e o termo δ representa o número de operações executadas na transação. É digno de nota que a avaliação em questão considera apenas condições favoráveis de execução, isto é, execuções livres de falhas dos protocolos HRDB (VANDIVER et al., 2007), Byzantium (GARCIA; RODRIGUES; PREGUIÇA, 2011), BFT-DUR (PEDONE; SCHIPER, 2012) e DBMS (PEDONE; GUERRAOUI; SCHIPER, 2003).

Tabela 4 – Análise dos protocolos de replicação de bancos de dados.

Aspecto Analisado	**Protocolo de Replicação**			
	HRDB	**Byzantium**	**BFT-DUR**	**DBSM**
# Réplicas	$(2f+1)+1$	$3f+1$	$3f+1$	$2f+1$
Complexidade	$O(n)$	$O(n^2)$	$O(n^2)$	$O(n)$
Latência	$6\delta+3$	$2\delta+2(TOMCast)+2$	$2\delta+(TOMCast)+1$	$2\delta+(TOMCast)+1$
Consistência	Serialização	*Snapshot*	Serialização	Serialização
Banco de Dados	Relacional	Relacional	Chave-Valor	Relacional
Controle	Centralizado	Distribuído	Distribuído	Distribuído
Faltas	Bizantina	Bizantina	Bizantina	Parada

Como se pode notar, o DBMS é o que exibe os melhores resultados, o que decorre do fato do mesmo tolerar apenas faltas de parada. Dentre os protocolos que toleram faltas bizantinas o HRDB é o que requer o menor número réplicas, e tem a menor complexidade e latência. Isto se deve ao fato da abordagem de controle centralizado, diferente dos demais que o fazem de maneira distribuída. O fato do HRDB utilizar o controlador não distribuído evita o uso da **difusão com ordem total** na terminação da transação, o que implica na redução da complexidade

de mensagens. Por outro lado, a falha do controlador causa a blocagem de todo o sistema no HRDB, o que não ocorre nos demais. Em relação aos demais protocolos, o BFT-DUR é mais eficiente em termos de latência em relação ao Byzantium porque ele é especificado a partir de bancos de dados chave-valor, o que permite a otimização da difusão das escritas da transação apenas na terminação. Como o Byzantium é especificado para bancos de dados relacionais com o *Snapshot Isolation*, são requeridas duas difusões com ordem total, uma para iniciar a transação, para que seja possível obter um mesmo *snapshot* em todas as réplicas, e outra para propagar as operações da transação quando de sua terminação.

3.2.2 Validação Atômica e Distribuída de Transações

Esta seção apresenta alguns dos trabalhos encontrados na literatura que tratam da terminação de transações através de protocolos de validação atômica e de validação distribuída. Conforme já apresentado na Seção 2.2.3.2, a validação atômica não-bloqueante (ou *Non-Blocking Atomic Commitment* – NBAC) (HADZILACOS, 1990; BABAOGLU; TOUEG, 1993; GUERRAOUI, 1995) é um problema de acordo que visa prover garantias de atomicidade e de progressão, no âmbito da terminação de uma transação num sistema de computação distribuída. Neste caso, um protocolo de validação atômica é dito ser não-bloqueante e tolerante a faltas se ele satisfaz a todas as propriedades especificadas para o NBAC (BABAOGLU; TOUEG, 1993) (vide Seção 2.2.3.2). Tal como ocorre em qualquer classe de problemas de acordo em sistemas de computação distribuída (vide Seção 2.2.3), a solubilidade do NBAC depende da natureza das faltas admissíveis e também da habilidade de detectá-las – esta última de capital relevância – , pois é amparada pelo sistema subjacente considerado no modelo de computação para o qual uma solução é especificada. Em outros termos, em um ambiente com sujeição apenas a faltas de parada, a habilidade de se ter um conhecimento preciso acerca da ocorrência de falhas depende apenas de aspectos relacionados ao sincronismo presente no sistema.

3.2.2.1 Protocolos de Validação Atômica Não-Bloqueante Tolerantes a Faltas por Parada

Antes de adentrar na discussão em torno de alguns protocolos de validação atômica não-bloqueante (NBAC), é importante ressaltar que já fora demonstrado na literatura que este problema não pode ser resolvido em ambientes com sujeição à faltas bizantinas (CHARRON-BOST; SCHIPER, 2004). Tal demonstração decorre da propriedade de **acordo uniforme** não poder ser atendida em alguns problemas de acordo na presença de faltas bizantinas (CHARRON-BOST, 2001; RAYNAL; SINGHAL, 2001) – dentre os quais está o NBAC. Porém, antes mesmo de tal insolubilidade ter sido demonstrada, Rothermel e Pappe (ROTHERMEL; PAPPE, 1993) já haviam abordado sobre as dificuldades encontradas para assegurar a validação atômica de maneira distribuída, em sistemas abertos onde alguns dos participantes estivessem sujeitos à faltas que pudessem comprometer seus comportamentos. Não obstante, em se tratando do NBAC em ambientes sujeitos a faltas de parada, as soluções encontradas na literatura podem ser classificadas de acordo com os modelos de interação considerados, isto é, síncrono ou assíncrono. São exemplos de protocolos NBAC síncronos o 3PC (*Three Phase Commit*) (SKEEN, 1981) e o ACP-UTRB (*Atomic Commit Protocol – Uniform Timed Reliable Broadcast*) (BABAOGLU; TOUEG, 1993). E alguns dos protocolos assíncronos de NBAC encontrados na literatura são, o DNB-AC (*Decentralized Non-Blocking Atomic Commitment*) (GUERRAOUI; SCHIPER, 1995a); o MD3PC (*Modular Decentralized Three Phase Commit*) (GUERRAOUI; LARREA; SCHIPER, 1996) e o NB-2PC (*Non-Blocking Two Phase Commit*) (GREVE; NARZUL, 2002). Ademais, na literatura são apresentadas diversas soluções envolvendo o problema do NBAC nos mais diversos cenários e ambientes. Entretanto, serão aqui apresentados apenas os ora mencionados, de modo que uma boa compilação das muitas faces definidas para o problema NBAC pode ser encontrada em (SHAHZA et al., 2008).

Protocolo de Validação em Três Fases

O protocolo de validação em três fases (do inglês, *Three-Phase Commit* – 3PC) constitui a primeira proposta em termos de protocolos de validação, para eliminar situações de blocagem (SKEEN, 1981). O 3PC é um dos protocolos NBAC mais conhecidos da literatura, podendo ser visto como uma extensão do clássico 2PC (*Two-Phase Commit*). A

essência por trás do 3PC é evitar a situação de blocagem que ocorre no 2PC, em que alguns participantes podem validar a transação após terem recebido uma mensagem de decisão, enquanto que outros ainda podem estar incertos quanto ao resultado da transação. Para tanto, no 3PC a solução para tal problema ocorre pela inserção de uma fase adicional, denominada pré-validação (ou *pre-commit*), entre as duas fases especificadas pelo 2PC. Neste caso, durante esta fase de pré-validação é possível de se chegar a uma decisão preliminar antes de ser tomada a decisão final. A Figura 27 ilustra o padrão de comunicação empregado no 3PC, no qual participam seis (6) sítios (ou nodos) e o sítio 1 (um) atua como coordenador do protocolo.

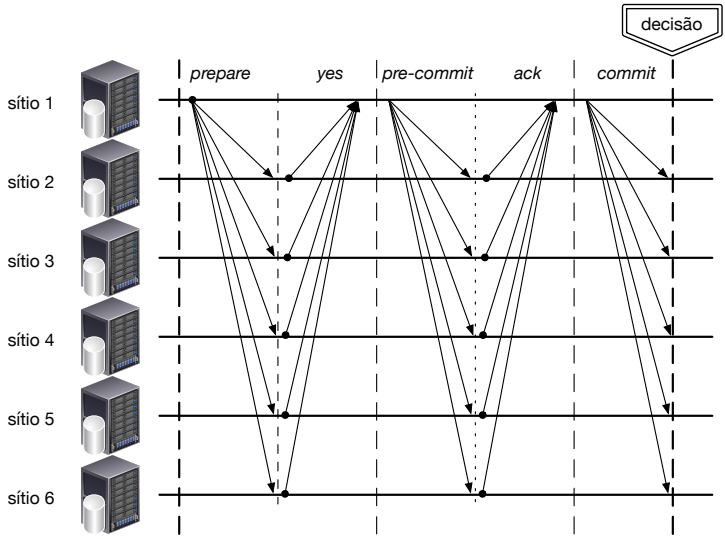


Figura 27 – Execução do protocolo 3PC.

Quando o coordenador verifica que uma transação está pronta para ser validada, ele envia uma mensagem *prepare* aos demais participantes, no intuito de verificar se eles também estão aptos a validar aquela transação. Ao receber a mensagem *prepare*, um participante responde ao coordenador segundo sua condição local, de modo que se a resposta for *yes* (i.e., `sim`) é porque aquele participante está apto a validar a transação; ou do contrário, envia um *no* (i.e., `não`) sinalizando que não pode validar a transação. Se o coordenador receber os votos `sim` de todos os participantes, ele envia uma mensagem *pre-commit* a

todos os participantes, que ao receberem tal mensagem passam a ter conhecimento de que todos os votos foram `sim` e então notificam o coordenador através de uma mensagem de reconhecimento *ack* – a partir daí os participantes saem da "zona de incerteza". Uma vez que o coordenador tenha recebido as notificações de todos os participantes quanto à mensagem *pre-commit*, ele decide pela validação da transação e envia sua decisão a todos os participantes. E por fim, um participante, ao receber a decisão pela validação advinda do coordenador, também decide pela validação – neste ponto, ele sabe que todos os demais participantes não se encontram no período de incerteza –, e com isso o protocolo é encerrado. Note que este se trata do caso mais favorável, isto é, aquele no qual se presume que não há falhas e que nenhum participante vota `não`.

Por outro lado, para lidar com situações de falha, algumas ações baseadas em um temporizador são especificadas, para que um participante possa continuar mesmo quando uma mensagem esperada em alguma fase do protocolo não for recebida. Outrossim, um participante que vier a falhar e se recuperar *a posteriori*, deve ser capaz de tomar uma decisão consistente com aquela tomada pelos participantes operacionais. No caso, pelo menos cinco situações devem ser consideradas: (*i*) um participante está a aguardar pela mensagem *prepare*; (*ii*) o coordenador está a aguardar por votos; (*iii*) um participante está a aguardar pela mensagem *pre-commit*; (*iv*) o coordenador está a aguardar por notificações pós-envio da mensagem *pre-commit*; e (*v*) um participante está a aguardar pela decisão. Para o caso (*i*), um participante pode decidir de maneira unilateral pela anulação, já que ele não declarou seu voto `sim` até aquele momento. Quanto à situação (*ii*), o coordenador pode decidir seguramente pela anulação, justamente porque até aquele ponto ele não chegou a nenhuma decisão, e neste caso, nenhum participante decidiu pela validação. Em se tratando da situação (*iv*), como o participante falhou antes de enviar a notificação, significa que ele já votou `sim`, e portanto, o coordenador pode ignorar esta falhas e decidir pela validação, como se nenhuma falha tivesse ocorrido.

De outro modo, para as situações (*iii*) e (*v*) os participantes não podem decidir por si próprios. Para tanto, em tais situações eles dão início a um protocolo de terminação, por meio do qual, eles se comunicam uns com os outros a fim de descobrir o que fora decido. Ainda, o protocolo deve garantir a ausência de blocagem (i.e., *liveness*) e permitir que todos os participantes corretos obtenham uma decisão consistente, sem ter de aguardar pela reparação dos participantes que porventura vieram a falhar. Na prática, isto é resolvido por meio da substituição do coor-

denador, que é eleito dentre o conjunto dos participantes não faltosos. Uma vez que um novo coordenador tenha sido escolhido/eleito, ele tem a tarefa de conduzir todos os participantes corretos para a validação ou anulação da transação, de acordo com suas condições locais. Posto que falhas podem ocorrer também durante a execução do protocolo de terminação, o novo coordenador verifica quanto à transição de seu estado local atual (i.e., *commit* ou *abort*), e aguarda pela confirmação dos demais participantes para então enviar sua decisão final. Note que o 3PC resolve o problema NBAC ao custo de 5 (cinco) passos de comunicação e da difusão de $5n$ mensagens. À vista disso, Dale Skeen discute sobre uma versão descentralizada do 3PC na tentativa de reduzir o número de passos de comunicação para três, porém, ao custo de uma complexidade de mensagens superior (SKEEN, 1981).

Protocolo ACP–UTRB

No intuito de reduzir o número de passos de comunicação do 3PC, tal como a versão descentralizada do mesmo (SKEEN, 1981), Babaoglu e Toueg propuseram o protocolo denominado ACP–UTRB (*Atomic Commit Protocol – Uniform Timed Reliable Broadcast*) (BABAOGLU; TOUEG, 1993). O ACP-UTRB é especificado sob a mesma estrutura do 2PC, porém, é capaz de assegurar a ausência de blocagem por meio do uso de uma primitiva de comunicação baseada em difusão, para disseminar as mensagens de decisão aos participantes da transação. Neste sentido, possíveis cenários que induzem à blocagem do protocolo são evitados por meio do uso da primitiva de difusão denominada *Uniform Timed Reliable Broadcast*, ou simplesmente UTRB. Para tanto, esta primitiva satisfaz as seguintes propriedades, em que $\Delta = \delta$ (BABAOGLU; TOUEG, 1993):

- **Validade** (*validity*): se um processo correto dissemina uma mensagem m a um grupo de processos $\mathcal{G}$, então todos os processos corretos em $\mathcal{G}$ finarão por entregar m;

- **Integridade-Uniforme** (*uniform-integrity*): para qualquer mensagem m, cada processo correto em $\mathcal{G}$ entrega m no máximo uma vez, e somente se m foi previamente disseminada por algum processo;

- **Atualidade-Δ-Uniforme** (*uniform-Δ-timeliness*): existe uma constante Δ conhecida, tal que, se a disseminação de uma men-

sagem m é iniciada no tempo real t, então nenhum processo em $\mathcal{G}$ recebe m após o tempo real $t + \Delta$;

- **Acordo-Uniforme** (*uniform-agreement*): se qualquer participante, correto ou faltoso, entrega uma mensagem m, então todos os participantes corretos em $\mathcal{G}$ finarão por entregar m.

O UTRB é definido em termos das primitivas $R\text{–}broadcast(m, \mathcal{G})$ e $R - deliver(m)$. Neste protocolo subjacente de disseminação de mensagens, cada processo em $\mathcal{G}$ retransmite todas as mensagens que recebe pela primeira vez aos demais processos, e em então entrega a mensagem. Com isso, é garantido que todos os processos corretos finarão por entregar m. Note que isso também permite satisfazer a propriedade de **acordo-uniforme**, mesmo que o processo que disseminou inicialmente a mensagem – ou mesmo outro que a replicou –, posteriormente vier a sofrer uma parada abrupta. Além disso, os autores demonstram que existe um atraso constante $\Delta = (F + 1)\delta$, pelo qual ocorre a entrega da mensagem m, sendo que F denota o limite de faltas durante a execução do protocolo de validação atômica. O diagrama que exemplifica uma execução favorável do protocolo ACP-UTRB é apresentado na Figura 28.

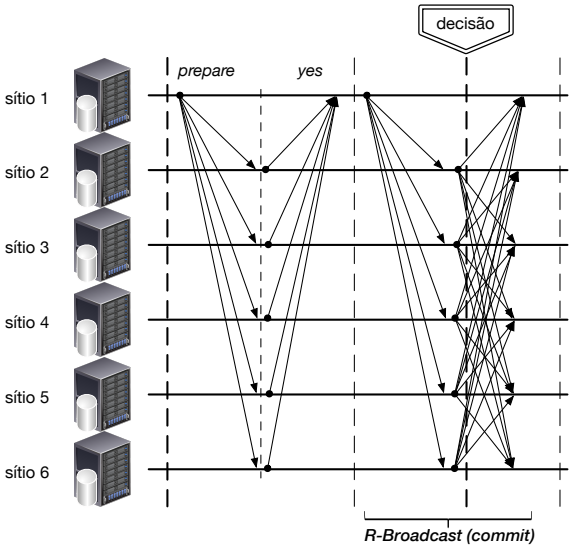


Figura 28 – O protocolo ACP–UTRB.

Note que o protocolo ACP-UTRB é uma derivação do 2PC, em que apenas a disseminação das mensagens de decisão são realizadas por meio da primitiva do UTRB, a fim de evitar as situações que culminam na blocagem daquele protocolo (SKEEN, 1981). Neste caso, um participante do ACP-UTRB, ao enviar seu voto após ter recebido a mensagem *prepare* (cfm. primeiro e segundo quadrantes da Figura 28), define um temporizador local para $\delta + \Delta$, onde δ representa o limite superior do intervalo de tempo necessário para que o seu voto chegue ao coordenador (i.e., o sítio 1), e Δ representa o limite superior de intervalo de tempo necessário para que a mensagem de decisão chegue em cada participante correto. Assim, se o coordenador vier a falhar por parada, os temporizadores dos participantes – que foram iniciados quando da espera pela mensagem de decisão – serão expirados, e por isso eles podem decidir de maneira segura pela anulação da transação, já que nenhum outro participante recebeu uma mensagem de decisão pela validação – cfm. as propriedades de **acordo-uniforme** e **atualidade-Δ-uniforme** do UTRB. Esta importante característica do ACP-UTRB elimina a possibilidade de uma espera indefinida, tal como ocorre no 2PC.

Protocolo NBAC Descentralizado

É importante salientar que os protocolos 3PC e ACP–UTRB se baseiam no modelo de interação síncrono. Em se tratando de soluções para o modelo assíncrono, conforme já descrito na Seção 2.2.3.2, o NBAC não pode ser resolvido neste modelo, mas apenas o NB-WAC. E baseado neste importante resultado, diversos protocolos foram desenvolvidos, como é o caso do DNB–AC (*Decentralized Non-Blocking Atomic Commitment*) (GUERRAOUI; SCHIPER, 1995a). Na ausência de suspeitas de falhas, o DNB–AC possui a mesma estrutura proposta para a versão descentralizada do 3PC, elaborada por Skeen para ambientes síncronos (SKEEN, 1981). Por outro lado, diferente do 3PC descentralizado, o protocolo de terminação do DNB–AC está encapsulado num protocolo de consenso uniforme (CHARRON-BOST; SCHIPER, 2004), o qual utiliza-se de um detector de falhas da classe $\diamond$S (CHANDRA; TOUEG, 1996), a fim de prover as condições requeridas para a progressão do consenso em ambientes assíncronos.

Como se pode verificar pela Figura 29, o protocolo DNB–AC é especificado a partir de três passos de comunicação. Da mesma maneira como nos demais protocolos, no primeiro passo é onde o coordenador

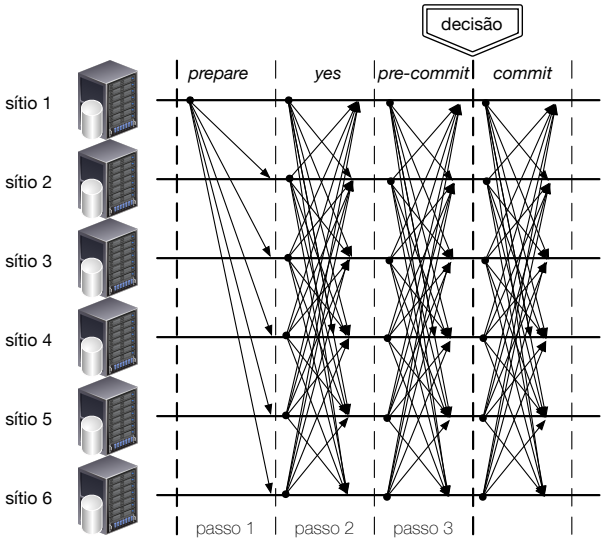


Figura 29 – Diagrama de execução do protocolo DNB–AC.

inicia o processo de validação através do envio da mensagem *prepare* a todos os participantes da transação. No segundo passo, um participante que vota `sim` envia seu voto a todos os outros participantes, que ao receberem os votos `sim` de todos, enviam uma mensagem *pre-commit* também a todos os participantes, no passo três. E finalmente, uma vez que um participante tenha recebido mensagens *pre-commit* para uma mesma transação de todos os participantes, ele decide pela validação (i.e., *commit*). Note que um participante que decide, encaminha sua decisão para todos os outros participantes – o que é requerido para assegurar a propriedade de acordo definida para o problema da validação atômica, de modo que, se um participante correto toma uma decisão, então todos os participantes corretos também tomarão uma decisão.

Note que o parágrafo anterior descreve o protocolo num cenário favorável, isto é, onde todos votam `sim` e não há suspeitas de falhas. Deste modo, se durante o primeiro passo do protocolo um participante suspeita que houve falha do coordenador ou vota `não`, ele toma uma decisão unilateral pela anulação da transação e dissemina sua decisão a todos os outros participantes. Entretanto, se alguma suspeita de falha ocorre durante o passo dois, um participante não pode decidir de maneira unilateral pela anulação. Neste caso, o processo envia uma men-

sagem *start-consensus* a todos os participantes, para que eles iniciem uma instância do protocolo subjacente de consenso uniforme, tendo *abort* como seu valor inicial – isto porque neste ponto o participante não tem conhecimento dos votos de todos os participantes. Assim, o resultado obtido pelo protocolo de consenso uniforme define o resultado da transação para aquele participante. Da mesma forma, se durante o passo três um participante suspeita de qualquer outro participante ou recebe a mensagem *start-consensus*, ele inicia a instância do protocolo subjacente de consenso uniforme com o valor inicial *commit* – note que neste ponto o participante já tem o conhecimento dos votos de todos. E como ocorre no passo dois, o resultado do consenso uniforme torna-se o resultado da transação para o participante.

O comportamento do DNB–AC na ausência de falhas é exatamente igual ao do protocolo 3PC descentralizado. Por outro lado, quando há alguma suspeita de falha o DNB–AC adota um protocolo de consenso uniforme como protocolo de terminação, assumindo com um detector de falhas da classe $\diamond$S. Neste caso, o DNB–AC tolera até f faltas, num cenário onde, pelo menos $\lceil \frac{(n+1)}{2} \rceil$ participantes são corretos.

Protocolo 3PC Modular e Descentralizado

No intuito de reduzir a complexidade de mensagens inerente ao protocolo DNB–AC, os mesmos autores propuseram o MD3PC (*Modular Decentralized Three Phase Commit*) (GUERRAOUI; LARREA; SCHIPER, 1996). O aspecto chave do MD3PC se baseia na ideia de ter um sub-protocolo para evitar situações de blocagem, o qual é executado apenas por um subconjunto dos participantes da transação – denotado por Set_{NB} –, cuja cardinalidade depende do número de faltas a ser tolerada durante a execução do protocolo. Neste sentido, uma vez que o protocolo possa contar com um detector de falhas da classe $\diamond$S, um limite de faltas $f < \frac{n}{2}$ deve ser observado, e portanto, $|Set_{NB}| = 2f + 1$. Ademais, diferente do DNB–AC que adota um protocolo subjacente de consenso uniforme para terminação, no caso de suspeitas de falhas, o MD3PC é baseado em um consenso de maioria (i.e., *majority consensus*). Esta variante do consenso é definida pelas mesmas propriedades do consenso uniforme (vide Seção 2.2.3.1), exceto pela validade, que é definida da seguinte maneira (GUERRAOUI; LARREA; SCHIPER, 1996):

- **Validade-Uniforme-Majoritária** (*majority-uniform-validity*): o valor de decisão deve ser o valor inicial de algum processo, e se uma maioria de valores iniciais é 1, então o valor de decisão deve

ser 1;

Note que no modelo transacional, o valor 1 corresponde a validação (*commit*), enquanto que 0 corresponde a anulação (*abort*). Neste contexto, a especificação da propriedade Validade-Uniforme-Majoritária permite que um participante do MD3PC decida pela validação tão logo tenha recebido mensagens *pre-commit* de uma maioria de participantes do conjunto Set_{NB}. E tal como ocorre com o consenso uniforme, o consenso de maioria pode ser resolvido com detectores de falhas da classe $\diamond$S (GUERRAOUI; LARREA; SCHIPER, 1996). A Figura 30 ilustra o protocolo MD3PC num cenário favorável, em que todos os participantes votam `sim` e não há suspeitas de falhas.

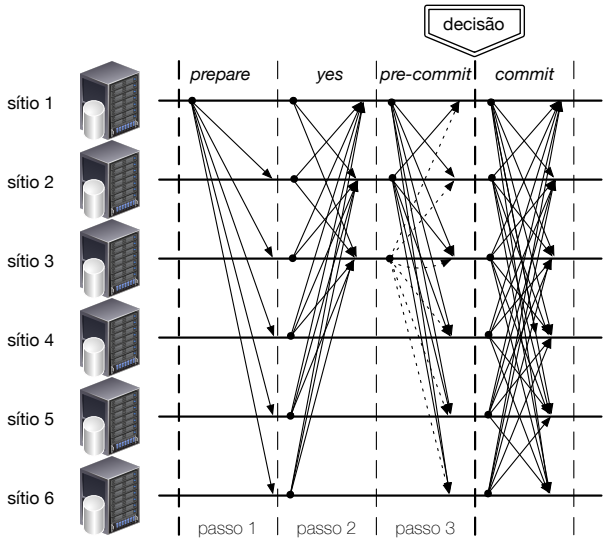


Figura 30 – Padrão de comunicação empregado no protocolo MD3PC.

O cenário da Figura 30 ilustra o caso onde $f = 1$ e $Set_{NB} = \{sitio_1, sitio_2, sitio_3\}$, tal que $|Set_{NB}| = 3$. Tal como ocorre nos demais protocolos até então apresentados, durante o primeiro passo do MD3PC o coordenador envia uma mensagem *prepare* a todos os participantes da transação. No entanto, no passo dois os votos dos participantes são enviados apenas aos membros do conjunto Set_{NB}. No passo três, quando um membro de Set_{NB} recebe votos `sim` de **todos** os participantes, ele envia uma mensagem *pre-commit* a todos. E uma vez que um partici-

pante tenha recebido mensagens *pre-commit* de uma maioria de Set_{NB} – note pela ilustração, que apenas os sítios 1 e 2 enviam a mensagem *pre-commit* –, ele envia sua decisão a todos os outros participantes e decide pela validação.

A considerar um cenário não favorável, se um participante vota `não` ou suspeita do coordenador, durante o passo 1, ele decide de maneira unilateral pela anulação da transação. De outro modo, caso suspeitas de falhas sejam verificadas num dos passos restantes, o protocolo subjacente de consenso de maioria é iniciado apenas pelos membros de Set_{NB}. Porém, se a suspeita recaiu durante a execução do passo 2, o consenso é iniciado tendo *abort* como valor inicial, enquanto que no passo 3, o valor inicial fornecido para o consenso é *commit*.

Protocolo 2PC Não-Bloqueante

O protocolo NB–2PC (*Non-Blocking Two Phase Commit*) proposto por Greve e Narzul (GREVE; NARZUL, 2002), é um protocolo de validação atômica fraca não-bloqueante que se baseia no mesmo princípio adotado pelo MD3PC, isto é, aquele que permite decisões antecipadas quando o ambiente reune condições favoráveis para tal (p. ex.: ausência de falhas). O NB–2PC é especificado a partir uma estrutura modular, onde um protocolo de consenso capaz de decidir em um único passo de comunicação é parte de sua especificação (BRASILEIRO et al., 2001). Em suma, o protocolo de consenso adotado, realiza o feito de decidir em um único passo de comunicação por meio da seguinte estratégia. Antecipadamente à execução do protocolo, um subconjunto de processos $\mathcal{S}$, de cardinalidade $|\mathcal{S}| \geq f + 1$, é escolhido dentre o universo de processos que participam do sistema (i.e., $\mathcal{S} \subset \Pi$), em que a composição de $\mathcal{S}$ é de conhecimento de todos os processos do sistema. No caso, numa execução do protocolo de consenso, primeiramente os processos trocam as proposições entre si; e se um processo coleta todas as propostas oriundas dos processos de $\mathcal{S}$ e elas são idênticas, ele pode decidir de imediato por aquele valor proposto. De outro modo, os processos que não conseguirem decidir nestes moldes, iniciam a execução de um outro protocolo de consenso subjacente.

Com base na ideia do consenso em um único passo, a dinâmica de funcionamento do NB–2PC é a seguinte. Na primeira fase, os processos enviam seus votos acerca do processamento de sua parte da transação – passo 1, primeiro quadrante da Figura 31 –, e coletam os votos dos demais. E tendo por base os votos recebidos, eles iniciam a segunda fase

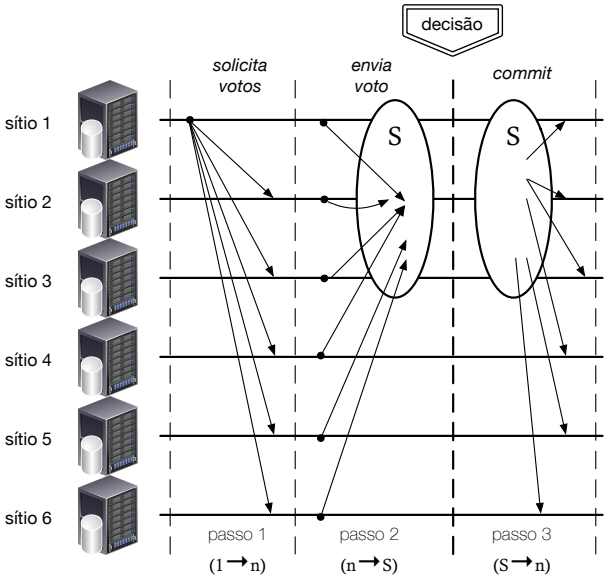


Figura 31 – Passos de comunicação e fase do protocolo NB–2PC.

já com suas respectivas proposições para o consenso – passo 3 da Figura 31 –, que é *commit* se todos os votos coletados são `sim`, ou *abort*, caso tenha havido alguma suspeita de falha durante a fase anterior. Uma vez que o módulo de consenso utilizado é aquele que permite decisões em único passo de comunicação, se os votos de todos os processos em $\mathcal{S}$ forem iguais, a decisão ocorre de imediato. Por outro lado, naqueles processos em que a decisão não pôde ocorrer de maneira antecipada, é iniciada uma terceira fase, onde um protocolo subjacente de consenso uniforme é instanciado como protocolo de terminação.

Nos moldes da Figura 31, note que os votos dos participantes são enviados apenas para os processos em $\mathcal{S}$, já que a decisão acerca do resultado da transação é delegada a eles. Neste caso, embora todos os processos do sistema tenham o direito de voto, apenas as proposições dos processos em $\mathcal{S}$ são computadas para fins de decisão pelo consenso. O mesmo ocorre pela decisão via o módulo consenso uniforme subjacente, que embora seja realizado por todos os processos do sistema, considera apenas as proposições provenientes dos processos em $\mathcal{S}$.

3.2.2.2 Protocolos de Validação Distribuída Tolerantes a Faltas Bizantinas

Por fim, em face da insolubilidade e dificuldade apontadas a respeito do NBAC com faltas bizantinas (CHARRON-BOST; SCHIPER, 2004), alguns dos trabalhos correlacionados encontrados na literatura não são baseados nas propriedades originalmente definidas pelo NBAC. Por conseguinte, estes trabalhos não podem ser caracterizados como protocolos de validação atômica de transações, mas sim protocolos de validação distribuída de transações. Tal afirmação se deve ao fato de que as propriedades de atomicidade e acordo uniforme não são satisfeitas por estes trabalhos. Alguns destes trabalhos envolvendo a validação distribuída de transações em ambientes com suporte a faltas bizantinas são apresentados na sequência. Todavia, embora existam diversas propostas de protocolos para validação distribuída de transações tolerantes a faltas bizantinas (MOHAN; STRONG; FINKELSTEIN, 1983; ROTHERMEL; PAPPE, 1993; ZHAO, 2007; MAHAJAN; SINGHAL, 2009), nesta seção nos limitamos a apresentar apenas aqueles que estão em maior consonância com o protocolo de validação atômica objeto de estudo e de proposição desta tese.

Validação Distribuída usando Acordo Bizantino

A menção ao problema causado pela ocorrência de faltas bizantinas no decurso da validação de transações distribuídas foi abordado pela primeira vez por Mohan, Strong e Finkelstein (MOHAN; STRONG; FINKELSTEIN, 1983). Neste trabalho, os autores propuseram uma alteração no protocolo padrão de validação em duas fases (do inglês, *Two Phase Commit* – 2PC) (SKEEN, 1981) por meio do uso de um algoritmo de acordo bizantino para orientar o processo de validação de transações em um banco de dados distribuído. Mais precisamente, a solução proposta foi a de integrar um protocolo de acordo bizantino ao protocolo padrão 2PC em substituição a segunda fase do 2PC, cujas finalidades eram aumentar a eficiência do protocolo; e possibilitar o tratamento de faltas de natureza arbitrária.

A ideia por trás desta proposta era executar o protocolo de acordo bizantino entre o coordenador e os demais participantes da transação, e também com algumas entidades (p. ex.: participantes) redundantes, todos residentes em um ambiente de *cluster*. O ambiente de *cluster* por sua vez, é organizado a partir de uma estrutura de ár-

vore de processos multi-nível, que é criada para cada transação a fim de realizar as atividades atinentes ao processamento da transação, e contém os processos dos sítios que fazem parte daquela transação. Para cada árvore, a raiz desta árvore é um dos processos do *cluster*, mais precisamente aquele que desempenha o papel de coordenador da transação, que também será o responsável pela interação com a aplicação que originou a transação.

O protocolo especificado neste trabalho é, em sua essência, um protocolo de validação distribuída, que assegura algumas propriedades tais como: a ausência de bloqueio, de modo que a falha do coordenador não afete a progressão dos demais participantes; tolera qualquer número de faltas de omissão até um número máximo necessário para particionar a rede, e então tolerar um número equivalente de faltas de comissão, as quais podem vir a ocorrer num período suficientemente tardio durante o processamento da validação da transação. É importante salientar que o protocolo não fornece confiabilidade em face da ocorrência de faltas de natureza arbitrária durante o decurso da transação como um todo, mas apenas durante a execução da validação desta. A abordagem empregada impede o coordenador do protocolo 2PC modificado de disseminar resultados conflitantes para as transações em processo de validação, para os diferentes participantes destas transações, sem ser detectado.

No contexto do trabalho, a modificação proposta para o 2PC foi especificada para duas variantes conhecidas do 2PC, as quais são o *Presumed Commit* (PC) e o *Presumed Abort* (PA) (MOHAN; LINDSAY, 1983). No caso, se o usuário decide por validar a transação, o processo raiz da árvore (i.e., o coordenador da transação) recebe uma mensagem `COMMIT TRANSACTION`, do contrário, se o usuário decide por anular a transação ele envia ao coordenador uma mensagem `ABORT TRANSACTION`. Por sua vez, a raiz da árvore propaga aos seus subordinados nos níveis inferiores da árvore a decisão recebida do usuário. E como os processos estão organizados na árvore, cada subordinado propaga a mesma mensagem aos seus subordinados em níveis inferiores, e assim sucessivamente. Mensagens de anulação (i.e., *abort*) enviadas nestas circunstâncias não necessitam ser reconhecidas, isto é, de confirmação. Por outro lado, se a raiz recebe um pedido de validação (i.e., *commit*), ele inicia o algoritmo de validação através do envio de uma mensagem de preparação aos seus subordinados. Naquele momento, a raiz pode optar pela variante do algoritmo 2PC que deseja executar para realizar a validação da transação, os quais são, o PC ou o PA.

Todavia, a abordagem proposta pelos autores possui algumas

desvantagens ou mesmo deficiências. A primeira é a exigência de que todos os elementos subordinados, de maneira recursiva, executem a instância do acordo bizantino a fim de obter uma decisão para cada transação. Isto pode implicar em um problema de sobrecarga quando o tamanho do *cluster* for relativamente grande. A segunda é que os autores assumem que o *cluster* pode possuir processos além daqueles cobertos pela área de abrangência da árvore, mas tolera faltas manifestadas por estes processos, sejam eles coordenadores ou subordinados. E a terceira advém da necessidade de todos os participantes do *cluster* raiz de conhecer os demais participantes que pertencem ao mesmo *cluster*, o que, por conseguinte impede que transações sejam propagadas de maneira dinâmica no ambiente. Além disso, tal aspecto soa um tanto estranho já que, de um modo geral, nos protocolos de validação apenas o coordenador é quem precisa conhecer os participantes de cada transação.

Validação Distribuída Tolerante a Faltas Bizantinas

O algoritmo denominado *Byzantine Fault Tolerant Distributed Commit*, ou simplesmente BFTDC (ZHAO, 2007) consiste em uma proposta de adaptação do PBFT para a aplicação no processamento e validação de transações executadas sobre arquiteturas orientadas a serviços, isto é, em serviços Web. A especificação do algoritmo do BFTDC parte de uma combinação do 2PC com o PBFT, na qual o PBFT é inserido entre as duas do 2PC. Em suma, o protocolo do PBFT que é inserido entre as fases 1 e 2 do 2PC clássico, agrega mais três fases ao protocolo de validação distribuída. Estas fases adaptadas pelo PBFT são denominadas *ba-pre-prepare*, *ba-prepare* e *ba-commit*.

E embora os protocolos de acordo e troca de visão do PBFT sejam aplicados no contexto do BFTDC, ambos diferem em seus objetos. Por exemplo, o acordo do PBFT é usado para que as réplicas cheguem a um acordo quanto à ordenação total das requisições dos clientes, já o acordo do BFTDC é usado para que as réplicas do coordenador da transação possam chegar a um acordo quanto ao resultado daquela transação. Se de um lado o PBFT utiliza apenas uma única instância do algoritmo de acordo para ordenar todas as requisições, o BFTDC cria e executa várias instâncias do algoritmo de acordo simultaneamente – uma para cada transação, e independentemente umas das outras.

No mesmo modo como ocorre no 2PC, o BFTDC envolve um

grupo de processos denominados participantes. Dentre os participantes da transação, um aspecto que merece destaque é que o protocolo considera que dentre eles há alguns que desempenham dois papéis necessários para a corretude do protocolo, o **coordenador** e o **iniciador**. Como já dito, o coordenador e iniciador são dois tipos especiais de participantes, dos quais o coordenador é responsável por conduzir o protocolo de validação distribuída, e também por executar alguns serviços essenciais para o funcionamento do protocolo, são eles: serviço de ativação, serviço de registro, serviço de coordenação e serviço de conclusão, serviço de conclusão/iniciador e serviço de participante. De outro modo, o iniciador é o responsável por iniciar e terminar a transação, e também por propagar os dados pertinentes às transações aos demais participantes comuns.

Tanto o coordenador como o iniciador são replicados em nós distintos, isto é, separados uns dos outros de tal forma que um mesmo sítio não abriga um coordenador e um iniciador. São necessárias pelo menos $3f + 1$ réplicas do coordenador, enquanto apenas $2f + 1$ réplicas do iniciador são necessárias, já que este último é uma entidade sem estado (i.e., *stateless*). Os participantes comuns não são replicados. A transação distribuída é iniciada quando o cliente solicita o início ao "iniciador". O iniciador, por sua vez, solicita ao serviço de ativação provido pelo coordenador, a inicialização da transação. O coordenador, por sua vez, aceita o pedido oriundo o iniciador se, pelo menos, $f + 1$ réplicas do iniciador enviaram a solicitação. A partir daí, a réplica primária do coordenador executa uma instância do PBFT e ao término da execução do acordo, as réplicas do iniciador aceitam um resultado advindo de, pelo menos, $f + 1$ réplicas do coordenador.

Ao concluir com êxito a execução de todas as operações que compõem a transação, o iniciador envia um pedido de validação para a transação em questão às réplicas do coordenador; do contrário, isto é, se a transação não foi executada em sua plenitude, o iniciador envia um pedido de anulação em vez de um pedido de validação. O coordenador só aceita o pedido de validação ou anulação ao receber $f + 1$ pedidos mutuamente consistentes de diferentes réplicas de iniciador. Assim, ao aceitar o pedido de validação ou anulação advindo do iniciador, as réplicas do coordenador iniciam a primeira fase do protocolo 2PC, sendo que ao final desta fase, um protocolo de acordo bizantino similar ao do PBFT é executado pelas réplicas, para que elas possam decidir/acordar acerca do resultado da transação. Se as réplicas do coordenador receberam um pedido de anulação, a segunda fase do protocolo 2PC é ignorada e a decisão final é enviada aos participantes. De

outro modo, se as réplicas do coordenador receberam um pedido de validação, ao concluir a rodada do acordo bizantino elas notificam as réplicas do iniciador sobre a decisão que culminou no resultado final da transação (p.ex., validação ou anulação). Uma réplica de iniciador aceita o resultado apenas se receber pelo menos $f + 1$ notificações correspondentes para uma mesma transação, advindas de diferentes réplicas do coordenador. De maneira similar, um participante comum aceita um pedido, uma validação ou uma anulação apenas se receber $f + 1$ mensagens mutuamente consistentes para a mesma transação, de diferentes réplicas do coordenador.

Algumas considerações em relação do BFTDC é que o mesmo não envolve o cliente da transação na execução do protocolo de validação, além de não atribuir números de sequência para ordenar transações. Outros aspectos que merecem destaque são: (i) o uso de várias instâncias do protocolo de acordo bizantino provido pelo PBFT – i.e., um para cada transação validada a partir do protocolo; (ii) o fato de haver um coordenador distinto para cada transação incorre em um custo adicional para o protocolo, já que qualquer participante pode exercer o papel de coordenador, e a considerar que apenas o coordenador e iniciador são replicados, há a necessidade de se efetuar o gerenciamento para a inicialização e o desligamento de réplicas para o coordenador de cada transação.

3.2.2.3 Eficiência dos Protocolos de Validação Atômica e Distribuída

Nos mesmos moldes do que foi realizado para os trabalhos correlacionados acerca da replicação de bancos de dados (apresentado na Seção 3.2.1.7), a apresentação dos protocolos de validação é finalizada através de uma comparação analítica entre os mesmos. Esta avaliação visa demonstrar a eficiência de alguns dos protocolos apresentados, a partir de algumas propriedades verificadas para os mesmos. Os resultados acerca da eficiência dos protocolos 3PC (SKEEN, 1981), ACP-UTRB (BABAOGLU; TOUEG, 1993), DNB-AC (GUERRAOUI; SCHIPER, 1995a), MD3PC (GUERRAOUI; LARREA; SCHIPER, 1996), NB-2PC (GREVE; NARZUL, 2002) e BFTDC (ZHAO, 2007) são enumerados na Tabela 5. É importante salientar que os resultados aludidos para esta avaliação levam em conta apenas condições favoráveis de execução dos protocolos.

Dentre os resultados exibidos na Tabela 5, alguns pontos merecem destaque. O clássico 3PC é o que pode requerer um menor número

de mensagens dependendo do número de participantes no protocolo. O ACP-UTRB requer um número menor de passos de comunicação, porém, ao custo de utilizar uma difusão confiável na fase de decisão – o que implica em sua complexidade ser quadrática. Os protocolos DNB-AC e MD3PC têm a mesma estrutura básica, sendo que a diferença entre eles é o número de mensagens requeridas para o acordo, mas ambos apresentam a mesma complexidade. Tanto o NB-2PC como o 3PC apresentam a menor complexidade. O MD3PC e o NB-2PC são desenvolvidos sob o mesmo princípio, e se baseiam na permissão de decisões antecipadas quando o ambiente no qual o protocolo está sendo executado reúne condições favoráveis para tal (p. ex.: ausência de falhas). No caso, a decisão é delegada a um subconjunto de processos denotados por Set_{NB} e S, em que ambos devem ser menores ou iguais ao conjunto formado por todos os processos do sistema, e cujos elementos são escolhidos previamente à execução do protocolo. A diferença entre eles está na forma como o Set_{NB} e o S são acionados e também pelas condições nas quais o protocolo termina.

Tabela 5 – Eficiência dos protocolos de validação de transações.

Protocolo	**# Passos**	**# Mensagens**	**Complexidade**	**Faltas**
3PC	5	$5n$	$O(n)$	parada
ACP-UTRB	3	$2n + n^2$	$O(n^2)$	parada
DNB-AC	3	$3n^2 + n$	$O(n^2)$	parada
MD3PC	3	$3nf + 3n + n^2$	$O(n^2)$	parada
NB-2PC	3	$2nf + 3n$	$O(n)$	parada
BFTDC	6	$4n + 2$ + PBFT*	$O(n^2)$	bizantina

Notadamente, os protocolos 3PC, ACP-UTRB, DNB-AC, MD3-PC e NB-2PC apresentam maior eficiência em relação ao BFTDC, o que decorre do fato deles tolerarem apenas faltas de parada, enquanto o último tolera faltas bizantinas. Por outro lado, as complexidades do protocolos menos eficientes que toleram faltas de parada são idênticas à do BFTDC, ao custo de um número maior de passos para resolver o problema, no caso do último. E, por fim, em relação ao trabalho de Mohan *et al.* (1983), embora o mesmo tenha sido a pedra fundamental em termos de protocolos de validação com faltas bizantinas, optamos por não demonstrar os resultados em razão de não ser possível expressar de uma maneira clara os valores adotados. Isso se deve principalmente ao fato de que naquele protocolo os processos são organizados em uma

estrutura hierárquica de árvore, onde cada elemento da árvore (i.e., um processo) pode ter um número arbitrário de subordinados, e assim recursivamente – o que dificulta em muito a expressão dos respectivos valores através de alguma equação.

3.3 CONSIDERAÇÕES DO CAPÍTULO

À luz do estudo apresentado no âmbito deste capítulo, esta tese segue numa perspectiva de que embora existam algumas soluções tanto para replicação de dados como para validação distribuída de transações, elas não atendem às todas as classes de aplicações. No caso de replicação de banco de dados, isso decorre do fato deles considerarem algumas hipóteses que são restritivas a determinadas classes de aplicações, por abrangerem a resolução de pontos específicos. Por conseguinte, tal aspecto torna estas soluções não previsíveis a todos os ambientes de bancos de dados que provêem suporte às aplicações. Por isso, pode-se dizer que tais soluções não são plenamente satisfatórias, no sentido de serem empregadas em ambientes computacionais onde, por exemplo, critérios de corretude mais fortes como é o caso da serialização de transações, são requeridos.

Em se tratando da validação atômica não-bloqueante de transações, como esta tese visa a proposição de soluções para cenários onde é admissível a ocorrência de falta bizantinas, as soluções encontradas na literatura apresentam diversas limitações. Por exemplo, ambos os trabalhos de Mohan *et al.* (1983) e Zhao (ZHAO, 2007) são protocolos baseados no 2PC, e portanto, não atendem às propriedades especificadas para o NBAC (BABAOGLU; TOUEG, 1993). Por outro lado, é plausível que argumentações contrárias possam vir a discorrer sobre o fato de que não é interessante resolver o NBAC com faltas bizantinas, já que o protocolo 2PC é um padrão *de facto* em sistemas onde se emprega algum tipo de validação distribuída de transações (SHAHZA et al., 2008). Todavia, se assim o for, note que todos os sistemas transacionais terão que ser compatíveis com o 2PC. Porém, a situação real mostra que isso ainda não é um consenso, e tampouco o caso para aplicações que requerem maiores garantias e confiabilidade acerca da terminação de transações.

Por esta razão, acredita-se que as limitações apontadas no estudo realizado neste capítulo constituem um forte argumento para a investigação, proposição e especificação de soluções que sejam capazes de resolvê-las. O que, por conseguinte, justifica os interesses de

investigação desta tese acerca da replicação de bancos de dados e da validação atômica não-bloqueante de transações, ambos em ambientes com sujeição a faltas bizantinas.

4 PROTOCOLO TOLERANTE A FALTAS BIZANTINAS PARA O PROCESSAMENTO DE TRANSAÇÕES EM BANCOS DE DADOS RELACIONAIS

Este capítulo versa sobre a primeira contribuição desta tese, onde é apresentado um protocolo para o processamento de transações a partir de bancos de dados relacionais sujeitos à faltas bizantinas. Neste contexto, o capítulo apresenta a formalização de protocolo através da especificação de um conjunto de algoritmos, e suas respectivas provas de corretude. Não obstante, é apresentada uma especificação de arquitetura de *middleware* que é resultado da implementação dos algoritmos desenvolvidos no presente capítulo.

4.1 INTRODUÇÃO

Conforme verificado no decorrer dos capítulos anteriores, os sistemas de bancos de dados são uma tecnologia vital para grande parte dos sistemas computacionais existentes na atualidade. Por outro lado, a literatura mostra que faltas de natureza arbitrária, tal como as bizantinas, são uma realidade na maioria dos sistemas computacionais da atualidade (DRISCOLL et al., 2003; SCHROEDER; GIBSON, 2007; SCHROEDER; PINHEIRO; WEBER, 2009; NIGHTINGALE; DOUCEUR; ORGOVAN, 2011), dentre os quais se incluem os sistemas de bancos de dados (GASHI; POPOV; STRIGINI, 2004). Embora as faltas de parada sejam, indiscutivelmente, as faltas mais frequentes, faltas oriundas de causas como corrupção de dados em disco ou em memória RAM devido a efeitos físicos (SCHROEDER; PINHEIRO; WEBER, 2009; NIGHTINGALE; DOUCEUR; ORGOVAN, 2011) ou ainda no software devido a *bugs* (GASHI; POPOV; STRIGINI, 2007), são bastante comuns nos sistemas computacionais (cfm. Seção 3.1). Os efeitos causados por tais situações podem ser caracterizados como faltas bizantinas, sendo que na maioria das vezes, eles não causam a parada abrupta do sistema, mas sim a corrupção do estado da aplicação. Em face da complexidade presente na concepção dos sistemas computacionais modernos, a propensão à ocorrência de faltas bizantinas nestes é cada vez mais crescente, principalmente pelas vulnerabilidades que são recorrentes nos mesmos (DRISCOLL et al., 2003). No caso específico de sistemas de bancos de dados, uma boa discussão acerca da influência de faltas bizantinas nestes sistemas

foi elucidada por Gashi, Popov e Strigini (GASHI; POPOV; STRIGINI, 2004, 2007).

Curiosamente, os trabalho de Gash, Popov e Strigini (2004, 2007) demonstraram que, historicamente, muitos erros têm sido encontrados em SGBDs comerciais, erros estes que muitas vezes não afetam a execução, mas comprometem o estado do banco de dados e incorrem diretamente na exatidão/corretude das operações executadas pelas transações sobre tais SGBDs. Note que a ocorrência de faltas por parada em SGBDs também pode comprometer a consistência dos dados. Porém, a manifestação de faltas bizantinas, ou mesmo os erros oriundos de *bugs* contidos no SGBD, podem resultar na execução incorreta de operações contidas nas transações, por exemplo, com valores incorretos retornados como resultado para operações de leitura; a escrita de valores incorretos sobre os itens de dados armazenados no banco de dados – situações que muitas vezes não impedem as transações de serem validadas (i.e., *commit*). Embora a manifestação de uma falta bizantina eventualmente possa incorrer em uma parada, quando esta não ocorre, é difícil de estimar se as operações executadas no âmbito de uma transação o foram de maneira correta. Neste caso, torna-se difícil mascarar o comportamento bizantino manifestado durante o processamento de uma transação, justamente porque não se pode estimar com exatidão se uma dada operação da transação foi executada corretamente, pelas razões já expostas.

Neste contexto, a proposta introduzida pela tolerância a faltas bizantinas no âmbito de sistemas computacionais é uma alternativa que pode ser empregada para tolerar faltas de software, hardware, sejam estas benignas ou maliciosas, por meio do uso de técnicas de replicação de componentes. Todavia, embora haja diversas propostas para se tolerar faltas bizantinas baseadas em replicação para sistemas computacionais, poucas iniciativas foram desenvolvidas especificamente para o âmbito de bancos de dados e transações. Esta é uma das razões que nos levar a crer que é imperioso investigar por estratégias que permitam mitigar os efeitos destas faltas no processamento de transações em sistemas de bancos de dados. É evidente que a manifestação deste tipo de faltas pode causar efeitos graves e irreversíveis ao sistema (p. ex.: exclusão e corrupção de dados, violação de consistência, etc.).

No que diz respeito ao processamento de transações em bancos de dados, num ambiente não-replicado a ocorrência de uma falha no âmbito do SGBD pode causar a parada imediata do sistema. No caso, como os sistemas que acessam um banco de dados geralmente o fazem por meio do processamento de transações, eles podem tirar vantagem

da semântica transacional para fins de recuperação do estado da aplicação e, neste caso, o único impacto sofrido pela aplicação seria aquele decorrente da inatividade do banco de dados. Porém, o sistema estaria indisponível até a recuperação do banco de dados, o que muitas vezes não é admissível dependendo da criticidade e do ambiente no qual o mesmo está em operação. E por assim dizer, é imperioso que sistemas de bancos de dados também sejam capazes de operar de maneira correta, a fim de oferecer um suporte confiável para o processamento de transações sobre os dados de sua responsabilidade. Não obstante, a necessidade de se melhorar alguns aspectos concernentes à tolerância a falhas nestes sistemas é intensa. Da mesma maneira que ocorre nos sistemas computacionais de propósito mais geral, uma alternativa bastante usual e plausível para se obter maior disponibilidade de dados em um banco de dados é também através da replicação (KEMME; PERIS; PATIÑO-MARTÍNEZ, 2010). Do mesmo modo, a replicação é interessante do ponto de vista de se manter algumas cópias (ou réplicas) do banco de dados, na medida em que ela permite reduzir substancialmente a correlação entre falhas das diferentes réplicas e não torna o(s) dado(s) inacessível(is) no caso da falha em alguma(s) delas.

Conforme visto na Seção 3.1.2, em se tratando de sistemas distribuídos as soluções empregadas para lidar com faltas bizantinas são em sua essência, protocolos baseados na Replicação de Máquina de estados (RME) (SCHNEIDER, 1990). Todavia, uma limitação quanto ao uso da RME no âmbito de transações é que esta abordagem de replicação garante uma consistência estritamente forte como a linearização (HERLIHY; WING, 1990), mas apenas para a execução de cada operação em individual. Não obstante, como o isolamento é uma propriedade requerida no processamento de transações, os bancos de dados tipicamente adotam protocolos de bloqueio para prover a exclusão mútua na manipulação dos itens de dados no contexto de transações (vide Seção 2.1.3.1). Por sua vez, se implementado através de uma RME, esta característica pode induzir o sistema de processamento de transações a um estado de bloqueio indefinido, de forma a causar um impasse quando ao menos duas transações concorrentes acessam um mesmo item de dados, independentemente da ordem na qual aquele item está sendo acessado. Este problema é ilustrado na Figura 32.

A Figura 32 apresenta o caso onde os clientes c_1 e c_2 estão a executar suas transações T_1 e T_2, respectivamente. No caso, ambos os clientes enviam uma operação de escrita sobre o item de dados x para execução, onde estas operações são entregues na ordem w_1 e w_2 em todas as réplicas, já que um protocolo de difusão com ordem total

(DÉFAGO; SCHIPER; URBÁN, 2004) é utilizado para cumprir o requisito acordo/ordem da RME (SCHNEIDER, 1990). No cenário em questão, como a mensagem que contém a operação w_1 é entregue primeiro, a transação T_1 adquire um bloqueio sobre x e a operação é executada. Então a mensagem com a operação w_2 é entregue e assim, quando a transação T_2 tenta realizar a operação de escrita em x ela é posta para aguardar, já que o item de dados x se encontra bloqueado pela transação T_1. A considerar que a execução de operações sobre a RME seguem o critério de linearização, a operação w_2 não poderá ser completada em razão do bloqueio, e por conseguinte, nenhuma mensagem ordenada posteriormente a que continha a operação w_2 será entregue para a RME – um requisito necessário para assegurar a linearização. E neste caso, como a operação w_2 só será concluída após a validação (*commit*) ou anulação (*abort*) de T_1, o sistema entra numa situação de impasse, pois nenhuma operação será executada até a conclusão de w_2 – o que vale também para a terminação de T_1.

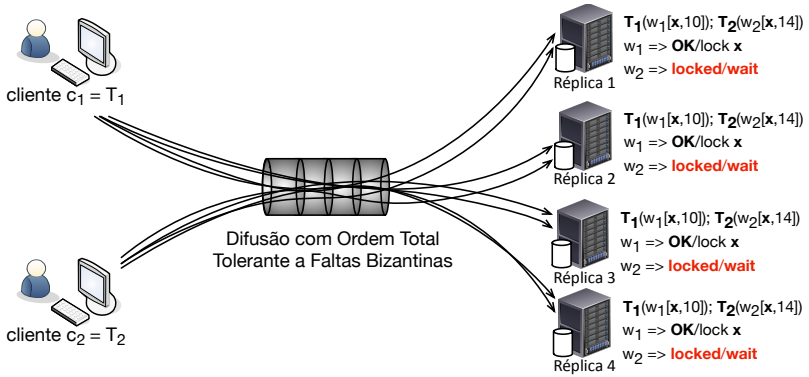


Figura 32 – Execução de transações a partir de uma RME BFT.

Uma alternativa ao uso da RME para o processamento de transações é a RMEDB (PEDONE; GUERRAOUI; SCHIPER, 2003), todavia, esta abordagem foi especificada apenas para o suporte a faltas por parada, de modo que faltas bizantinas não são suportadas. Neste sentido, um dos elementos que se busca no âmbito desta tese, é a proposição e especificação de um protocolo nos mesmos moldes da RMEDB, que seja capaz de tolerar faltas bizantinas no decurso da transação. Os detalhes acerca da formalização deste protocolo são apresentados nas próximas seções deste capítulo.

4.2.1 Predicados, Bloqueios, Conflitos e Precedência

Comumente a literatura define uma transação como uma sequência de operações de leitura e/ou de escrita executadas em um SGBD de maneira agrupada (GRAY; REUTER, 1992; CONNOLLY; BEGG, 2005; GARCIA-MOLINA; ULLMAN; WIDOM, 2009; BERNSTEIN; NEWCOMER, 2009). Esta definição induz o leitor a pensar em operações de leitura ou escrita sobre itens de dados pontuais e nomeados. Todavia, como a manipulação de dados, nos bancos de dados relacionais da atualidade, é realizada a partir de instruções de alto nível escritas na linguagem SQL (GARCIA-MOLINA; ULLMAN; WIDOM, 2009), muitas vezes as operações são formadas por instruções complexas, o que inclui inserções, exclusões, atualizações e até mesmo a seleção de um conjunto de dados que atenda a uma determinada condição. Neste sentido, nesta tese considera-se que as transações são especificadas a partir de instruções da linguagem SQL, de modo que uma transação pode realizar leituras baseadas em predicados (ou *predicate reads*) (GRAY; LORIE; PUTZOLU, 1975), escrita de itens de dados e leitura de itens de dados.

Uma leitura baseada em predicado é uma operação que identifica um conjunto de itens de dados contidos no banco de dados, no qual estes itens de dados satisfazem a um determinado predicado, de acordo com o estado atual do banco de dados. A linguagem SQL adota o uso de predicados para realizar operações de leitura ou atualização de itens de dados (ou linhas/registros). Como os sistemas de gerência de bancos de dados comerciais da atualidade adotam esquemas baseados em bloqueios (cfm. Seção 2.1.3.1) para prover a exclusão mútua dos itens de dados afetados pelas instruções executadas no banco de dados (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007), no caso das leituras baseadas em predicados, o bloqueio não ocorre sobre o predicado, mas individualmente sobre cada um dos itens de dados que são satisfeitos pelo predicado fornecido na operação.

Em ambientes de banco de dados, a abordagem mais empregada para efetuar a verificação acerca da serialização e corretude da transação, é aquela baseada na noção de conflitos existentes entre as transações executadas simultaneamente (vide Seção 2.1.3). Para tanto, a literatura adota os termos *read-set* (ou *RS*) e *write-set* (ou *WS*) para denotar os itens de dados que foram afetados por uma transação em execução (KEMME; PERIS; PATIÑO-MARTÍNEZ, 2010). O *write-set*

contém os itens de dados que foram alterados no banco de dados no âmbito da transação, enquanto que o *read-set* contém os itens de dados que foram apenas lidos pela transação. Estes conjuntos por sua vez, consistem no mecanismo utilizado para viabilizar a detecção de conflitos entre as transações. No caso, pode-se dizer que o *write-set* contém os itens de dados que são afetados pela execução dos comandos SQL `INSERT`, `DELETE` e `UPDATE`, de modo que o *read-set* é populado a partir da execução do comando `SELECT` e também dos comandos `DELETE` e `UPDATE` – estes últimos realizam a leitura prévia dos itens de dados antes de proceder com as alterações.

No âmbito do protocolo a ser apresentado neste capítulo, a execução de transação é realizada da seguinte maneira: (*i*) a aplicação solicita o início de uma transação a partir de uma instrução/operação `BEGIN`, e; (*ii*) após iniciada, operações de leitura e/ou de escrita (p. ex.: instruções SQL) são realizadas no banco de dados, e por fim; (*iii*) solicita a validação (`COMMIT`) ou anulação (`ABORT`) da transação. Do mesmo modo, a lista de operações que compõem a transação, incluindo as instruções de início e de término, é denominada unidade transacional. Duas transações T_i e T_j, tal que $i \neq j$, são concorrentes se há uma interseção entre o intervalo das operações `BEGIN`(T_i) e `COMMIT`(T_i) e do intervalo `BEGIN`(T_j) e `COMMIT`(T_j), os quais são denotados pelas marcas de tempo (i.e., *timestamp*) obtidas quando da execução de tal operação. Esta marca de tempo consiste na ordem em que o protocolo de difusão com ordem total entregou a mensagem. De maneira oposta, isto é, quando não há uma interseção entre os intervalos descritos e o *timestamp* da operação `COMMIT`(T_i) é menor do que o *timestamp* da operação `BEGIN`(T_j), é dito que T_i precede T_j, o que é formalmente denotado pela expressão $T_i \mapsto T_j$. A partir daí, é caracterizado como conflito o cenário no qual T_i e T_j são concorrentes (i.e., $T_i \nmapsto T_j$ ou $T_j \nmapsto T_i$) e a seguinte condição é verificada:

$$(RS(T_j) \cap WS(T_i) \neq \varnothing) \vee (WS(T_j) \cap RS(T_i) \neq \varnothing). \tag{4.1}$$

Para uma melhor compreensão acerca da relação de precedência entre transações e verificação de conflitos empregados no protocolo, a Figura 33 ilustra a forma pela qual o algoritmo estabelece tal relação, a fim de determinar quais transações devem ser consideradas para fins de certificação. Esta verificação é realizada com base na ordem de entrega das mensagens que denotam os eventos de sincronização `BEGIN` e `COMMIT`, das transações executadas no âmbito do protocolo (i.e., seus

timestamps). Esta é a razão pela qual é empregada a difusão com ordem total para o evento que denota o início da transação (i.e., da mensagem `BEGIN`). Neste caso, é associado um *timestamp* a cada evento de entrega de mensagem ocorrido pelo protocolo de difusão com ordem total subjacente, o qual indica a ordem de entrega daquela mensagem, sendo este incrementado em uma unidade a cada evento de entrega ocorrido. No exemplo ilustrado na Figura 33, é revelada a relação de precedência para a transação T_8, onde $ts(valor)$ indica o *timestamp* da operação em questão (i.e., `BEGIN` ou `COMMIT`). Como T_8 tem início no *timestamp* 10, todas as transações cuja entrega da mensagem `COMMIT` ocorreu antes de $ts(10)$ são consideradas como precedentes em relação à T_8, como é o caso de T_1. É digno de nota, que uma transação precedente não é considerada para fins de certificação.

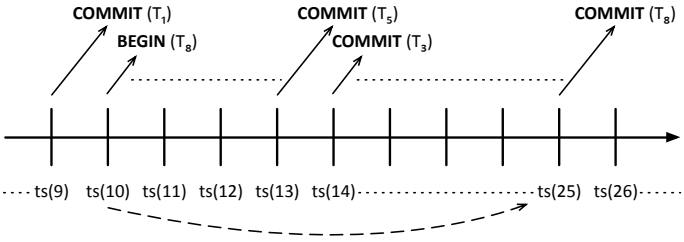


Figura 33 – Relação de precedência quanto à validação da transação.

Por outro lado, todas as transações cuja mensagem `COMMIT` é entregue no intervalo entre os eventos de sincronização `BEGIN` e `COMMIT` de T_8, são consideradas concorrentes à T_8. Considere ***ts(x)*** como o *timestamp* da mensagem `COMMIT` da transação a ser analisada, se $ts(10) <$ ***ts(x)*** $< ts(25)$ ela é concorrente à T_8. Este é o caso das transações T_5 e T_3 na Figura 33, pois, embora elas tenham sido iniciadas antes de T_8, suas validações ocorreram no intervalo entre o início e a terminação de T_8, nos *timestamps* 13 e 14, respectivamente. Nesta condição, T_5 e T_3 são consideradas concorrentes em relação à T_8, porque as escritas efetuadas por elas sobre os itens de dados do banco de dados não puderam ser lidas por T_8. E como a fase de `execução` da transação ocorre de maneira otimista, se T_8 foi executada por uma réplica líder diferente das que executaram T_5 e T_3; e T_8 faz referência a algum item de dados escrito por T_5 ou T_3 (ou ambas), T_8 pode estar em conflito com T_5 e/ou T_3. A verificação acerca dos conflitos é realizada pelo teste de certificação, cuja finalidade é determinar pelos conflitos, se a

transação está em consonância com o critério de serialização e pode ser validada, ou do contrário, se ela deve ser anulada.

4.2.2 Modelo de Sistema

O modelo de sistema consiste em um ambiente de sistema distribuído, no qual o universo de processos $\mathcal{U}$ é subdividido em dois conjuntos, nomeadamente $\mathcal{C} = \{c_1, c_2, ..., c_n\}$ e $\mathcal{R} = \{r_1, r_2, ..., r_n\}$. O conjunto $\mathcal{C}$ denota os processos que implementam os clientes do serviço e contém um número arbitrário (não infinito) de elementos. De outro modo, o conjunto $\mathcal{R}$ denota os processos que implementam as réplicas no ambiente de banco de dados confiável e tolerante a faltas bizantinas. Neste caso, $\mathcal{R}$ tem cardinalidade $|\mathcal{R}| \geq 3f + 1$, o que é necessário para que seja possível tolerar até $f \leq \lceil \frac{n-1}{3} \rceil$ réplicas com comportamento(s) bizantino(s). Por questões de corretude do protocolo, assumimos que um número ilimitado de clientes podem falhar apenas por parada (*crash*), de modo que eles não manifestam faltas bizantinas.

Para o caso das réplicas, consideramos que aquelas faltosas podem desviar arbitrariamente de suas especificações, podendo parar, omitir o envio, a recepção ou a entrega das mensagens, enviar respostas incorretas às operações dos clientes, bem como atuar em conluio com outras réplicas faltosas visando a corrupção do ambiente de banco de dados. Por outro lado, assumimos a independência de falhas, de modo que a ocorrência de uma falta em determinada réplica é independente da ocorrência daquela falta nas demais. Esta premissa pode ser substanciada na prática através do uso de diversidade (OBELHEIRO; BESSANI; LUNG, 2005).

Considera-se que toda a comunicação entre os processos ocorre por meio de passagem de mensagens, que podem acontecer a partir de primitivas um-para-um ou um-para-muitos. A comunicação um-para-um se dá a partir canais ponto-a-ponto confiáveis, autenticados e com ordem FIFO (i.e., *fist-in first-out* para que as mensagens sejam recebidas na ordem em que foram enviadas), e é definida pelas primitivas *send* e *receive*. De outro modo, a comunicação um-para-muitos depende de um suporte para difusão com ordem total tolerante a faltas bizantinas, o qual é definido pelas primitivas *TO-multicast* e *TO-deliver* – estas garantem que todos os processos não faltosos (ou corretos) concordem com a entrega das mensagens em uma mesma ordem (DÉFAGO; SCHIPER; URBÁN, 2004) (maiores detalhes na Seção 2.2.3.3).

É assumido que um banco de dados $\mathcal{D}$ consiste em uma coleção

finita de itens de dados, tal que $\mathcal{D} = x_1, x_2, ..., x_n$. Cada item de dados x_i, tal que $1 \leq i \leq n$, está associado à uma chave que identifica de maneira única o item de dados, e a um conjunto arbitrário (não infinito) de atributos com valores. Para tanto, no âmbito do protocolo proposto a granularidade de um item de dados compreende a um registro/linha/tupla, termos que denotam uma ocorrência de item de dados existente no banco de dados e recuperado pela transação (WEIKUM; VOSSEN, 2002; RAMAKRISHNAN; GEHRKE, 2003; CONNOLLY; BEGG, 2005). Os clientes interagem com o ambiente de banco de dados por meio da invocação de transações, isto é, unidades atômicas de execução composta por um conjunto de operações de leitura/escrita sobre os itens de dados. E conforme visto na Seção 4.2.2, as operações de uma transação são delimitadas pelas instruções `BEGIN` e `COMMIT` ou `ABORT`. É assumido que um cliente submete apenas uma transação por vez, e no contexto de uma transação uma operação é enviada apenas se não há operações pendentes de execução.

Transações constituídas exclusivamente por operações de leitura são denominadas por transações de somente-leitura, e as transações que contenham qualquer operação que modifique algum dado são denominadas transações de atualização. No âmbito de uma transação, o protocolo provê suporte ao modelo CRUD (acrônimo para *Create*, *Read*, *Update* e *Delete*) no que concerne à manipulação de itens de dados no banco de dados. Duas importantes suposições são consideradas, a fim de preservar as propriedades do protocolo, as quais são: (*i*) é assumido o modelo de transações planas, isto é, transações de apenas um nível (ou *flat transactions*), de modo que transações aninhadas não são tratadas pelo protocolo, e portanto, não consideradas; (*ii*) o protocolo não provê suporte à execução de operações de definição de dados no âmbito de uma transação (i.e., DDL transacional), sendo esta, uma premissa bastante razoável já que na prática, muitos SGBDs comerciais não suportam estes tipos de comandos (FIREBIRD, 2014; ORACLE, 2014b, 2014a).

As suposições acerca da consistência do ambiente replicado visam o critério de corretude baseado na serialização das transações (cfm. Seção 2.1.3), isto é, o critério mais rigoroso e livre de quaisquer anomalias e fenômenos relacionados à concorrência (cfm. Seção 2.1.5). Tal hipótese implica que cada réplica de banco de dados deve suportar, para a execução local de transações, o isolamento serializável (ANSI, 1992). Para assegurar a consistência e corretude do banco de dados, algumas suposições são requeridas no que concerne às réplicas de banco de dados (ou SGBDs): (*i*) todas são relacionais e transacionais; (*ii*) elas supor-

tam a reversão de operações (p. ex.: *rollback*); (*iii*) elas processam localmente as transações, em conformidade com a semântica do protocolo de bloqueio em duas fases rigoroso (*strict two phase locking* – 2PL rigoroso); (*iv*) as operações modificam o banco de dados de forma atômica, sem efeitos colaterais.

Não obstante, por se tratar de um protocolo baseado na replicação de banco de dados, cópias de um item de dados devem ser armazenadas em cada réplica no sistema, sendo que as várias cópias de um mesmo item de dados devem aparecer como um único item de dados lógico no contexto de uma transação. Esta propriedade é conhecida como **equivalência de uma cópia** (*one-copy equivalence*) (WEIKUM; VOSSEN, 2002), e deve ser atendida pelo protocolo de replicação. Neste sentido, o critério de consistência para o ambiente replicado de banco de dados é o *one-copy serializability* (BERNSTEIN; HADZILACOS; GOODMAN, 1987) – que garante não apenas a equivalência de uma cópia, mas também a execução serializável de transações, a despeito das múltiplas cópias de itens de dados existentes. A especificação do protocolo considera que cada réplica mantém uma cópia completa do banco de dados. As réplicas são deterministas, logo, a execução de uma operação produz o mesmo resultado em todas as réplicas.

E, finalmente, como o protocolo depende de uma primitiva de difusão com ordem total tolerante a faltas bizantinas, a qual é baseada no algoritmo de consenso Paxos Bizantino (ZIELINSKI, 2004), o modelo assume uma premissa de sincronismo fraca para assegurar não apenas a terminação do protocolo de consenso, mas também das transações: os atrasos de comunicação não crescem exponencialmente. Para tanto, considera-se como modelo de interação aquele baseado em sincronismo terminal (ou *eventually synchronous*) (DWORK; LYNCH; STOCKMEYER, 1988) – um modelo bastante realista já que se porta de forma assíncrona em grande parte do tempo, mas durante períodos de estabilidade o tempo de transmissão de mensagens e das computações é limitado, porém desconhecido. Por fim, considera-se também a existência de uma função de resumo criptográfico resistente a colisão (i.e., *hash*) e uma função para a autenticação de mensagens, ambas para assegurar a integridade e a autenticidade das mensagens (BELLARE; CANETTI; KRAWCZYK, 1996). É assumido que uma entidade bizantina não pode subverter os mecanismos criptográficos adotados (p. ex.: ela não dispõe de recursos computacionais para tal).

4.3 VISÃO GERAL DO PROTOCOLO

Embora a replicação de dados seja uma tecnologia de grande maturidade, em se tratando de soluções baseadas em replicação para o processamento de transações em ambientes sujeitos a faltas bizantinas, ela ainda pode ser considerada como um problema desafiador. Apesar da literatura dispor de algumas soluções que visam a resolução do problema, os trabalhos encontrados apresentam algumas limitações e são focados em problemas e ambientes mais específicos, de modo que o problema não é coberto em sua plenitude.

Por exemplo, o HRDB (VANDIVER et al., 2007) é uma solução que depende de um elemento centralizado e que não pode falhar, para conduzir um protocolo de replicação de bancos de dados tolerantes a faltas bizantinas. Dependendo do caso, é difícil justificar o pressuposto de se ter um elemento confiável num ambiente sujeito a faltas bizantinas. Por outro lado, o Byzantium (GARCIA; RODRIGUES; PREGUIÇA, 2011) é um *middleware* para lidar com a replicação de bancos de dados BFT, em que o protocolo opera de maneira distribuída, sem depender de um elemento confiável centralizado. No entanto, o Byzantium é especificado considerando apenas o processamento de transações segundo a semântica de consistência baseada no *snapshot isolation* (BERENSON et al., 1995). Embora a consistência provida pelo *snapshot isolation* seja bastante atrativa e adotada por diversos SGBDs comerciais (ORACLE, 2014b; FIREBIRD, 2014; MICROSOFT, 2014), em algumas circunstâncias ela permite a corrupção de dados por meio da intercalação de transações, que individualmente preservam a consistência (NORMANN; OSTBY, 2010).

Um aspecto que corrobora para tal afirmação decorre do fato de que alguns trabalhos têm estudado alternativas para se obter a serialização completa sobre a semântica do *snapshot isolation* (FEKETE et al., 2005; CAHILL; RÖHM; FEKETE, 2008). Alguns destes demonstraram que isto só é possível se forem realizadas modificações em nível de aplicação (FEKETE et al., 2005; CAHILL; RÖHM; FEKETE, 2008) – o que vai contra um dos princípios básicos para o uso de bancos de dados por parte das aplicações, isto é, o de facilitar o desenvolvimento de aplicações de tal forma que os desenvolvedores não precisem se preocupar com problemas de integridade de banco de dados decorrente da execução concorrente de transações. Cabe salientar que alguns fabricantes de SGBDs comerciais, por meio dos manuais de utilização do produto, advertem os desenvolvedores quanto à possíveis problemas de consistência, caso alguns cuidados não sejam adotados no desen-

volvimento das aplicações. Dentre as soluções apontadas também se encontra o BFT-DUR (PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO, 2011), que é especificado apenas para o processamento de transações sobre dados contidos em sistemas de armazenamento chave-valor, e portanto, não está alinhado aos objetivos desta tese.

Neste ensejo, o protocolo a ser apresentado nesta seção visa prover uma nova solução para o suporte ao processamento de transações em bancos de dados relacionais tolerantes a faltas bizantinas, a partir da estratégia de replicação baseada em certificação (*certification-based database replication*) (WIESMANN et al., 2000), nos mesmos moldes da RMEDB (PEDONE; GUERRAOUI; SCHIPER, 2003) – que tolera apenas faltas por parada. A estratégia escolhida se baseia na abordagem otimista/especulativa quanto ao processamento da transação – similar ao modelo estabelecido por Kung e Robinson (1981) –, e na abordagem de propagação *deferred update* (BERNSTEIN; HADZILACOS; GOODMAN, 1987). No caso, pode-se dizer que o protocolo consiste em uma extensão da abordagem clássica de replicação de bancos de dados baseada em certificação para lidar com faltas bizantinas, ao mesmo tempo que pode ser vista como uma RMEDB tolerante a faltas bizantinas. A escolha destas abordagens/estratégias se justificam por diversas razões, dentre as quais a principal é o desempenho bastante satisfatório verificado em ambientes com faltas por parada (PEDONE; GUERRAOUI; SCHIPER, 2003; CORREIA JR. et al., 2005).

O primeiro e principal objetivo do protocolo é superar algumas das limitações verificadas nos trabalhos encontrados na literatura, que tratam de replicação de bancos de dados para tolerar faltas bizantinas. Neste sentido, se tem como perspectivas preservar as propriedades básicas de transações – ACID (Atomicidade, Consistência, Isolamento, Durabilidade) e o critério de consistência baseado em serialização, a fim de que o sistema se mantenha correto e consistente, mesmo que até $f \leq \lceil \frac{n-1}{3} \rceil$ réplicas se desviem arbitrariamente de suas especificações. Também se busca obter uma solução escalável e com desempenho aceitável através do processamento otimista de transações (KUNG; ROBINSON, 1981) sobre um banco de dados replicado, a partir da abordagem baseada em certificação (WIESMANN et al., 2000). Para tanto, durante o processamento/execução da transação a idéia consiste em permitir que o cliente interaja apenas com uma réplica, denominada de primária/líder, de modo que a propagação das operações que compõem a transação para as demais réplicas ocorra apenas na terminação da transação, segundo a estratégia do *deferred update* (BERNSTEIN; HADZILACOS; GOODMAN, 1987).

Um aspecto que é digno de nota e esclarecimento, é que o propósito da solução apresentada nesta parte da tese, e portanto, do protocolo objeto desta seção, é evitar que faltas bizantinas ocorridas no contexto de transações executadas em bancos de dados relacionais violem a semântica de consistência serializável do banco de dados. Note que a solução não impede a execução/ocorrência de ações oriundas de faltas bizantinas, mas evita que seus efeitos sejam refletidos no estado do banco de dados. Isso posto, e a considerar os aspectos explanados nos parágrafos anteriores, acredita-se que a solução proposta deve superar algumas das limitações verificadas para os demais protocolos.

4.4 DINÂMICA DE FUNCIONAMENTO DO PROTOCOLO

Um aspecto particularmente interessante que decorre da semântica transacional, e portanto, adotada para a especificação do protocolo, é que, de acordo com a definição de isolamento (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007), transações em execução simultânea não visualizam os resultados intermediários de alterações efetuadas por transações ainda não concluídas e validadas, mas apenas do último estado válido do banco de dados que precede o início daquelas que se encontram em execução. Com isso, os resultados de uma transação só tomam efeito no banco de dados após a validação da mesma (ou *commit*), onde as alterações daquela transação passam a integrar o estado do banco de dados. Esta importante característica pode ser encarada como algo determinante para a especificação de um protocolo transacional BFT, a despeito do requisito **determinismo de réplica** estipulado pela RME, em que cada operação deve ser totalmente ordenada antes de sua execução pela máquina de estados (cfm. Seção 2.2.3.4). Neste sentido, levando em consideração a semântica transacional, é possível especificar um protocolo de replicação de bancos de dados baseado no conceito de máquina de estados, tal como a RME, mas a partir do enfraquecimento parcial do requisito de ordenação.

Em conformidade com a semântica transacional, onde a transação toma efeito apenas quando da sua validação, o requisito de ordenação deve ser atendido somente no momento da validação da transação, de modo que não é necessário adotá-lo para a execução de cada operação. Neste caso, a corretude e consistência dependerá da ordenação total do procedimento que realizará a validação da transação, portanto, não sendo necessária a ordenação total de todas as operações que compõem a unidade lógica da transação. O protocolo proposto se apoia

sobre esta hipótese, razão pela qual emprega um protocolo de difusão com ordem total apenas para a execução dos eventos de sincronização da transação (i.e., *begin*, *commit*, *abort*), em vez de usá-la para a execução de cada operação, tal como ocorre na RME clássica. No protocolo proposto, a execução dos eventos de sincronização compreendem às fases de `início` e `terminação` da transação. Note que um ambiente de banco de dados replicado implementado como uma máquina de estados (PEDONE; GUERRAOUI; SCHIPER, 2003) requer a ordenação total apenas da `terminação` da transação; porém, o protocolo proposto requer a ordenação total também para o `início` da transação, para que seja possível assegurar as propriedades e a consistência do banco de dados. Maiores detalhes serão apresentados adiante neste capítulo.

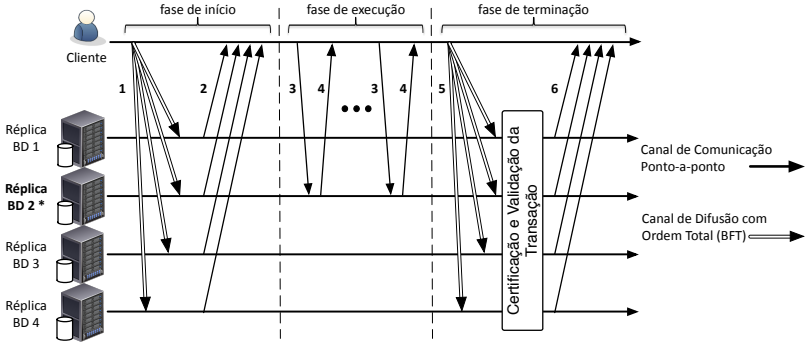


Figura 34 – Diagrama de execução do protocolo em três fases.

A Figura 34 ilustra um diagrama temporal, o qual representa a execução de uma transação a partir do protocolo proposto. Note que o protocolo realiza o processamento de uma transação através de três fases, os quais representam o ciclo de vida da transação: `início`, `execução` e `terminação`. Para cada transação executada, uma das réplicas é selecionada com líder/primária e as restantes são secundárias. No caso da ilustração em questão, a réplica 2 que está destacada em negrito é a líder daquela transação. No decurso da transação, a réplica líder será a responsável por executar de maneira otimista as operações da transação[1] (fase de `execução`), e portanto, será a única a interagir com o cliente até o momento da validação da transação (fase de `terminação`). É importante salientar que o modelo de transações pra-

[1] otimista quanto à execução da transação no ambiente replicado, o que não deve ser confundido com o controle de concorrência otimista

ticado no protocolo é o interativo (PEDONE; GUERRAOUI, 1997), isto quer dizer que entre cada duas operações há interação do cliente com a réplica líder, de modo que, quando o cliente submete uma operação, ele aguarda o resultado desta para então submeter uma próxima operação – i.e., ele não submete mais de uma operação simultaneamente. No que segue, uma breve explicação sobre cada uma das fases é realizada, sendo que os detalhes acerca da especificação do protocolo serão explicados nas seções posteriores, por meio da formalização dos algoritmos.

Fase de Início. Em suma, uma transação tem início quando um cliente abre uma conexão no banco de dados, o que é realizado a partir do envio da operação `BEGIN` a todas as réplicas através da primitiva de difusão com ordem total (etapa 1). Ao entregar esta mensagem, as réplicas atribuem um mesmo *id* único para a transação – o que é possível devido a elas entregarem a mensagem na mesma ordem. Após atribuir o *id* para a transação, elas definem de maneira determinista, a réplica que será a líder e terá o papel de executar de maneira otimista as operações que compõem a transação. Ao concluir esta primeira etapa, cada réplica envia uma notificação ao cliente contendo o *id* único da transação, e também as informações da réplica líder, para que o cliente possa enviar as operações diretamente a ela (etapa 2). O cliente por sua vez, aceita o resultado da operação se recebe pelo menos $f + 1$ notificações mutuamente consistentes e advindas de diferentes réplicas.

Fase de Execução. Esta fase é ilustrada pelas etapas 3 e 4 da Figura 34, as quais são repetidas até que não haja mais operações a serem executadas para a transação. A fase tem início no momento em que o cliente envia para a réplica líder a primeira operação (p. ex.: uma leitura ou escrita) no âmbito da transação (etapa 3). Após receber a operação do cliente, a réplica líder a executa sobre o banco de dados, aguarda até o término desta[2], e então retorna o resultado daquela operação ao cliente (etapa 4). É digno de nota, que durante esta fase não ocorre algum tipo de interação entre as réplicas, já que o protocolo se baseia na abordagem otimista quanto ao processamento da transação, e portanto, as réplicas secundárias daquela transação não recebem as operações nesta fase. A execução da transação termina quando o cliente solicita a validação (`COMMIT`) da transação.

Fase de Terminação. A fase de `terminação` é iniciada quando o cliente solicita a validação da transação pelo envio da operação/mensagem `COMMIT` através de difusão com ordem total a todas as réplicas – in-

[2] A execução da operação no banco de dados é bloqueante

cluindo a líder (etapa 5). Resumidamente, a mensagem `COMMIT` contém todas as operações executadas pela transação e o resumo criptográfico (i.e., um *hash*) dos respectivos resultados recebidos da réplica líder. Ao entregar a mensagem `COMMIT`, as réplicas iniciam um procedimento de certificação, que visa assegurar a serialização da transação, a despeito de conflitos verificados através de transações concorrentes que foram validadas durante o intervalo de execução da transação em questão. Em suma, o procedimento de certificação verifica duas coisas: (*i*) a validade e atualidade do *read-set* (i.e., as leituras) daquela transação, e; (*ii*) se os resultados contidos no *write-set* e obtidos na execução da transação estão em consonância com o que é observado no momento da certificação pela réplica secundária em questão. A atualidade do *read-set* é verificada a partir da intersecção deste com os *write-set*'s das transações concorrentes (maiores detalhes na Seção 4.2.1). Se o resultado da intersecção for um conjunto vazio, a transação em certificação pode ser serializada após a última transação validada. Do contrário, a transação deve ser anulada (`ABORT`) e o efeito intermediário descartado.

Não obstante, a validade do *read-set* é verificada a partir da execução de todas as leituras no banco de dados – o que impede a validação de uma transação que obteve resultados incorretos de uma réplica líder bizantina. Ademais, operações de escrita da transação também são executadas pelas réplicas secundárias – a líder já as executou, e é verificada a exatidão dos resultados pela comparação com aqueles recebidos da réplica líder. Este procedimento também é adotado para impedir a validação de operações espúrias executadas por réplicas bizantinas. Se todas as verificações estiverem de acordo, então a transação é validada e seu estado toma efeito no banco de dados. Ao término desta fase, as réplicas enviam o resultado da transação ao cliente (etapa 6), que o aceita como correto e válido ao receber pelo menos $f+2$ respostas mutuamente consistentes para a transação, enviado por diferentes réplicas. Uma vez que não mais de f réplicas são faltosas, isto impede que o cliente aceite o resultado de uma réplica bizantina, a qual pode ter validado de maneira unilateral uma transação não serializável ou inválida. Note que, como a validação é realizada nas réplicas a partir da entrega da mensagem `COMMIT` pelo protocolo de difusão com ordem total (DÉFAGO; SCHIPER; URBÁN, 2004), isto implica que não apenas a validação, mas também o efeito da transação, serão integrados ao banco de dados na mesma ordem em todas as réplicas. O que, portanto, atende ao critério de correção *one-copy serializability* (BERNSTEIN; GOODMAN, 1985) – vide Seção 2.1.3.2.

4.5 ESPECIFICAÇÃO E DETALHES DO PROTOCOLO

Esta seção apresenta os detalhes acerca da especificação do protocolo que consiste na primeira parte da contribuição desta tese. Neste sentido, são apresentados os algoritmos que dão suporte ao protocolo e suas respectivas provas de correção.

4.5.1 Declaração de Transações

Conforme mencionado na Seção 4.2.2, em bancos de dados relacionais as transações são declaradas por meio de instruções escritas na linguagem SQL, uma linguagem declarativa de alto nível que suporta um conjunto de operações para a leitura e a escrita de dados em um banco de dados. É importante salientar que uma instrução SQL não declara a maneira pela qual a instrução será executada no banco de dados, mas apenas o propósito da operação de leitura ou de escrita sobre os objetos e itens de dados desejados. O protocolo objeto deste capítulo suporta o subconjunto da linguagem SQL provido pelas seguintes instruções:

- Instrução `SELECT` para a execução de operações de leitura sobre objetos e seus itens de dados, e portanto, considerada pelo protocolo como operação de leitura;

- Instrução `INSERT` para a execução de operações de escrita de novos itens de dados sobre os objetos do banco de dados, sendo considerada como operação de escrita;

- Instruções `UPDATE` e `DELETE` para a execução de operações de atualização de itens de dados contidos nos objetos do banco de dados, e estas sendo consideradas como operações de leitura-escrita, já que elas realizam a leitura dos itens de dados que serão atualizados por aquela operação;

- Instruções de controle para o gerenciamento de transações, tais como `BEGIN`, `COMMIT` e `ABORT`, as quais são empregadas para iniciar, validar e anular uma transação no âmbito do protocolo, respectivamente.

4.5.2 Princípio de Funcionamento

A especificação do protocolo ilustrado no diagrama da Figura 34, é norteada pelo mapeamento da transação em uma sequência finita de estados, conforme a Figura 35. Note que as transações partem de um estado inicial e evoluem através de alguns estados intermediários (`active`, `validating` e, em alguns casos `aborted`), até atingir um dos possíveis estados finais (`committed` ou `aborted`), de acordo com a situação da transação. As transições de estado para os estados intermediários acontecem de acordo com o tipo de mensagens recebidas (pelo canal ponto-a-ponto), entregues (pelo protocolo de difusão com ordem total) ou eventos de controle, enquanto que as transições para os estados finais acontecem de acordo com o resultado da transação após a passagem pela certificação (maiores detalhes na Seção 4.5.3.3).

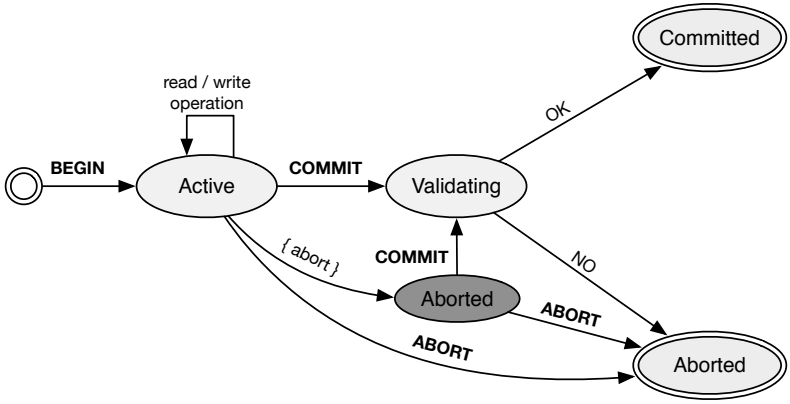


Figura 35 – Máquina de estados que denota as etapas da transação.

De maneira resumida, o caso normal de operação do protocolo para uma transação é o seguinte. Uma transação é iniciada quando um cliente c_i executa uma operação `BEGIN` através de alguma aplicação (p. ex.: um sistema de informação), em que tal operação é enviada às réplicas de banco de dados via protocolo de difusão com ordem total. Ao entregar esta mensagem, as réplicas levam a transação de um estado inicial para o estado `active`. Enquanto no estado `active`, apenas a réplica líder recebe e executa as operações que compõem a unidade lógica da transação. Após a execução de todas as operações da transa-

ção, o cliente solicita a validação da transação, que é efetuada através do envio da mensagem `COMMIT` por meio de difusão com ordem total. Ao entregar a mensagem `COMMIT`, todas as réplicas passam a ter conhecimento das operações executadas na transação e de seus resultados, e assim alteram o estado da mesma para `validating`. Neste momento, é iniciado nas réplicas o processo de validação, onde a transação passa por um teste de certificação para determinar se ela cumpre os critérios de serialização e pode ser validada. E por assim dizer, se o resultado do teste de certificação for positivo, a transação segue para o estado `committed` e torna-se persistente no banco de dados. Do contrário, a transação é conduzida para o estado `aborted` e não toma efeito no banco de dados.

É importante observar que, conforme preconiza o modelo de transações (GRAY; REUTER, 1992; WEIKUM; VOSSEN, 2002), é permitido ao cliente anular a transação sem tê-la concluído. Neste caso, no protocolo proposto o cliente o faz por meio do envio da mensagem `ABORT` para todas as réplicas. As réplicas, ao entregar tal mensagem fazem o seguinte: (*i*) a réplica líder imediatamente descarta as alterações intermediárias da transação, a fim de liberar os bloqueios sobre os itens de dados; (*ii*) todas as demais, incluindo a líder, descartam as variáveis que denotam o contexto daquela transação; e todas modificam seu estado para `aborted`. É importante esclarecer que as operações `BEGIN`, `COMMIT` e `ABORT` são executadas por todas as réplicas, na mesma sequência/ordem equivalente, uma vez que as mensagens destas operações são difundidas via o protocolo de difusão com ordem total.

Note pelo diagrama da Figura 35, que há um estado intermediário `aborted`, o qual ocorre devido à uma anulação local e unilateral da transação (i.e., *local abort*) por parte da réplica líder, quando tal transação estiver em execução naquela réplica. É digno de nota, que a anulação unilateral da transação pela réplica líder, não necessariamente conduzirá a transação para anulação global, no ambiente replicado. Por esta razão, uma transação é conduzida para este estado somente num caso muito específico, o que significa que, em circunstâncias normais, a transação não passa pelo mesmo. Maiores detalhes serão explanados na Seção 4.5.3.1.

4.5.3 Base Algorítmica

No que segue, são apresentados os algoritmos que formam a base algorítmica do protocolo. No intuito de facilitar a compreensão acerca

da especificação do protocolo, os códigos estão divididos em quatro partes, que são apresentadas pelos Algoritmos 1, 2, 3 e 4. No caso, o Algoritmo 1 corresponde à parte do protocolo executada pelo cliente, enquanto que os Algoritmos 2, 3 e 4 são as partes executadas pelas réplicas.

4.5.3.1 Início e Execução da Transação

Nesta seção é apresentada a parte do protocolo que é executada pelos processos que implementam os clientes, que é formalizada a partir do algoritmo 1. Para melhor entendimento acerca do algoritmo em questão, é importante ressaltar o que já fora descrito no modelo de sistema (cfm. Seção 4.2.2), mais precisamente que todas as comunicações são realizadas por meio de canais ponto-a-ponto confiáveis e autenticados, e com ordenação FIFO. Neste caso, as réplicas somente liberam para o protocolo de gerenciamento de replicação as mensagens oriundas de clientes autenticados – para o caso da execução da transação, e de réplicas autenticadas – no caso do protocolo subjacente de difusão com ordem total. Esta autenticação ocorre com o auxílio de códigos de autenticação de mensagens (MACs) (BELLARE; CANETTI; KRAWCZYK, 1996). Do mesmo modo, para evitar que entidades não autorizadas tenham acesso aos objetos e itens de dados armazenados no banco de dados, o protocolo conta com um suporte de controle de acesso baseado em **listas de controle de acesso** (ou *Access Control Lists* – ACLs), o qual é provido pelo SGBD de cada uma das réplicas utilizadas no protocolo.

A parte do protocolo que trata da interação do cliente com as réplicas em todas as fases do protocolo é apresentada no Algoritmo 1. Note que a única atribuição do cliente no âmbito do protocolo é a submissão de operações das transações às réplicas, o que é feito de maneira interativa e por meio da função ***execute_statement*** (código entre as linhas 7 – 49). O código descrito nas linhas 1 a 6 corresponde à declaração das variáveis utilizadas pelo cliente, para a manutenção da transação. Já o código das linhas 50 e 51 é executado apenas quando o cliente recebe uma notificação de anulação não-espontânea da transação por parte da réplica líder – maiores detalhes adiante. Para o gerenciamento da transação no lado cliente, algumas variáveis de controle são definidas: $t.rs$ e $t.ws$, que são utilizadas para armazenar, respectivamente o *read-set* e o *write-set* observado após o recebimento do resultado de cada operação enviada à réplica primária; $t.op$, que consiste num conta-

dor de operações da transação, o qual é incrementado a cada operação enviada pelo cliente para aquela transação; $t.stmts\langle\rangle$ que denota uma lista que contém todas as operações executadas pela transação (p. ex.: $t.stmts\langle 3\rangle$ retorna a terceira operação/instrução executada pela transação) – tal lista é enviada para o protocolo de terminação quando o cliente solicita a validação da transação; e, por fim, $t.leader$ e $t.started$, que indicam, respectivamente, a réplica líder daquela transação e se a transação teve êxito em sua inicialização.

Uma nota a respeito do protocolo é que, conforme citado na Seção 4.2.2, é considerado que um cliente não pode ter mais de uma transação ativa simultaneamente, premissa esta que pode ser relaxada para k transações a partir de uma pequena alteração na condição da linha 8 do Algoritmo 1. Esta simples alteração consiste na inclusão de um contador, o qual seria verificado a cada transação iniciada pelo cliente, de modo a evitar a transposição do valor definido para k. Todavia, para fins de simplificação da especificação no âmbito desta tese, considera-se apenas uma transação ativa por cliente.

Uma transação começa quando o cliente (descrito no algoritmo como c_i) manifesta sua intenção em iniciá-la junto ao banco de dados, o que acontece por meio da execução da instrução `begin`, que resulta no envio de uma mensagem `BEGIN` às réplicas através da difusão com ordem total. Porém, o envio da mensagem `BEGIN` (linha 13) é precedido pela inicialização das variáveis de controle da transação no cliente (linhas 9 a 12), o que ocorre apenas se não há nenhuma transação ativa para este cliente no momento da emissão da instrução `begin` (condição da linha 8). Após o envio do pedido de início da transação às réplicas, o cliente aguarda pelo recebimento de $f + 1$ notificações de êxito no início da transação (i.e., mensagens `ACTIVE` mutuamente consistentes e assinadas, originadas de diferentes réplicas), o que impede o cliente de aceitar notificações unilaterais de réplicas faltosas – que podem querer se passar por líder da transação. Ademais, como no máximo f réplicas podem falhar, o recebimento de $f + 1$ mensagens mutuamente consistentes assegura que, pelo menos, 1 (uma) delas procede de uma réplica correta, e que todas no quórum de $f+1$ possuem conteúdo válido. Pelas mensagens de notificação recebidas, o cliente passa a ter conhecimento de qual réplica foi selecionada como líder/primária para executar sua transação (i.e., pelo valor contido na variável $leader\_id$), e a partir daí passará a contatar somente esta réplica durante a execução da transação (linha 15). Por fim, o cliente define como iniciada a transação e retorna o resultado para a aplicação (linhas 16 e 17).

Algoritmo 1 Execução otimista da transação – cliente c_i

Variáveis:

(1) $t.rs = \perp$ /* *read-set* da transação observado pelo cliente */
(2) $t.ws = \perp$ /* *write-set* da transação observado pelo cliente */
(3) $t.op = \perp$ /* Contador de operações da transação */
(4) $t.stmts\langle\rangle = \perp$ /* Lista de operações da transação - *hashtable* */
(5) $t.leader = \perp$ /* *id* da réplica líder/primária da transação */
(6) $t.started = \perp$ /* Indicador de inicialização da transação */

procedure $execute\_statement(stmt)$
(7) **if** $type(stmt)$ = `begin` **then**
(8) **if** $t.started = \perp$ **then**
(9) $t.rs := \varnothing$
(10) $t.ws := \varnothing$
(11) $t.op := 0$
(12) $t.stmts\langle\rangle := \varnothing$
(13) $TO\text{-}multicast(c_i, \langle \texttt{BEGIN}, t\rangle_\sigma)$ to $\forall r_j \in \mathcal{R}$
(14) **wait until** $[(receive(\langle c_i, t, \texttt{ACTIVE}, leader\_id\rangle_\sigma)$ from $f+1 \neq$ replicas r_j : $\forall r_j$ received $\Rightarrow verifysig(r_j, \langle c_i, t, \texttt{ACTIVE}, leader\_id\rangle_\sigma))]$
(15) $t.leader := \{leader\_id$ from message $\langle c_i, t, \texttt{ACTIVE}, leader\_id\rangle\}$
(16) $t.started := true$
(17) **return** $t.started$
(18) **else if** $type(stmt)$ = `read` $\vee$ $type(stmt)$ = `read-write` $\vee$ $type(stmt)$ = `write` **then**
(19) **if** $t.started = true$ **then**
(20) $retries := 0$; $response := \perp$; $t.op := t.op + 1$; $t.stmts\langle t.op\rangle := stmt$
(21) **repeat**
(22) $send(c_i, \langle t, t.op, stmt\rangle_\sigma)$ to $t.leader$
(23) **wait until** $[(receive(\langle t, t.op, result\rangle_\sigma)$ from $t.leader$: $verifysig(t.leader, \langle t, t.op, result\rangle_\sigma))$ **or** $(\Delta_{t.leader}$ expires)]
(24) **if** $\Delta_{t.leader}$ **not** expired **then**
(25) $response := \{result$ **from** message $\langle t, t.op, result\rangle\}$
(26) **if** $type(stmt)$ = `read` **then**
(27) $t.rs := t.rs \cup \{response\}$
(28) **else if** $type(stmt)$ = `read-write` **then**
(29) $t.rs := t.rs \cup \{response\}$; $t.ws := t.ws \cup \{response\}$
(30) **else**
(31) $t.ws := t.ws \cup \{response\}$
(32) **return** $response$
(33) **else**
(34) $retries := retries + 1$
(35) **until** $retries$ = N
(36) **raises** $exception$
(37) **else if** $type(stmt)$ = `commit` **then**
(38) **if** $t.started = true \wedge t.op > 0$ **then**
(39) $hash\_rs := digest(t.rs)$
(40) $hash\_ws := digest(t.ws)$
(41) $TO\text{-}multicast(c_i, \langle \texttt{COMMIT}, t, hash\_rs, hash\_ws, t.stmts\langle\rangle\rangle_\sigma)$ to $\forall r_j \in \mathcal{R}$
(42) **wait until** $[(receive(\langle c_i, t, \texttt{OUTCOME}\rangle_\sigma)$ from $f+2 \neq$ replicas r_j : $\forall r_j$ received $\Rightarrow verifysig(r_j, \langle c_i, t, \texttt{OUTCOME}\rangle_\sigma))]$
(43) $t.started := \perp$
(44) **return** (`OUTCOME` = $commit$) ? $committed$: $aborted$
(45) **else if** $type(stmt)$ = `abort` **then**
(46) **if** $t.started = true \wedge t.op > 0$ **then**
(47) $TO\text{-}multicast(c_i, \langle \texttt{ABORT}, t\rangle_\sigma)$ *to* $\forall r_j \in \mathcal{R}$
(48) $t.started := \perp$
(49) **return** $aborted$

upon $receive(\langle c_i, t, \texttt{ABORT}\rangle_\sigma)$ from $t.leader$: $verifysig(t.server, \langle c_i, t, \texttt{ABORT}\rangle_\sigma)$
(50) $t.started := \perp$
(51) **raises** $exception$

Uma vez que o cliente tenha iniciado a transação – denotada no algoritmo por t, ele pode executar operações de leitura e de escrita sobre os objetos e itens de dados contidos no banco de dados. O código que trata da execução de operações por parte do cliente é especificado nas linhas 18 a 36. É importante verificar que durante esta fase do protocolo, isto é, a fase de `execução` conforme definido na Figura 34, o cliente executa as operações da transação de maneira otimista, em que apenas a réplica líder é contatada e as demais réplicas não são envolvidas durante o período da execução da transação t (linha 22 – *t.leader*). No caso, para cada operação da transação, o cliente envia a instrução à réplica líder e aguarda a resposta (linhas 22 e 23). Quando o cliente recebe uma resposta, ele obtém daquela mensagem o resultado da operação e o adiciona ao seu *read-set* ou *write-set* (ou ambos), de acordo com o tipo da operação executada (linhas 27, 29 e 31). E por fim, o cliente retorna o resultado recebido para a aplicação de nível superior (linha 32). Duas observações importantes a respeito desta fase: (*i*) o protocolo permite que o cliente envie operações apenas para transações previamente iniciadas (linha 19); (*ii*) se a réplica líder não enviar uma resposta para a operação num intervalo definido de tempo (p. ex.: longo o suficiente), um temporizador é esgotado e o cliente reenvia a operação (linhas 33 e 34 do laço). Se a operação for reenviada um número pré-determinado de vezes (definido no algoritmo pela variável N), o evento é tratado como uma falta de parada por parte da réplica líder (condição de repetição do laço – linha 35). Neste caso, é lançada uma exceção e a transação é considerada como anulada, sendo que o cliente deverá reiniciar a transação.

A fase de `execução` da transação t é concluída quando não há mais operações a serem executadas e o cliente solicita a validação da transação (`COMMIT` – linha 37), ou a anulação desta (`ABORT` – linha 45), por exemplo, se ele verifica que o resultado de alguma operação não estava de acordo com o esperado. Tanto o pedido de validação como o de anulação – apenas um deles – é enviado por meio do protocolo subjacente de difusão com ordem total, para assegurar que todas as réplicas entregarão a mensagem, e consequentemente terminarão a transação na mesma ordem, tendo em vista a manutenção da consistência do estado das réplicas (i.e., do ambiente replicado de banco de dados). No caso do cliente ter solicitado a validação da transação, após o envio da mensagem `COMMIT` às réplicas (linha 41), ele aguarda o recebimento de $f + 2$ notificações/respostas mutuamente consistentes de diferentes réplicas para o pedido de validação, para dar por completa a transação e determinar seu respectivo resultado/estado final (linha 42). Como

não mais de f réplicas são faltosas, isto evita que o cliente aceite o resultado de uma réplica faltosa, que por exemplo, valida uma transação não-serializável. Por fim, o cliente atribui ao indicador de inicialização um valor nulo para permitir que novas transações sejam iniciadas (linha 43) e retorna o resultado para a aplicação (linha 44). Note que, no caso de um pedido de anulação espontânea por parte do cliente (linhas 45 a 49), o mesmo não aguarda pela confirmação das réplicas, o que decorre do fato de que a anulação não altera o estado do banco de dados. Maiores detalhes sobre o protocolo de validação são fornecidos na Seção 4.5.3.3.

É importante notar que a despeito das usuais $f + 1$ respostas iguais requeridas pelos protocolos de replicação para BFT (CASTRO; LISKOV, 1999; KOTLA et al., 2007; VANDIVER et al., 2007; GARCIA; RODRIGUES; PREGUIÇA, 2011; PEDONE; SCHIPER, 2012), o protocolo proposto requer $f + 2$ respostas iguais de diferentes réplicas. Esta necessidade advém de um possível cenário verificado na execução do protocolo, no qual uma réplica bizantina maliciosa pode tentar causar um equívoco no cliente quanto ao resultado correto da transação, situação que pode ocorrer no intervalo entre o envio da mensagem `COMMIT` e a entrega desta às réplicas. Suponha um cenário onde duas transações T_i e T_j são concorrentes e potencialmente conflitantes (p. ex.: ambas operam sobre os mesmos objetos/itens de dados), sendo que T_i está sendo executada de maneira otimista pela réplica líder r_1, e que para T_j a réplica r_2 foi escolhida como líder. Portanto, embora elas sejam potencialmente conflitantes, isso não afetará a execução das mesmas, já que estão sendo processadas por réplicas primárias distintas e o(s) conflito(s) será(ão) detectado(s) somente na terminação destas. Suponha também, que os clientes responsáveis pelas transações enviam os pedidos de validação para suas respectivas transações num mesmo instante – note que não há sincronização entre eles. Como a mensagem `COMMIT` é enviada via difusão com ordem total, elas serão entregues uma após a outra. Considere que T_j é entregue antes de T_i, e que $T_j \prec T_i$ (i.e., $commit(T_j) \prec commit(T_i)$).

No protocolo de terminação proposto, a validação da transação está condicionada à sua aprovação num teste de certificação (maiores detalhes na Seção 4.5.3.3), e deste modo, ao certificar T_j na réplica r_1 o conflito será detectado e T_i terá de ser anulada de forma preemptiva (i.e., não espontaneamente) para liberar os recursos (p. ex.: os bloqueios sobre os itens de dados em conflito) e permitir que T_j complete sua fase de `terminação`. Nesta situação, antes mesmo de entrega da mensagem `COMMIT` já enviada para T_i, o cliente receberá uma notifica-

ção de anulação não espontânea da réplica r_1 (líder de T_i) que não é faltosa. Então, as réplicas faltosas podem se aproveitar da situação e enviar a notificação de anulação espúria para o cliente – sem ter entregue a mensagem `COMMIT` para T_i. Nesta situação, como o cliente recebeu 1 notificação da réplica líder que era correta, se vier a receber mais f notificações poderá dar por concluída a transação, o que não é o caso. Por esta razão é que há a necessidade de $f + 2$ respostas iguais para a conclusão da fase de `terminação`. Pois, por especificação/construção até f podem falhar; e numa situação de conflito, se 1 (uma) réplica primária correta porventura tiver que liberar de maneira compulsória os bloqueios em sua posse, isto implicará na anulação da transação. Pelo algoritmo especificado para o protocolo de validação, as réplicas corretas não líderes para aquela transação somente notificarão o cliente após a entrega da mensagem `COMMIT`, e a réplica líder notificará o cliente mais uma vez ao concluir a validação da transação pela entrega da mensagem `COMMIT`. Isso garante que o cliente receberá $f+2$ mensagens iguais e mutuamente consistentes de diferentes réplicas, pois $n - f \geq f + 2$. O que, portanto, impede que ações arbitrárias de réplicas faltosas desvirtuem a corretude do protocolo.

O cenário descrito pelo parágrafo anterior culmina na condução de uma transação t_i, para a qual foi identificado um conflito durante sua fase de `execução`, para o estado intermediário `aborted`. Tal situação decorre da anulação por preempção de t_i, realizada pela sua réplica líder, para permitir a progressão de uma transação T_j em conflito, cuja fase de `terminação` estava em processamento naquela réplica. Uma vez que a transação estiver no estado intermediário `aborted`, os estados alcançáveis são o estado intermediário `validating` e o estado final `aborted`. Note que, conforme mencionado, a réplica líder de t_i ao anular por preempção tal transação, notifica o cliente quanto ao ocorrido. Deste modo, em circunstâncias normais, o cliente, ao ser notificado, envia por difusão com ordem total uma mensagem `ABORT` para todas as réplicas. Tal situação conduzirá a transação t_i para o estado final `aborted` nas réplicas, quando ocorrer a entrega da mensagem `ABORT` pelo protocolo. De outro modo, se o cliente tiver enviado a mensagem `COMMIT` para a transação t_i, mas esta ainda não tiver sido entregue às réplicas, o cliente deverá aguardar por $f + 2$ respostas de diferentes réplicas. Neste caso, se a réplica líder anular a transação t_i por preempção, ao entregar a mensagem `COMMIT` para a mesma – que fora enviada antes da anulação desta – conduzirá t_i para o estado `validating`, e o mesmo será efetuado pelas réplicas não líderes, pois o conflito não foi verificado em vista da execução da transação apenas na réplica líder.

4.5.3.2 Processamento da Transação

Nesta seção é apresentada a especificação formal algorítmica, que corresponde às fases de início, execução e terminação por anulação do protocolo, as quais são executadas pelas réplicas do banco de dados. É digno de nota que o código especificado no Algoritmo 2 é implementado por uma entidade denominada gerenciador de réplicas (*replica manager*), sendo esta executada sobre as instâncias do SGBDs locais, que mantém cada uma das réplicas do banco de dados. Note que os gerenciadores de réplica definem algumas variáveis globais – para todas as transações (linhas 1 a 4), e locais – no contexto de cada transação (linhas 5 a 9), a fim de preservar os dados de controle das transações e também auxiliá-los nas tarefas que dizem respeito ao processamento das transações, controle de concorrência, etc.

A fim de facilitar a explicação acerca da especificação do Algoritmo 2, considere um cliente c_i que inicia a transação t_i, para a qual a réplica r_j é escolhida como primária/líder. No lado das réplicas, uma transação é iniciada quando elas entregam a mensagem `BEGIN` enviada pelo cliente c_i via difusão com ordem total. Isto implica que todas as réplicas darão início à transação na mesma ordem – o que é imperioso para nortear o processo de certificação da transação quando de sua validação e um requisito para prover a corretude do banco de dados. Ao entregar a mensagem `BEGIN`, as réplicas primeiramente verificam se aquele cliente já tem alguma transação ativa no banco de dados (i.e., no estado `active`), ou se aquele é um pedido de reinício para uma transação anulada não espontaneamente (condições da linha 10). Se uma das condições for atendida, as réplicas seguem adiante com o início da transação, do contrário, a transação é ignorada. Se as réplicas aceitam o pedido de início para a transação t_i, elas inicializam as variáveis de controle para t_i (linhas 11 a 18) e apenas a líder, que foi definida na linha 14, inicia a transação no banco de dados local (linhas 19 e 20) – as demais apenas o farão quando da terminação da transação, para evitar a alocação de recursos no banco de dados. Por fim, todas as réplicas enviam uma notificação quanto ao aceite e início da transação t_i ao cliente c_i (linha 21). Note que o critério adotado para a escolha da réplica líder/primária para cada transação, segue o princípio do algoritmo de *round-robin* (i.e., circular) (TANENBAUM, 2007). Deste modo, o critério também visa a equidade quanto ao número de transações em execução em cada uma das réplicas.

Ao adentrar para a fase de `execução` da transação, a réplica primária passa a permitir/aceitar a execução dos comandos que compõem

Algoritmo 2 Processamento da transação – réplica r_j

Variáveis globais da réplica:

(1) $\mathcal{H} = \perp$ /* Histórico de transações executadas e validadas (escalonamento) */
(2) $\mathcal{A}[\,] = \perp$ /* Estrutura de dados que contêm informações sobre as transações iniciadas pelos clientes */
(3) $T_{id} = 0$ /* Identificador global da transação - um contador */
(4) $written\_objects\langle\rangle = \perp$ /* Lista de objetos escritos/atualizados pelas transações */

Variáveis mantidas para cada transação:

(5) $leader = \perp$ /* Réplica líder/primária da transação T_{id} */
(6) $\texttt{WS} = \perp$ /* *write-set* da transação T_{id} */
(7) $read\_objects\_tx = \perp$ /* Lista dos objetos lidos pela transação T_{id} */
(8) $write\_objects\_tx = \perp$ /* Lista dos objetos escritos/atualizados pela transação T_{id} */
(9) $stmts\langle\rangle = \perp$ /* Lista de operações da transação - uma *hashtable* */

upon $TO\text{-}deliver(c_i, \langle \texttt{BEGIN}, t\rangle_\sigma)$
(10) **if** $\mathcal{A}[c_i] = \perp \vee \mathcal{A}[c_i]\langle t, * \rangle = t$ **then**
(11) $\quad T_{id} := T_{id} + 1$
(12) $\quad \mathcal{A}[c_i] := \langle t, T_{id}\rangle$
(13) $\quad state(t) := \texttt{active}$
(14) $\quad leader := T_{id}$ **mod** $|\mathcal{R}|$
(15) $\quad write\_objects\_tx := \emptyset$
(16) $\quad stmts\langle\rangle := \emptyset$
(17) $\quad \texttt{WS} := \emptyset$
(18) $\quad handle\_tx\langle T_{id}\rangle := \langle leader, write\_objects\_tx, \texttt{WS}, stmts\langle\rangle\rangle$
(19) $\quad$ **if** $leader = myself$ **then**
(20) $\quad\quad begin\_db\_tx(\mathcal{D}, T_{id})$
(21) $\quad send(r_j, \langle c_i, t, \texttt{ACTIVE}, leader\_id\rangle_\sigma)$ *to* c_i

upon $receive(\langle t, t.op, stmt\rangle_\sigma)$ from $c_i : verifysig(c_i, \langle t, t.op, stmt\rangle_\sigma)$
(22) **if** $\mathcal{A}[c_i]\langle t, * \rangle = t \ \wedge \ state(t) = \texttt{active}$ **then**
(23) $\quad T_{id} := get\_global\_id(\mathcal{A}[c_i])$
(24) $\quad \langle leader, write\_objects\_tx, \texttt{WS}, stmts\langle\rangle\rangle := handle\_tx\langle T_{id}\rangle$
(25) $\quad$ **if** $leader = myself \ \wedge \ t.op = (last\_key(stmts\langle\rangle) + 1)$ **then**
(26) $\quad\quad stmts\langle t.op\rangle := stmt$
(27) $\quad\quad result := \perp$
(28) $\quad\quad$ **if** $type(stmt) = \texttt{read}$ **then**
(29) $\quad\quad\quad result := execute\_db\_read(\mathcal{D}, \langle T_{id}, stmt\rangle)$
(30) $\quad\quad$ **else if** $type(stmt) = \texttt{write} \vee type(stmt) = \texttt{read-write}$ **then**
(31) $\quad\quad\quad write\_objects\_tx := write\_objects\_tx \cup \{db\_objects\_stmt(stmt)\}$
(32) $\quad\quad\quad result := execute\_db\_write(\mathcal{D}, \langle T_{id}, stmt\rangle)$
(33) $\quad\quad\quad \texttt{WS} := \texttt{WS} \cup \langle T_{id}, result\rangle$
(34) $\quad\quad$ **if** $result \neq \perp$ **then**
(35) $\quad\quad\quad send(r_j, \langle t, t.op, result\rangle_\sigma)$ *to* c_i
(36) $\quad$ **else if** $leader = myself \ \wedge \ t.op = last\_key(stmts\langle\rangle)$ **then**
(37) $\quad\quad send(r_j, \langle t, t.op, result\rangle_\sigma)$ *to* c_i

upon $TO\text{-}deliver(c_i, \langle \texttt{ABORT}, c_i, t\rangle)$
(38) **if** $\mathcal{A}[c_i]\langle t, * \rangle = t \ \wedge \ (state(t) = \texttt{active} \vee state(t) = \texttt{aborted})$ **then**
(39) $\quad T_{id} := get\_global\_id(\mathcal{A}[c_i])$
(40) $\quad release\_resources\_tx(T_{id})$
(41) $\quad$ **if** $leader = myself$ **then**
(42) $\quad\quad abort\_db\_tx(\mathcal{D}, T_{id})$
(43) $\quad state(t) := \texttt{aborted}$
(44) $\quad send(r_j, \langle c_i, t, \texttt{ABORT}\rangle_\sigma)$ *to* c_i

a transação t_i enviados pelo cliente c_i. O código que especifica os detalhes acerca do processamento da transação, a partir das instruções enviadas pelo cliente é descrito entre as linhas 22 a 37. Este código é executado por uma tarefa que é criada para cada transação em atividade naquela réplica líder (p. ex.: uma *thread*). Ao receber uma operação de c_i para a transação t_i, a tarefa executada pela réplica líder verifica se t_i foi iniciada para o cliente c_i e também se ela está no estado `active` (condição da linha 22) – o único estado que permite a execução de operações na transação (cfm. Seção 4.5.2). Se ambas as condições forem atendidas, a réplica r_j procede com a execução da operação recebida; do contrário, a operação é simplesmente descartada. Ao proceder com a execução da operação, a réplica líder obtém as informações de controle da transação para a qual está sendo solicitada a execução da operação (i.e., t) – estas informações estão armazenadas na estrutura de dados *handle_tx* (linhas 18 e 24). Então, r_i verifica se ela é a líder daquela transação e também se a operação recebida é a que deve ser executada com respeito à ordem da operação (i.e., se o identificador da operação em questão é uma unidade maior que o da última operação executada). Após as verificações, a réplica líder executa a operação de acordo com o tipo desta (p. ex.: leitura ou escrita – linhas 28 e 30) e envia o resultado recebido do banco de dados ao cliente (linhas 34 e 35). Note que se for uma operação de escrita, a réplica verifica sobre qual objeto do banco de dados está sendo realizada a operação (i.e., tabela(s) – a função *db_objects_stmt* retorna o(s) nome(s) do(a)(s) objeto(s)/tabela(s) referenciado(a)(s) na operação passada como argumento – SQL), e também guarda o resultado junto ao *write-set* (`WS`) daquela transação. Tanto o nome do(s) objeto(s) como o *write-set* são utilizados posteriormente para identificar possíveis conflitos com outras transações, no momento da validação de t_i.

É importante salientar que o procedimento descrito no parágrafo anterior se repete enquanto houver operações a serem executadas para a transação t_i. Neste caso, a tarefa que implementa a execução otimista das operações (linhas 22 a 37), é mantida em atividade até a entrega do pedido de validação para a transação t_i. O código entre as linhas 38 a 44 consiste na especificação do procedimento de anulação espontânea para a transação, isto é, aquele que é manifestado pelo cliente. No caso, se ao término da execução da transação t_i o cliente c_i verifica que alguma coisa não saiu conforme o esperado no âmbito de t_i, ele solicita a anulação desta às réplicas por meio do envio da mensagem `ABORT` por difusão com ordem total. Ao entregar esta mensagem, as réplicas simplesmente liberam os recursos alocados pela transação (p.

ex.: variáveis de controle, etc.), e apenas a réplica líder descarta os resultados intermediários do banco de dados – já que até então apenas ela havia iniciado a transação no banco de dados (cfm. linhas 19 e 20), no que segue elas definem o estado da transação t_i como `aborted` e, por fim, notificam o cliente. A anulação é um procedimento bastante simples porque o mesmo não envolve a alteração do estado do banco de dados, mas apenas a reversão das operações executadas no mesmo (i.e., *rollback*). Por outro lado, a validação da transação envolve um procedimento mais complexo, que será abordado em detalhes na próxima seção.

4.5.3.3 Terminação, Certificação e Validação da Transação

A parte do protocolo que trata da terminação da transação é formalizada a partir dos Algoritmos 3 e 4. Para tanto, o Algoritmo 3 especifica a tarefa do protocolo de terminação, a qual é iniciada pela entrega da mensagem `COMMIT` para a transação. Já o Algoritmo 4, consiste na especificação dos testes de certificação e de validação, ambos referenciado pelo protocolo de terminação, e portanto, instanciados pelo Algoritmo 3. Note que esta é uma das parte mais importantes do protocolo de replicação, pois é ela que visa assegurar a corretude da transação e a consistência do banco de dados em consonância com o critério 1-SR (*one-copy serializability*), a despeito da ocorrência e de faltas bizantinas. A especificação destes algoritmos está pautada nos seguintes aspectos: (*i*) em conformidade com a semântica transacional, o efeito de uma transação só se torna permanente e durável a partir da sua validação (GRAY; REUTER, 1992); (*ii*) em um ambiente de replicação por máquina de estados, cada operação processada sobre a máquina de estados conduz o sistema a um novo estado (SCHNEIDER, 1990); (*iii*) mensagens enviadas por difusão com ordem total são entregues na mesma ordem por todos os processos corretos (DÉFAGO; SCHIPER; URBÁN, 2004). Isto posto, a especificação do algoritmo executado na fase de `terminação` é baseado numa máquina de estados, onde o pedido de validação (i.e., a mensagem `COMMIT`) é enviado às réplicas por meio de difusão com ordem total. Com efeito, se todas as réplicas iniciam no mesmo estado e a mensagem de validação da transação é entregue na mesma ordem por todas as réplicas, o efeito produzido pela transação levará todas as réplicas para um mesmo novo estado – já que a validação será executada por elas na mesma ordem.

A especificação do protocolo de terminação é norteada pelo prin-

cípio da certificação de Kung e Robinson (1981), no sentido de verificar se a transação pode ser serializada de acordo com a ordem de entrega da respectiva mensagem `COMMIT`. Como as diretrizes do princípio de Kung e Robinson não consideram a sujeição a faltas de nenhuma natureza, embora o protocolo de terminação proposto se baseie nestas, ele emprega diversas modificações e especificidades para acomodar a sujeição a faltas bizantinas tanto no processamento, como na validação da transação. Em suma, para que a transação possa ser considerada válida e seu efeito tornar-se durável no banco de dados no âmbito do protocolo proposto, a fase de `terminação` executa dois testes, dos quais um é para certificar a transação – averiguar a atualidade dos itens de dados em condições de conflito(s) com transações concorrentes, e o outro é para verificar a validade dos resultados obtidos durante a fase de `execução` – aqueles que foram observados pelo cliente. Os resultados destes dois testes determinam se a transação está em conformidade com a semântica serializável e pode ser serializada no banco de dados, após a última transação válida do histórico $\mathcal{H}$.

Conforme ilustrado pela Figura 34 e explanado nas Seções 4.4 e 4.5.3.2, a execução da transação t_i se dá por concluída quando o cliente c_i envia o pedido de validação para ela. Este pedido é realizado por meio do envio da mensagem `COMMIT` via difusão com ordem total a todas as réplicas. Junto a esta mensagem o cliente envia dois resumos criptográficos (i.e., *hash*) calculados a partir dos resultados recebidos para as operações de leitura ($hash\_rs$) e de escrita ($hash\_ws$), durante a fase de `execução` da transação; e também a lista que contém todas as operações executadas pela transação ($t.stmts\langle\rangle$). Note que o cliente c_i assina a mensagem `COMMIT` (denotado nos algoritmos por σ) para evitar que outra entidade que não ele, forje um pedido de validação para uma transação espúria – por exemplo, uma réplica bizantina. Ao entregar a mensagem `COMMIT`, as réplicas primeiramente verificam se a transação para a qual a validação está sendo solicitada (i.e., t) consta como uma transação ativa para o cliente c_i, e se o estado de t é `active` ou `aborted` (condição da linha 1, Algoritmo 3). Se a transação está no estado `active`, significa que ela seguiu o fluxo normal de execução e pode entrar na fase de `terminação`. De outro modo, se a transação esta no estado intermediário `aborted` (cfm. Figura 35, único estado `aborted` possível em que a transação pode adentrar em sua fase de `terminação`), é decorrente da seguinte situação:

* o cliente c_i solicita a validação para sua transação t_i, mas antes da entrega da mensagem `COMMIT` e início de sua fase de `terminação`, uma transação concorrente e conflitante t_j de um cliente c_j, entra

em sua fase de `terminação`. Neste caso, quando t_j entra em sua fase de `terminação`, ela obtém prioridade na aquisição/concessão de bloqueios sobre os itens de dados em todas as réplicas (linha 21 do Algoritmo 4). Como t_i e t_j estão em conflito (i.e., elas operam sobre algum item de dados em comum), t_i que está em execução na réplica líder r_k já detém a concessão do(s) bloqueio(s) sobre o(s) item(ns) de dados que t_j necessita operar/certificar em r_k. Razão pela qual t_i tem de ser anulada por preempção (i.e., não espontaneamente) por sua réplica líder r_k, para que t_j possa operar sobre o(s) item(ns) de dados em conflito e assim continuar com sua validação – transações em fase de `terminação` tem prioridade sobre transações em fase de `execução`. Tal situação culmina na condução de t_i para o estado intermediário `aborted`; o que não a impede de entrar em sua fase de `terminação`, já que o conflito não é detectado nas réplicas não líderes. Note que, se porventura a certificação da transação t_j vier a falhar, haverá probabilidade de sucesso na validação e terminação de t_i, cuja mensagem `COMMIT` será entregue posteriormente.

No caso da transação estar no estado final `aborted`, a condição da linha 1 do Algoritmo 3 será avaliada como falsa. Pois, como o cliente é notificado quanto à anulação de sua transação, ele enviará uma mensagem `ABORT` à todas as réplicas, caso não tenha solicitado a validação da mesma. As réplicas por sua vez, ao entregarem a mensagem `ABORT` apagarão todas as variáveis de controle mantidas para aquela transação, de modo que $\mathcal{A}[c_i] = \perp$. Por esta razão, apenas uma transação que está no estado `aborted` intermediário é aceita para entrar na fase de `terminação`. Isto posto, ao adentrar em sua fase de `terminação`, isto é, se a condição da linha 1 for avaliada como verdadeira, as réplicas põem a transação t_i no estado `validating` (cfm. ilustrado pela Figura 35) e recuperam as variáveis de controle para aquela transação (linhas 3 a 6 – Algoritmo 3). Nos próximos passos, cada réplica verifica a condição da transação para dar início à transação no banco de dados. No caso, cada réplica verifica se ela não é a líder daquela transação e se o estado anterior era `active`, ou caso seja a líder, se o estado anterior era `aborted`. Portanto, a única maneira de uma transação que está no estado intermediário `aborted` iniciar sua fase de `terminação`, é através de sua réplica líder. Ambas as condições da linha 7 (Algoritmo 3) são imperiosas para as ações executadas nas linhas 8 e 9 (Algoritmo 3), pois as réplicas não líderes não iniciam a transação no banco de dados quando ela é criada (linhas 19 e 20, Algoritmo 2), e no caso da líder

que a anulou por preempção (i.e., não espontaneamente) em decorrência da situação já mencionada, a transação deve ser novamente iniciada no banco de dados. A variável da linha 8 indica que é requerido o teste de certificação para aquela transação.

Algoritmo 3 Terminação da transação – réplica r_j

```
upon TO-deliver(c_i, ⟨COMMIT, t, hash_rs, hash_ws, t.stmts⟨⟩⟩_σ)
(1)  if A[c_i]⟨t, *⟩ = t ∧ (state(t) = active ∨ state(t) = aborted) then
(2)      previous_state := state(t)
(3)      state(t) := validating
(4)      T_id := get_global_id(A[c_i])
(5)      ⟨leader, write_objects_tx, WS, stmts⟨⟩⟩ := handle_tx⟨T_id⟩
(6)      final_outcome := ⊥
(7)      if (leader ≠ myself ∧ previous_state = active) ∨
           (leader = myself ∧ previous_state = aborted) then
(8)        certify := true
(9)        begin_db_tx(D, T_id)
(10)     else if leader = myself ∧ previous_state = active then
(11)       certify := false
(12)       validate := false
(13)       final_outcome := passed
(14)     if certify = true then
(15)       final_outcome := certify_and_validate(T_id, hash_rs, hash_ws, t.stmts⟨⟩)
(16) if final_outcome = passed then
(17)     state(t) := committed
(18)     written_objects⟨T_id⟩ := write_objects_tx
(19)     append_on_tail(H, T_id)
(20)     commit_db_tx(D, T_id)
(21)     OUTCOME := committed
(22) else
(23)     state(t) := aborted
(24)     abort_db_tx(D, T_id)
(25)     OUTCOME := aborted
(26) send(r_j, ⟨c_i, t, OUTCOME⟩_σ) to c_i
(27) release_resources_tx(T_id)
```

De outro modo, a condição descrita na linha 10 do Algoritmo 3 só é passível de verificação pela réplica líder daquela transação em questão, isto é, se o estado anterior era `active` – o que indica que a transação seguiu todo o fluxo de execução definido pelo protocolo e não sofreu uma anulação não-espontânea por preempção. Pela especificação do protocolo, se uma transação t_i segue o fluxo normal de execução e consegue alcançar com êxito sua fase de `terminação`, significa que no decurso da transação, isto é, desde a fase de `execução` até o início da fase de `terminação`, não foi verificado nenhum conflito de t_i com outras transações concorrentes t_j (tal que $i \neq j$). Por conseguinte, t_i não precisa ser certificada, tampouco validada em sua réplica líder. Isto ocorre porque t_i já detém os bloqueios sobre os itens de dados afetados pela transação em sua réplica líder, e seus resultados intermediários não foram vistos por nenhuma outra transação em execução na mesma réplica. E já que localmente as réplicas executam as tran-

sações em conformidade com as semânticas do protocolo 2PL rigoroso e do isolamento serializável (vide Seção 4.2.2), se a réplica líder não é faltosa estas asserções indicam que a transação já atende aos critérios requeridos pela serialização, e consequentemente pode ser validada naquela réplica (indicador $final\_outcome$ – linha 13, Algoritmo 3). É lícito ressaltar que esta condição só é verdadeira para a réplica líder daquela transação, já que as demais não haviam iniciado a transação em seus bancos de dados locais (linha 9). Deste modo, todas as demais réplicas – exceto a líder se a condição da linha 10 for verdadeira, devem submeter a transação t_i aos testes de certificação e de validação para determinar se ela atende aos critérios requeridos pela serialização e pode ser serializada localmente em seus respectivos bancos de dados – o que é realizado pela `procedure` referenciada/executada na linha 15 do Algoritmo 3.

A `procedure` referenciada na linha 15 do Algoritmo 3 é especificada pelo Algoritmo 4. No código do Algoritmo 4, as linhas 6 a 18 correspondem à especificação do teste de certificação, enquanto que o teste de validação compreende ao código entre as linhas 19 a 33 (Algoritmo 4). Se nos passos anteriores da fase de `terminação` da transação, pelas verificações preliminares, o algoritmo definiu que é necessária a certificação desta (linha 8 – Algoritmo 3), a condição da linha 14 (Algoritmo 3) é atendida e o protocolo realiza o teste de certificação para a transação. Em suma, o objetivo do teste de certificação é assegurar que uma transação t_i só pode ser validada se cada item de dados lido durante a fase de `execução`, e portanto, observado pelo cliente, é atualizado (p. ex.: se não tornou-se obsoleto em decorrência de atualizações por transações t_j concorrentes). Note que, se a certificação fosse desprezada, a transação t_i em fase de `terminação` poderia ser alvo do fenômeno da *leitura suja* (cfm. Seção 2.1.4.1), já que os dados atualizados pelas transações concorrentes t_j não eram válidos quando do início da transação t_i. E por conseguinte, a transação t_i não seria uma transação serializável.

Ao adentrar no procedimento de certificação e validação especificado pelo Algoritmo 4, num momento duas variáveis são declaradas, a que irá conter o resultado dos testes de certificação e validação, e a outra que define como obrigatória a validação daquela transação (linhas 1 e 2 do Algoritmo 4). As demais variáveis declaradas serão alimentadas no decorrer das etapas especificadas neste procedimento, a partir dos dados contidos nos *write-set*'s das transações concorrentes a t_i. Uma transação t_j é considerada concorrente em relação à t_i, se ela não precede o início de t_i, isto é, se t_j foi validada após o `início` e antes da

Algoritmo 4 Certificação e validação da transação – réplica r_j

```
procedure certify_and_validate(T_id, hash_rs, hash_ws, t.stmts⟨⟩)
(1)  outcome := ⊥
(2)  validate := true
(3)  written_obj_list := ∅                    /* lista de objetos atualizados por transações concorrentes */
(4)  NWS := ∅                                 /* itens de dados criados por transações concorrentes */
(5)  UWS := ∅                                 /* itens de dados atualizados por transações concorrentes */
                                              /* etapa de certificação da transação */
(6)  for all T_j ∈ C(H) such that T_j ↦ T_id do
(7)      written_obj_list := written_obj_list ∪ {written_objects⟨T_j⟩}
(8)      WS := WS ∪ WS(T_j)
(9)  NWS := {∀x ∈ WS : x is a new data item}
(10) UWS := {∀y ∈ WS : y is an updated data item}
(11) read_objects_tx := {∀stmt ∈ t.stmts⟨⟩ : type(stmt) = read ∨ read-write;
                         db_objects_stmt(stmt)}
(12) write_objects_tx := {∀stmt ∈ t.stmts⟨⟩ : type(stmt) = write ∨ read-write;
                          db_objects_stmt(stmt)}
(13) if (written_obj_list ∩ read_objects_tx) ≠ ∅ then
(14)     for each stmt ∈ (t.stmts⟨⟩ : type(stmt) = read ∨ read-write) do
(15)         result := evaluate_predicate((UWS ∖ NWS), stmt_predicate(stmt))
(16)         if result = true then
(17)             validate := false
(18)             break
                                              /* etapa de validação da transação */
(19) if validate = true then
(20)     tmp_rs := ∅; tmp_ws := ∅
(21)     priority_on_locks⟨T_id⟩ := true
(22)     for each stmt ∈ t.stmts⟨⟩ do
(23)         if type(stmt) = read then
(24)             result := execute_db_read((D ∖ NWS), stmt)
(25)             tmp_rs := tmp_rs ∪ {result}
(26)         else if type(stmt) = read-write then
(27)             result := execute_db_write((D ∖ NWS), stmt)
(28)             tmp_rs := tmp_rs ∪ {result}; tmp_ws := tmp_ws ∪ {result}
(29)         else if type(stmt) = write then
(30)             result := execute_db_write(D, stmt)
(31)             tmp_ws := tmp_ws ∪ {result}
(32)     if digest(tmp_rs) = hash_rs ∧ digest(tmp_ws) = hash_ws then
(33)         outcome := passed
(34) return outcome
```

terminação de t_i – i.e., durante o período que compreende toda a execução de t_i. No âmbito do protocolo proposto, a relação de precedência é denotada pelo símbolo $\mapsto$, e sua negação pelo símbolo $\nmapsto$. No caso, a condição $T_i \mapsto T_j$ denota que a transação T_i precede a transação T_j, o que indica que a operação COMMIT de T_i ocorreu antes da operação BEGIN de T_j. Para tanto, a verificação acerca das transações concorrentes t_j em relação à transação t_i é realizada por meio do mapeamento da relação de precedência entre elas, o que é realizado na linha 6 do Algoritmo 4.

Na condição da linha 6 (Algoritmo 4) são recuperadas do histórico de transações $\mathcal{H}$, todas as transações que não são precedentes à transação em processo de certificação – i.e., todas as que são considera-

das concorrentes em relação à T_{id}. Para cada transação concorrente é recuperada a lista de objetos (p. ex.: tabelas) escritos/atualizados pela mesma (linha 7 – Algoritmo 4), e também os *write-set*'s que contém os identificadores dos itens de dados escritos/atualizados (linha 7, Algoritmo 4) – ambas as informações são essenciais para os passos seguintes da etapa de certificação. No passo seguinte, as informações contidas no *write-set* (o conjunto `WS`) são divididas em dois conjuntos: (*i*) o subconjunto que contém os novos itens de dados escritos pelas transações concorrentes são colocados num conjunto `NWS`, e; (*ii*) os itens de dados que já existiam no banco de dados e que apenas foram atualizados pelas transações concorrentes, são colocados num conjunto `UWS`. Na sequência são recuperados, a partir das operações executadas pela transação em certificação (i.e., T_{id}), todos os objetos lidos e escritos, de acordo com o tipo de operação executada sobre este, por exemplo, `SELECT` (`read`), `INSERT` (`write`), `UPDATE` (`read-write`) e `DELETE` (`read-write`).

O código entre as linhas 13 a 18 do Algoritmo 4 consiste no teste de certificação, que é executado a partir das informações obtidas nos passos anteriores. O primeiro aspecto a ser avaliado para a aprovação da transação T_{id} na certificação, é a verificação quanto à existência de conflitos de leitura/escrita com as transações concorrentes a T_{id}. Num primeiro momento, a existência de conflitos é verificada a partir da lista que contém os objetos escritos pelas transações concorrentes ($written\_obj\_list$) e da lista que contém os objetos lidos pela transação a ser certificada T_{id} ($read\_objects\_tx$). Esta verificação é realizada a partir da interseção destas duas listas, de modo que se o resultado for um conjunto vazio, indica que nenhuma operação de escrita foi realizada sobre os objetos lidos por T_{id}, e portanto, não pode ter havido nenhum conflito. Neste caso, a transação pode seguir adiante para o teste de validação (linha 19 – Algoritmo 4). De outro modo, se houve a indicação de conflito na condição da linha 13, os predicados fornecidos junto às operações de leitura e de leitura-escrita na fase de `execução`, são avaliados sobre a parte do *write-set* que contém os itens de dados atualizados pelas transações concorrentes (do conjunto `UWS`). O propósito desta avaliação é verificar se houve acesso concomitante aos mesmos itens de dados lidos pela transação T_{id}, que culminaram na alteração destes valores. No caso de uma alteração de valores, não mais será possível assegurar a consistência de leitura, e portanto, T_{id} deverá ser anulada. E se assim não fosse, T_{id} estaria sujeita à uma **leitura-suja** (vide Seção 2.1.4.1) – o que não é permitido numa execução serializável.

É importante verificar que há a possibilidade de que uma tran-

sação concorrente T_j tenha incluído um item de dados que combine com o predicado fornecido pela operação, e que a transação T_k também concorrente a T_{id} o tenha atualizado, situação em que o item de dados será incluído no conjunto `UWS`. E é evidente que se o item de dados combina com o predicado de uma operação de leitura, ele será retornado na execução da operação, na fase de validação. Tal situação não é correta em uma execução serializável e violaria o **isolamento** quanto à execução da transação T_{id} – que não pode ter lido um item de dados escrito e posteriormente atualizado por transações que não a precedem. E para evitar esta situação, os predicados são avaliados sobre o conjunto resultante da operação `UWS` $\setminus$ `NWS`, isto é, somente sobre os itens de dados existentes antes do início da transação T_{id}. Este cenário é ilustrado exatamente pela Figura 36. No caso, se o predicado de alguma operação for avaliado como verdadeiro (i.e., *true* – linha 29), é porque algum item de dados foi alterado desde o início da transação T_{id}, e deste modo, não mais será possível assegurar a consistência das operações de leitura. Neste caso, a transação T_{id} não será aprovada na certificação e será conduzida diretamente para anulação (linhas 30 e 46). Para a certificação, os itens de dados que foram criados por transações concorrentes à T_{id} não são considerados (`NWS`), já que aquelas transações não precedem T_{id}. Isto se deve ao fato de que, como eles foram criados por alguma transação não validada que estava em execução antes do início de T_{id}, para respeitar a propriedade de isolamento (serializável), eles não poderiam ser lidos (e o não foram) por T_{id} durante sua fase de execução – isto seria caracterizado como uma **leitura-suja** –, tendo em vista que tais itens de dados foram validados após o início de T_{id} (i.e., passaram a integrar o estado do banco de dados durante a execução de T_{id}).

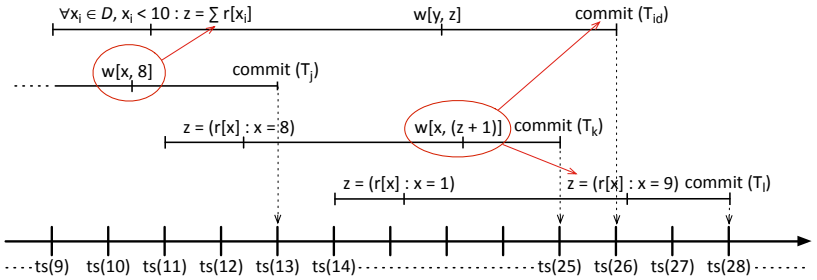


Figura 36 – Cenário de conflito na execução de transações.

Uma vez que a transação T_{id} é aprovada no teste de certificação, a próxima etapa consiste em submetê-la ao teste de validação (linhas 19 a 33 do Algoritmo 4). Como sugere o nome, este teste tem por finalidade verificar a validade das operações executadas de maneira otimista pela réplica líder. É importante salientar que pelo fato de a réplica líder estar sujeita a faltas bizantinas, ela pode ter manifestado o comportamento faltoso durante o processamento das operações, por exemplo, retornando resultados incorretos para estas, e portanto, diferentes daqueles esperados pelo cliente; ou omitindo a execução de alguma operação ou do predicado fornecido; entre outros. Note que o comportamento faltoso pode ter origem de uma falta bizantina benigna, por exemplo, corrução de dados em disco ou memória; ou de uma falta bizantina intencional (maliciosa), ou seja, aquela em que a réplica líder propositadamente desvia seu comportamento da especificação normal.

Ao iniciar o teste de validação, as réplicas definem duas variáveis auxiliares, tmp_rs e tmp_ws (linha 20, Algoritmo 4), as quais serão utilizadas para armazenar os resultados intermediários obtidos a partir da execução das operações da transação. Em seguida o protocolo informa ao gerenciador de transações – entidade que controla todas as transações em execução pelo protocolo nas réplicas, que a transação T_{id} passa a ter prioridade na aquisição de bloqueios naquela réplica (linha 21 – Algoritmo 4). Isto se faz necessário porque as transações que estão na fase de `execução` nas réplicas podem estar detendo a concessão de bloqueios sobre os mesmos itens de dados da transação T_{id}, e isto impedirá a terminação de T_{id} devido a semântica do protocolo 2PL rigoroso, que é utilizado pelas réplicas para a execução local de transações (vide Seção 4.2.2). Deste modo, para permitir a progressão na terminação da transação T_{id}, os bloqueios que estão em posse de transações em fase de `execução` são liberados de maneira compulsória. No caso, as réplicas anulam de maneira não espontânea as transações que são identificadas como detentoras destes bloqueios, para que T_{id} possa adquiri-los e realizar a validação sobre os itens de dados – elas também notificam os respectivos clientes quanto à anulação.

Na etapa seguinte (linhas 22 a 33, Algoritmo 4), cada operação realizada pela transação é executada sobre o banco de dados, a fim de checar a validade dos itens de dados em conformidade com os predicados fornecidos nas operações. É digno de nota que a validação das operações que realizam leituras despreza os itens de dados criados pelas transações concorrentes a T_{id}, estes presentes no conjunto `NWS` (operação $\mathcal{D} \setminus$ `NWS` – linhas 24 e 27 do Algoritmo 4) que fora alimentado na etapa de certificação (linha 10, Algoritmo 4). Esta operação se faz

necessária, pois, tendo havido a inclusão de novos itens de dados que atendam aos predicados fornecidos pelas operações de T_{id}, a validação destas operações retornaria um conjunto de dados diferente daquele obtido na fase de `execução`, e por conseguinte causaria a invalidação dos mesmos devido à ausência da consistência de leitura. Note que, como os itens de dados incluídos pelas transações concorrentes foram validados durante a execução de T_{id}, pela propriedade de **isolamento** (vide Seções 2.1.2 e 2.1.3) na réplica líder de T_{id} eles não estarão disponíveis e visíveis para aquela transação. De outro modo, como T_{id} é iniciada nas réplicas secundárias somente na fase de `terminação` (linha 9 do Algoritmo 3), os novos itens de dados incluídos serão visíveis para T_{id}, já que para a transação ser considerada como concorrente ela deve ter sido validada antes da entrega da mensagem `COMMIT` para T_{id} (cfm. a Figura 33). E se assim não o fosse, as operações executadas na etapa de validação poderiam incorrer em dois fenômenos, o da *leitura-suja* e o da *leitura não-repetível* (vide Seção 2.1.4.1), o que por sua vez, violaria a serialização para T_{id}.

Para cada operação da transação executada na etapa de validação, são alimentados os *read-set* e *write-set* de acordo com o tipo de operação (linhas 25, 28 e 31, Algoritmo 4). Deste modo, ao término da execução das operações, estes conjuntos são comparados, a partir de seus respectivos resumos criptográficos, com aqueles enviados pelo cliente na mensagem `COMMIT` (condição da linha 32 – Algoritmo 4). Se estas comparações forem avaliadas como verdadeiras, é um indicativo de que os itens de dados são válidos e estão em consonância com aqueles obtidos pelo cliente durante a fase de `execução` da transação, e portanto, a transação pode ser validada (linha 33 do Algoritmo 4); ou do contrário, a transação será anulada. Ao concluir o procedimento o resultado dos testes é retornado ao protocolo de terminação – variável $final\_outcome$ (linha 15 – Algoritmo 3). Se a transação é aprovada pelo teste de validação, ela executa o código entre as linhas 16 a 21 (Algoritmo 3), que consiste na última etapa da fase de `terminação`. Nesta última etapa, a transação é alterada para o estado de validada e a lista dos objetos por ela escritos é integrada à lista que contém o mesmo para as demais transações já validadas. Por fim, a transação em questão é apensada ao histórico de transações válidas no sistema ($\mathcal{H}$), seu efeito torna-se persistente no banco de dados (linha 20 – Algoritmo 3) e seu resultado final é definido como *committed*. Por outro lado, a reprovação em qualquer um dos testes do protocolo de terminação (i.e., de certificação ou de validação) resulta na anulação da transação, o que é realizado pelo código das linhas 22 a 25 do Algoritmo 3. Note que,

neste caso, o estado da transação é definido como anulada, seus efeitos intermediários são simplesmente destacados do banco de dados, e seu resultado é definido como *aborted*.

Os últimos passos do protocolo de terminação consistem no envio da notificação contendo o resultado final de transação para o cliente e na liberação dos recursos alocados nas réplicas, para aquela transação. Por sua vez, o cliente considera válido um resultado final para sua transação (i.e., *committed* ou *aborted*), quando receber $f + 2$ notificações mutuamente consistentes e oriundas de diferente réplicas. Note que, uma vez que os testes de certificação e validação são deterministas, e a terminação da transação ocorre na mesma ordem em todas as réplicas corretas – em decorrência do uso de um protocolo de difusão com ordem total, cada réplica correta obterá o mesmo resultado final para a transação, ao término da fase de `terminação`. Um aspecto de relevância sobre o protocolo de terminação decorre do fato de que, logicamente em uma execução serializável onde duas transações concorrentes T_i e T_j são validadas uma após a outra (p. ex.: T_j após T_i), T_j teria acesso a todos os itens de dados escritos por T_i. Todavia, como T_i e T_j são transações concorrentes, no protocolo proposto elas podem ter sido executadas de maneira otimista por réplicas primárias distintas. Por esta razão, o protocolo adota uma abordagem conservadora para assegurar a serialização de transações, no sentido de evitar execuções não-serializáveis em decorrência de conflitos, visto que estes só são passíveis de detecção na fase de `terminação` da transação.

4.6 DISCUSSÃO

Uma discussão sobre a capacidade de tolerar/mascarar/isolar faltas no protocolo proposto pode ser realizada em termos de clientes e réplicas faltosas. É importante recapitular as hipóteses acerca de faltas que foram definidas na Seção 4.2.2, isto é, que um número ilimitado de clientes podem falhar por parada, enquanto que até $f \leq \lceil \frac{n-1}{3} \rceil$ de um total de n réplicas podem falhar de maneira bizantina. Todavia, antes de entrar no mérito da discussão, se faz necessário explicar o motivo pelo qual o protocolo proposto não foi especificado para tolerar clientes que falham de maneira bizantina, a exemplo dos demais trabalhos verificados na literatura (VANDIVER et al., 2007; GARCIA; RODRIGUES; PREGUIÇA, 2011; PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO, 2011).

Ao realizar um retrospecto acerca dos trabalhos correlacionados, com excessão do BFT-DUR (PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO,

2011), que não foi especificado para operar sobre bancos de dados relacionais, tanto o Byzantium (GARCIA; RODRIGUES; PREGUIÇA, 2011) como o HRDB (VANDIVER et al., 2007) se apoiam sobre o controle de concorrência local provido pelos SGBDs das réplicas do banco de dados, como mecanismos de base para assegurar a exatidão/corretude do processamento de transações em seus respectivos protocolos. E o mesmo ocorre com o protocolo proposto por esta tese, conforme já fora especificado na Seção 4.2.2. Um aspecto que é digno de nota, é que a literatura descreve que mecanismos de controle de concorrência presentes nos SGBDs comerciais, que asseguram níveis de consistência mais fortes, tal como o *serializable* e *snapshot* (cfm. Seção 2.1.3.1), são baseados na combinação do protocolo 2PL (BERNSTEIN; HADZILACOS; GOODMAN, 1987) com o mecanismo desejado/implementado por aquele SGBD (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007; JOHNSON; PANDIS; AILAMAKI, 2009; BERNSTEIN; NEWCOMER, 2009). Isto significa que, mesmo quando utilizada uma abordagem multiversão para o controle de concorrência local (p. ex.: por meio do *snapshot isolation*), a exclusão mútua dos itens de dados em atualização pelas transações é realizada com o auxílio de bloqueios (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007; JOHNSON; PANDIS; AILAMAKI, 2009). Tal aspecto implica em uma fragilidade quanto à especificação de protocolos de replicação tolerantes a faltas bizantinas para bancos de dados relacionais, que requeiram consistência mais fortes. Não obstante, esta fragilidade se torna um problema ainda maior quando é suposto que não apenas as réplicas, mas também os clientes podem falhar de maneira bizantina. Tolerar ações arbitrárias por parte de clientes num ambiente replicado de banco de dados relacional, pode sacrificar as propriedades do sistema (p. ex.: *safety* e *liveness*).

É importante ressaltar que, de um modo geral, todos os trabalhos no âmbito de BFT encontrados na literatura assumem como hipótese que um número **ilimitado** de clientes e até $\lceil \frac{n-1}{3} \rceil$ réplicas podem falhar de maneira bizantina (CASTRO; LISKOV, 1999; YIN et al., 2003; KOTLA et al., 2007; GUERRAOUI et al., 2010; VANDIVER et al., 2007; GARCIA; RODRIGUES; PREGUIÇA, 2011; PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO, 2011). De fato, estabelecer um limite para o número clientes faltosos durante uma janela de vulnerabilidade do sistema não é algo usual, tampouco trivial, além de algo difícil de ser controlado, dada a imprevisibilidade exibida pelo comportamento bizantino. Entretanto, em se tratando da semântica transacional a dificuldade é ainda maior, pois no âmbito de uma transação os bloqueios sobre os itens de dados manipulados são mantidos até o término desta (BERNSTEIN; HADZILACOS;

GOODMAN, 1987). No caso, um cliente bizantino pode simplesmente executar uma transação, adquirir a posse dos bloqueios sobre os itens de dados daquela transação e não pedir sua validação. Isso impedirá a concessão dos bloqueios sobre os itens de dados em posse da transação do cliente bizantino à outras transações que queiram manipular aqueles itens de dados, o que afetará a progressão destas, que de acordo com a semântica do 2PL ficarão em estado de espera até a liberação dos bloqueios. No caso do protocolo HRDB, o fato deste depender de um coordenador centralizado, para controlar o processamento das transações, evita não completamente, mas em partes, um agravo maior no sistema quando da ocorrência do problema apontado.

Por outro lado, protocolos que são especificados a partir de uma abordagem otimista para o processamento de transações, como é o caso do Byzantium e do protocolo proposto nesta tese, o agravo ao sistema em decorrência da ação de clientes bizantinos é grande. Note que, diferente do protocolo proposto nesta tese, o Byzantium considera que os clientes podem falhar de maneira bizantina. Para um melhor entendimento sobre o problema em questão, a Figura 37 ilustra um cenário no qual um conjunto de clientes bizantino estão em conluio, conduzindo o sistema a uma situação de blocagem, onde a progressão (i.e., *liveness*) do protocolo é sacrificada. Para a explicação, considere, sem perda de generalidade, um banco de dados $\mathcal{D}$ que contém os itens de dados w, x, y e z. Como na abordagem otimista, durante o processamento da transação, os clientes interagem apenas com uma réplica – a líder, num cenário onde $n = 4$ e $f = 1$, se pelo menos 4 clientes forem bizantinos e estiverem a executar suas transações em réplicas líderes distintas, e sem manifestarem a intenção de validá-las, as demais transações de clientes corretos em execução otimista que tentarem realizar operações sobre os itens de dados em poder destas serão impedidos de fazer progresso.

Note que uma prerrogativa da abordagem otimista de replicação é evitar qualquer tipo de sincronização e/ou comunicação entre as réplicas durante o processamento da transação (WIESMANN et al., 2000), característica esta que é empregada nos protocolos desta tese e também no Byzantium. Como a concessão dos bloqueios durante o processamento da transação ocorre apenas no âmbito local da réplica líder, uma réplica não tem conhecimento dos itens de dados que estão bloqueados nas outras. Neste sentido, considere para o cenário da Figura 37, um caso onde a réplica 1 recebe uma transação T_5, que é executada de maneira otimista e concorrente com T_1, e que esta transação T_5 realiza as operações $w_5[x, 30], w_5[z, 10]$ e $w_5[w, 2]$. A transação T_5 será executada normalmente até o momento em que ela é posta em

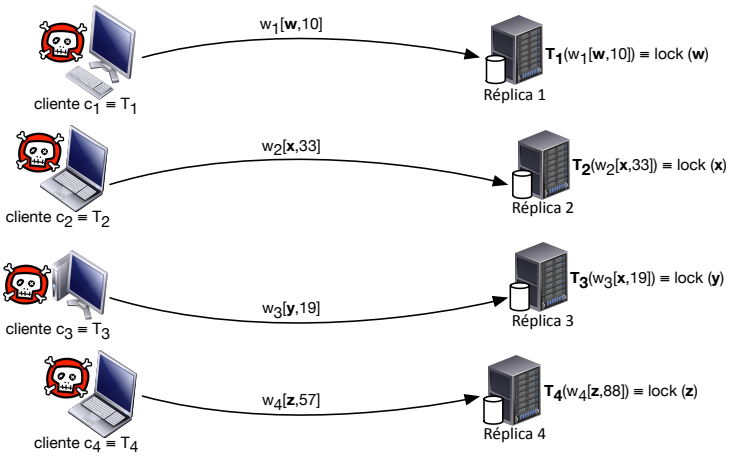


Figura 37 – Fenômeno induzido pela ação de clientes bizantinos.

estado de espera e suspensa, devido à atualização concomitante com T_1 sobre o item de dados w. Como T_5 já havia executado as operações de escrita sobre x e z que não estavam bloqueados localmente na réplica 1, estes itens de dados passam a estar bloqueados na réplica 1, e também só serão liberados ao término de T_5 Neste cenário, há uma clara tendência de que em algum momento o banco de dados da réplica 1 será bloqueado por inteiro, já que a transação emitida pelo cliente bizantino (que se encontra no topo da fila de transações) permanece com a posse dos bloqueios até sua validação, a qual pode nunca ocorrer. O cenário pode acontecer em todas as réplicas, se novas transações que forem executadas fizerem referência aos itens de dados em posse das transações dos clientes faltosos, sendo que, num determinado momento, todo o sistema será bloqueado por completo.

Pela explicação dada ao fenômeno ilustrado na Figura 37, pode-se dizer que o mesmo é análogo a um impasse (i.e., *deadlock*), exceto pelo fato de que as transações que se encontram no topo da fila das réplicas não estão em estado de bloqueio/suspensas, mas intencionalmente paradas devido à ação de clientes bizantinos maliciosos. No caso, pode-se pensar que uma possível solução seria as réplicas anularem as transações que se encontram no topo da fila de bloqueios. Todavia, esta ação um tanto rigorosa poderia agravar a execução do protocolo por duas razões, primeiro porque não se pode determinar se estas transa-

ções são oriundas de clientes bizantinos – o que penalizaria transações oriundas de clientes corretos; segundo porque protocolos de replicação otimista são inerentes à anulações não-espontâneas (KUNG; ROBINSON, 1981), de modo que tal solução agravaria ainda mais a situação. O problema é bastante similar ao que ocorre na comutação de pacotes em protocolos baseados em fluxos de transmissão, como é o caso do TCP. O *head-of-line blocking* (GRIGORIK, 2013) como é conhecido na literatura, ocorre quando um segmento TCP é perdido e segmentos subsequentes chegam fora da ordem. No caso, os segmentos subsequentes são mantidos em *buffer* até que o primeiro segmento seja retransmitido e chegue ao receptor.

É digno de nota que, pelo fato do Byzantium ser baseado na abordagem otimista quanto ao processamento da transação, e presumir que os clientes falham de maneira bizantina, o mesmo está sujeito ao fenômeno da Figura 37, e portanto, o protocolo pode ter sua progressão (*liveness*) sacrificada na ocorrência de tal fenômeno. Esta é a razão pela qual se admite que os clientes não falham de maneira bizantina no âmbito da especificação do protocolo propostos no presente capítulo desta tese. Como os clientes não manifestam o comportamento bizantino, está descartada a hipótese de ocorrência de tal fenômeno durante a execução do protocolo de replicação proposto. Por outro lado, esta situação pode ocorrer quando um cliente cuja transação está no topo da fila sofre uma parada abrupta (um *crash*). Para esta situação, uma solução bastante simples quando a réplica verificar que uma transação em execução não faz progresso (i.e., não evolui) é enviar uma notificação (p. ex.: um *heartbeat*) ao cliente para verificar se ele está ativo. Se o cliente não responder a notificação, a réplica anula a transação para liberar os recursos e evitar gargalos. De outro modo, se o cliente responder, é porque ele pode estar processando os dados recuperados do banco de dados pela última operação executada pela transação. Esta solução é empregada no protocolo proposto.

Em se tratando do comportamento bizantino por parte das réplicas, a situação que causa o maior agravo sobre o protocolo é aquela onde as f réplicas faltosas podem comprometer a semântica local das propriedades ACID. Com isso, todas as transações que forem executadas de maneira otimista sobre elas estarão sujeitas ao recebimento de resultados incorretos ou mesmo a omissão destes. O caso da omissão (p. ex.: ausência do envio de resultados) é tratado pelo Algoritmo 1 onde todas as operações submetidas/enviadas pela transação e não recebidas são reenviadas até que a resposta seja recebida, de modo que, se após um número pré-definido de tentativas a réplica líder não responder, é

lançada uma exceção e o cliente deve reiniciar a transação. No caso da violação da semântica das propriedades durante o processamento da transação por parte de uma réplica comprometida, o protocolo resolve este problema através da fase de `terminação`, de modo que a transação somente é considerada como válida após ser aprovada nos testes de certificação e validação (vide Seção 4.5.3.3) – o que só ocorre se a transação apresentar um estado consistente em cada réplica. Do contrário, ela é anulada e o cliente deve executá-la novamente, o que provavelmente será realizado por uma réplica líder diferente da que o fez na tentativa anterior. Note que qualquer comportamento anômalo manifestado pela réplica líder no processamento otimista da transação é passível de observação pelas réplicas quando da sua terminação. No caso da terminação, o cliente apenas aceita um resultado ao receber $f + 2$ notificações iguais e mutuamente consistentes de diferentes réplicas. Como os procedimentos executados na fase de `terminação` são deterministas, $n - f \geq f + 2$ e no máximo f réplicas podem falhar, sempre haverá um quórum formado por respostas iguais advindas de $f + 2$ réplicas.

4.7 CORREÇÃO DOS ALGORITMOS

Nesta seção é realizada a demonstração de que o protocolo formalizado a partir dos Algoritmos 1, 2, 3 e 4 satisfaz as propriedades que dizem respeito à consistência do banco de dados replicado (i.e., *safety*) atendendo ao critério 1-SR (*one-copy serializability*), bem como da progressão das transações (i.e., *liveness*), de modo que o protocolo seja livre de blocagem – as transações não permanecem bloqueadas indefinidamente e sempre terminam.

Como o protocolo visa o provimento de confiabilidade no processamento de transações, antes mesmo de provar que ele satisfaz as propriedades de consistência e progressão, é imperioso provar que as transações executadas através do mesmo atendem às propriedades básicas e requeridas pelo modelo transacional, conhecidas tradicionalmente como propriedades **ACID** (GRAY, 1981).

Teorema 4.1 *As transações executadas pelo protocolo especificado pelos Algoritmos 1, 2, 3 e 4, quando validadas no banco de dados, satisfazem as propriedades fundamentais* ***ACID****.*

Prova (esboço): Por construção, as réplicas do banco de dados já satisfazem as propriedades ACID, no que concerne à execução local/otimis-

ta da transação, sendo estas providas pelo SGBD local daquela réplica. Deste modo, resta-nos provar que as transações executadas sobre o ambiente replicado asseguram a atomicidade, a consistência, o isolamento e a durabilidade globais (i.e., no âmbito do ambiente replicado). É digno de nota que provas são demonstradas em termos de réplicas corretas (i.e., não faltosas), no sentido de que o comportamento das réplicas faltosas não afeta a corretude do ambiente replicado.

(Atomicidade e Consistência Globais) Pelos Algoritmos 1, 2 pode-se notar que protocolo adota uma abordagem otimista para o processamento da transação. Neste caso, durante a fase de `execução`, a transação é executada apenas numa réplica – a líder/primária. Como cada réplica já garante a atomicidade local para as transações que são executadas sobre elas, isso implica que a atomicidade será garantida quando da execução otimista daquela transação em qualquer réplica. Se durante ou após a fase de `execução` o cliente opta por anular a transação (linha 45, Algoritmo 1), ele o faz com o envio da mensagem `ABORT` via difusão com ordem total a todas as réplicas. Do contrário, se ele optar por validar a transação, ele o faz pelo envio da mensagem `COMMIT` via difusão com ordem total a todas as réplicas (linha 37, Algoritmo 1). Se a opção foi pela anulação, ao entregar a mensagem `ABORT` as réplicas liberaram os recursos alocados para a transação, e a réplica primária descarta os resultados intermediários – ela foi a única que executou a transação até aquele ponto (linhas 41 e 42, Algoritmo 2). Se a opção for pela validação, quando da entrega da mensagem `COMMIT`, as réplicas submetem a transação aos testes de certificação (linhas 6 a 18, Algoritmo 4) e de validação (linhas 19 a 33, Algoritmo 4) para checar se a transação é serializável e se os resultados obtidos na fase de execução são consistentes, respectivamente. Se a transação for aprovada nos dois testes, ela é validada ou, do contrário, é anulada. No caso da anulação, os resultados intermediários são descartados de todas as réplicas, causando a impressão de que a transação não foi executada. Em se tratando da validação, a transação tornar-se-á válida apenas se o estado produzido pela execução otimista puder ser reproduzido em todas as réplicas. Ambos os casos demonstram que em todas as réplicas a transação não é executada parcialmente, mas apenas em sua totalidade, o que, por conseguinte, prova a **Atomicidade Global**. E a considerar que tanto a anulação como a validação são realizadas em todas as réplicas na mesma ordem, e portanto, sobre um mesmo estado em virtude do uso da difusão com ordem total: (*i*) a anulação mantém os estados das réplicas do banco de dados consistentes, já que nenhuma alteração é realizada; (*ii*) o provimento das propriedades ACID em cada réplica

implica que a validação só é possível se conduzir os bancos de dados de um estado consistente para outro estado consistente. Estas asserções provam a **Consistência Global**.

(Isolamento Global) Em decorrência da abordagem otimista (KUNG; ROBINSON, 1981) empregada no protocolo na fase de `execução`, uma transação é processada inicialmente na réplica líder (linhas 22–23 do Algoritmo 1, e 22–37 do Algoritmo 2), sendo que as demais réplicas iniciam a mesma quando da terminação desta (linhas 41 do Algoritmo 1, e 9 do Algoritmo 3). Pelo isolamento local serializável fornecido em cada réplica, durante a fase de `execução` a transação é isolada de outras que estão em execução na mesma réplica líder, e também daquelas cujo processamento da fase de `execução` está sendo realizado pelas demais réplicas. Note que uma réplica líder faltosa, e portanto, comprometida, pode relevar resultados intermediários de uma transação às outras transações em execução na mesma réplica. Pelo Algoritmo 4, iniciado a partir da linha 14 do protocolo de terminação (Algoritmo 3), tal comportamento implicará na reprovação no teste de certificação ou de validação, ou ambos, das transações que não foram isoladas localmente durante sua execução, o que culminará na anulação destas. Em todos os casos, os efeitos intermediários de uma transação só são visíveis às outras transações após a validação desta, o que portanto, assegura o **Isolamento Global** e prova a propriedade.

(Durabilidade Global) De acordo com o protocolo de terminação formalizado pelos Algoritmos 3 e 4, a materialização do efeito produzido pelas alterações/inclusões de uma transação só será realizado pelas réplicas após a validação de transação (linha 20, Algoritmo 3), o que implica na aprovação nos testes de certificação e validação (linha 13 do Algoritmo 3 ou 33 do Algoritmo 4). Note que o protocolo assume que até $f \leq \lceil \frac{n-1}{3} \rceil$ réplicas podem falhar de maneira bizantina, e portanto, não executar a validação. Pelo atendimento local às propriedades ACID por parte de cada réplica, após executar a função $commit\_db\_tx(\mathcal{D}, T_{id})$, as alterações da transação passam a integrar o estado do banco de dados, e não podem mais ser desfeitos. E como $n - f = 2f + 1$ réplicas são corretas, elas executam a função da linha 20 (Algoritmo 3) e tornam as alterações permanentes no banco de dados, o que prova a **Durabilidade Global**. □

No que segue, se faz necessário realizar a demonstração de que o protocolo proposto atende aos seus requisitos funcionais, no sentido de satisfazer as propriedades requeridas em termos de consistência (*safety*) e de progressão (*liveness*). Para a realização destas provas, cabem

alguns esclarecimentos preliminares à luz da literatura pertinente, a fim de subsidiar a demonstração em lide.

Tendo em vista que o critério de correção para a execução de transações é a **serialização** (cfm. Seção 2.1.3), o formalismo adotado para a demonstração em questão é o **grafo de serialização** (do inglês, *Serialization Graph – SG*) (BERNSTEIN; HADZILACOS; GOODMAN, 1987). E conforme a Definição 2.3, também formalizada na Seção 2.1.3, $\mathcal{H}$ compreende o histórico composto por todas as transações executadas e já concluídas (cfm. Definição 4.1) no âmbito de um sistema de processamento de transações.

Definição 4.1 *Uma transação T_i é considerada como concluída (ou terminada) se* $\exists(\mathcal{O}_i, <_i) \in T_i, j = |\mathcal{O}_i| \wedge \exists o_{ij} \in \mathcal{O}_i : o_{ij} = a_i \vee o_{ij} = c_i$.

Para provar que o protocolo proposto satisfaz a propriedade de consistência, o **grafo de serialização** (SG) deve ser mapeado a partir do conjunto que contém apenas as transações cuja terminação resultou na validação destas. No caso, $C(\mathcal{H})$ é a função que retorna uma projeção de $\mathcal{H}$, na qual apenas as operações executadas por transações validadas estão presentes, isto é, despreza-se todas as operações de transações que foram anuladas. Formalmente, a função $C(\mathcal{H})$ pode ser definida como $C(\mathcal{H}) = \{\forall T_i \in \mathcal{H}, 1 \leq i \leq n, (\mathcal{O}_i, <_i) \in T_i, j = |\mathcal{O}_i|, o_{ij} \in \mathcal{O}_i : o_{ij} = c_i\}$ (BERNSTEIN; HADZILACOS; GOODMAN, 1987).

Um aspecto que é digno de nota é que, embora a especificação do protocolo considere cada cópia do banco de dados como *single-version* (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007), o fato do processamento das transações ocorrer de maneira otimista pelas réplicas, permite a ocorrência de atualizações concomitantes sobre um mesmo item de dados, se duas transações que acessam aquele item de dados forem executadas por réplicas líderes distintas. Por conseguinte, esta situação pode resultar em diferentes visões/versões para aquele item de dados que, por razões de consistência, deve convergir para um mesmo estado final em cada uma das réplicas do banco de dados, após a validação de cada transação envolvida no conflito. Neste caso, considere $\mathcal{H}_i$ como o histórico que contém as transações executadas sobre a réplica i, o qual pode ser utilizado como argumento para a função $C()$ para projetar as transações validadas naquela réplica (i.e., $C(\mathcal{H}_i)$) e mapeá-las para o **grafo de serialização** – SG_i. Devemos provar que $\forall r_i, r_j \in \mathcal{R}, i \neq j, C(\mathcal{H}_i) \rightarrow SG_i, C(\mathcal{H}_j) \rightarrow SG_j : SG_i \equiv SG_j$. A demonstração para esta asserção é realizada a partir do Lema 4.1.

Lema 4.1 *Para quaisquer duas réplicas $r_i, r_j \in \mathcal{R}, i \neq j$, tal que r_i e r_j não são réplicas faltosas, se SG_i e SG_j são mapeados após a validação*

da transação $T_k, \forall k > 0$*, a partir de* $C(\mathcal{H}_i)$ *e* $C(\mathcal{H}_j)$*, respectivamente, então* $SG_i \equiv SG_j$*.*

Prova (esboço): Esta prova é demonstrada por indução. Como base de indução, considere que todas as réplicas $r_i \in \mathcal{R}$ do banco de dados, quando iniciadas, partem de um mesmo estado inicial, no qual $\mathcal{H}_i = \varnothing$. Passo de indução: após a validação da transação T_k, se uma réplica correta r_i adiciona T_k ao seu histórico $\mathcal{H}_i$, a projeção $C(\mathcal{H}_i)$ exibirá T_k; então todas as réplicas corretas r_j $(\forall j \neq i)$ também adicionam T_k aos seus respectivos históricos $\mathcal{H}_j$, cujas projeções $C(\mathcal{H}_j)$ exibirão T_k. Neste caso, uma inspeção no Algoritmo 3 – no qual a transação é validada –, mostra que, se T_k foi validada é porque ela executou com êxito sua fase de `terminação` (linhas 16 a 21). Note que a transição para a fase de `terminação` está condicionada à entrega da mensagem `COMMIT`, por todas as réplicas corretas. E para que isso tenha ocorrido, o cliente c_i enviou a mensagem `COMMIT` via difusão com ordem total a todas as réplicas (linha 41, Algoritmo 1). Pelas propriedades de **atomicidade** e **ordenação total** da difusão confiável (vide Seção 2.2.3.3), se a réplica r_i entregou a mensagem `COMMIT` do cliente c_i para a transação T_k, então todas as réplicas corretas r_j $(\forall j \neq i)$ também entregaram aquela mensagem na mesma ordem. Neste caso, todas as réplicas corretas iniciaram a fase de `terminação` para T_k na mesma ordem, e portanto, realizaram sua certificação e validação (linhas 6 a 33 do Algoritmo 4), sobre um mesmo estado anterior contendo as mesmas transações – i.e., um mesmo histórico $\mathcal{H}$. Se T_k teve êxito nos testes de certificação e de validação (condição da linha 16, Algoritmo 3), ela foi adicionada ao histórico local de todas as réplicas corretas e também integrada ao estado do banco de dados local de cada réplica correta (linhas 19 e 20 do Algoritmo 3, respectivamente). Ou então, se T_k não teve êxito em algum dos testes, as réplicas não alteraram os estados de seus bancos de dados locais (linha 24 – Algoritmo 3). Note que o protocolo executado para a fase de `terminação` é especificado como uma máquina de estados replicada, e deste modo, pelo princípio da RME (vide Seção 2.2.3.4), réplicas partindo de um mesmo estado inicial (p. ex.: $\mathcal{H} = \varnothing$) e sujeitas à mesma sequência de operações $(\mathcal{O}_k, <_k)$, contidas numa transação T_k (i.e., com o estado alterado após a validação), chegam a um mesmo estado final. Tal aspecto, implica que todas as réplicas corretas chegam à uma mesma decisão acerca de uma transação, quando da sua `terminação`. E por conseguinte, isto prova que após a validação de T_k, todas as réplicas corretas exibem uma mesma projeção $C(\mathcal{H})$ sobre o histórico $\mathcal{H}$. Portanto, se $r_i, r_j \in \mathcal{R} : i \neq j$ e $C(\mathcal{H}_i) \rightarrow SG_i, C(\mathcal{H}_j) \rightarrow SG_j$, então $SG_i \equiv SG_j$ – isto é, o **grafo de**

serialização mapeado a partir da projeção que contém apenas as transações validadas sobre histórico local das réplicas corretas é o mesmo. □

De acordo com a formalização apresentada por Bernstein *et al.* (1987) para a verificação acerca da corretude de transações, um grafo de serialização SG é serializável apenas se ele é um dígrafo (i.e., um grafo direcionado) (RABUSKE, 1992) e acíclico, onde seus vértices denotam as transações validadas e suas arestas a relação de precedência, baseada nos conflitos entre as transações. Considere que $\mathcal{H}$ denota o histórico de transações para o ambiente replicado, e que $C(\mathcal{H})$ é uma projeção de todas as transações validadas em $\mathcal{H}$; pela correção do Lema 4.1 se verifica que $\forall i \in \mathcal{R}, \mathcal{H} : C(\mathcal{H}_i) \equiv C(\mathcal{H})$, isto é, o histórico local de cada réplica correta é equivalente ao histórico global de transações validadas no ambiente replicado, o que resulta no mesmo grafo de serialização SG. De outro modo, a demonstração acerca da aciclicidade de SG requer uma verificação sobre a relação de precedência para as transações contidas em $\mathcal{H}$. O formalismo adotado na literatura para esta verificação é a relação *reads-from* (BERNSTEIN; HADZILACOS; GOODMAN, 1987) – já explicada na Seção 2.1.3.2. Neste caso, para provar que SG é acíclico, deve ser demonstrado que $\forall T_i, T_j \in \mathcal{H}, i \neq j$, se $T_i \mapsto T_j$, então $\exists\{c_i\} \in \mathcal{O}_i, \exists\{c_j\} \in \mathcal{O}_j : c_i \mapsto c_j$ – nos termos da Definição 4.1.

Lema 4.2 *Se a relação reads-from é verificada para as transações* $T_i, T_j \in \mathcal{H}$*, tal que* $i \neq j$ *e* T_j *reads-from* T_i*; então TO-deliver*$\langle$`COMMIT`$, T_i\rangle \prec$ *TO-deliver*$\langle$`BEGIN`$, T_j\rangle$.

Prova (esboço): Esta prova é uma consequência direta da semântica do protocolo 2PL rigoroso (cfm. Seção 2.1.3.1), empregado como mecanismo para o controle de concorrência em cada uma das réplicas do banco de dados (vide Seção 4.2.2). Pelas regras especificadas pelo 2PL (BERNSTEIN; HADZILACOS; GOODMAN, 1987; WEIKUM; VOSSEN, 2002), num escalonamento serializável uma transação só pode ler do banco de dados valores iniciais de itens de dados, ou valores de itens de dados escritos por transações que já foram validadas. Neste caso, para que a relação T_j *reads-from* T_i seja verificada, $\exists\{w_i[x, v], ..., c_i\} \in \mathcal{O}_i$ e $\exists\{r_j[x], ...\} \in \mathcal{O}_j$, tal que $c_i <_H r_j$. Isto implica que $T_i \mapsto T_j$ e, portanto, que $c_i <_H c_j$. Como as regras impostas pelo 2PL já asseguram a correção do Lema em lide, quanto à execução local da transação em cada réplica, resta-nos provar que o mesmo ocorre para as transações executadas pelo protocolo proposto, sobre o ambiente replicado de banco de dados. Analisaremos os casos nos quais esta asserção é verificada, vejamos:

- **As fases de `execução` de T_i e T_j foram executas por réplicas líderes distintas**: para este caso, uma inspeção nos Algoritmos 2, 3 e 4 mostra que T_j só pode ter lido algum valor escrito por T_i, se T_i já concluiu sua fase de `terminação` e foi aprovada nos testes de certificação e de validação (linhas 6 – 33, Algoritmo 4), o que culminou em sua validação em todas as réplicas (linhas 16 e 20 do Algoritmo 3). De outro modo, suponha que as fases de `execução` de T_i e T_j ocorreram de maneira concorrente, nas réplicas $r_k, r_l \in \mathcal{R}$, respectivamente. Pelo Algoritmo 2, se vê que em nenhuma hipótese T_j pode ter lido algum valor escrito por T_i, já que os efeitos intermediários das operações executadas por T_i estão completamente isolados (vide Seção 2.1.5) no contexto da réplica r_k (linhas 19 – 20 e 30 – 33). Ambas as argumentações demonstram que se $T_i, T_j \in \mathcal{H}, i \neq j : T_j$ *reads-from* T_i, é porque *TO-deliver*$\langle$`COMMIT`$, T_i\rangle \prec$ *TO-deliver*$\langle$`BEGIN`$, T_j\rangle$. Portanto, o Lema segue provado.

- **As fases de `execução` de T_i e T_j foram executas pela mesma réplica líder**: neste caso, para uma contradição, suponha que a validação de T_i não ocorreu antes da validação de T_j, e que as fases de `execução` de ambas ocorreram de maneira concorrente na réplica $r_k \in \mathcal{R}$. Nesta situação, se T_j leu algum valor escrito por T_i, ela fez o como uma **Leitura Suja** (vide Seção 2.1.4.1) – um fenômeno que fere a propriedade de Isolamento (ACID) –, que por construção, não ocorre no controle de concorrência baseado no 2PL. Isto contradiz o argumento em questão, e corrobora com a prova de que, se $T_i, T_j \in \mathcal{H}, i \neq j : T_j$ *reads-from* T_i, é porque *TO-deliver*$\langle$`COMMIT`$, T_i\rangle \prec$ *TO-deliver*$\langle$`BEGIN`$, T_j\rangle$. O que satisfaz o Lema.

Em ambos os casos é possível verificar que se T_i e T_j são duas transações no histórico $\mathcal{H}$, em que a relação *reads-from* é verificada, tal que T_j *reads-from* T_i; é porque *TO-deliver*$\langle$`COMMIT`$, T_i\rangle$ ocorreu antes de *TO-deliver*$\langle$`BEGIN`$, T_j\rangle$. Por consequência, está provada a correção do Lema em questão. □

Lema 4.3 *Se as transações T_i e T_j são exibidas pela projeção $C(\mathcal{H})$, tal que T_j reads-from T_i; então $T_i \mapsto T_j$ e TO-deliver$\langle$`COMMIT`$, T_i\rangle \prec$ TO-deliver$\langle$`COMMIT`$, T_j\rangle$.*

Prova (esboço): Pelo Lema 4.2, a asserção $T_i, T_j \in \mathcal{H}, i \neq j : T_j$ *reads-from* T_i, decorre de *TO-deliver*$\langle$`COMMIT`$, T_i\rangle \prec$ *TO-deliver*$\langle$`BEGIN`$, T_j\rangle$.

Se $T_i, T_j \in C(\mathcal{H})$, é porque T_i e T_j foram validadas no banco de dados, o que sugere a preservação da mesma relação *reads-from* pela projeção $C(\mathcal{H})$. Portanto, para $T_i, T_j \in C(\mathcal{H}) : i \neq j$, onde *TO-deliver*$\langle$`COMMIT`$, T_i\rangle \prec$ *TO-deliver*$\langle$`BEGIN`$, T_j\rangle$, implica que *TO-deliver* $\langle$`COMMIT`$, T_i\rangle \prec$ *TO-deliver*$\langle$`COMMIT`$, T_j\rangle$; de modo que o Lema é satisfeito. □

Tendo os Lemas 4.1, 4.2 e 4.3 em consideração, podemos provar que o protocolo especificado pelos Algoritmos 1, 2, 3 e 4 satisfaz as propriedades de corretude/consistência (*safety*) e de progressão (*liveness*).

Teorema 4.2 *Para toda projeção $C(\mathcal{H}_i)$, que exibe a transação $T_k, \forall k > 0$ validada pelo protocolo, tal que $1 \leq i \leq |\mathcal{R}|$, o critério de consistência one-copy serializability (1-SR) é mantido.*

Prova (esboço): Para esta prova, considere a lemmata 4.1, 4.2 e 4.3. A correção do Lema 4.1 demonstra que, após a validação da transação $T_k, \forall k > 0$; a projeção $C(\mathcal{H})$ sobre o histórico $\mathcal{H}$ de cada uma das réplicas corretas será o mesmo, e por conseguinte resultará no mapeamento de um mesmo grafo de serialização SG por tais réplicas, tal que $C(\mathcal{H}) \rightarrow SG$. Pela correção dos Lemas 4.2 e 4.3, é demonstrado que, quando é verificada a relação *reads-from* entre transações $T_i, T_j, T_k \in \mathcal{H} : i \neq j \wedge i \neq k \wedge j \neq k$, em que T_j *reads-from* T_i e T_j *reads-from* T_k, então $T_i \mapsto T_j \in \mathcal{H}$ e $T_k \mapsto T_j \in \mathcal{H}$. Isso significa que há uma relação de precedência $\forall T_k \in \mathcal{H}$, de modo que, se $C(\mathcal{H}) \rightarrow SG$, então SG é acíclico. De acordo com o Teorema do Grafo de Serialização demonstrado por Bernstein *et al.* (1987), se um grafo de serialização SG é acíclico, então todas as transações denotadas pelos seus vértices são serializáveis. Como $\forall i, j \in \mathcal{R}, i \neq j, 1 \leq i, j \leq |\mathcal{R}| : SG_i \equiv SG_j$ (demonstrado pelo Lema 4.1), isto comprova o teorema. □

Teorema 4.3 *O protocolo é livre de blocagem (i.e., ele sempre termina).*

Prova (esboço): Esta prova é demonstrada por contradição. Para a contradição, suponha, sem perda de generalidade, um cenário onde duas transações T_1 e T_2, dos clientes c_1 e c_2, respectivamente, estão a realizar suas fases de `execução`, e posteriormente suas fases de `terminação`. Considere também que, em algum momento, ambas permanecerão paradas para sempre, uma à espera da outra, caracterizando uma situação de impasse. Demonstraremos que em nenhuma hipótese tal situação pode ocorrer. E, em conformidade com a especificação do protocolo, devemos analisar em quais circunstâncias tal cenário é passível de verificação; assim, temos:

1. **As fases de `execução` de T_1 e T_2 ocorreram em réplicas líderes distintas**: neste cenário, considere que T_1 foi executada por uma réplica r_i, enquanto que T_2 foi executada pela réplica r_j. Pelas linhas 19, 20 e 25, 36 do Algoritmo 2, se T_1 e T_2 estão em conflito (i.e., elas manipulam pelo menos um item de dados em comum), como T_1 é executada pela réplica r_i, ela adquiriu a concessão dos bloqueios sobre os itens de dados apenas naquela réplica, sendo que o mesmo ocorre para T_2, que adquiriu a concessão dos bloqueios sobre os itens de dados apenas em r_j. No caso, o conflito será caracterizado somente no momento em que uma delas tiver entrado em sua fase de `terminação` (Algoritmo 3). Em tal circunstância, a transação que executar sua fase `terminação` antes da outra (p. ex.: T_1 ou T_2), indicará prioridade sobre a concessão dos bloqueios em todas as réplicas, inclusive em r_i e r_j (linha 21, Algoritmo 4). Isto posto, a transação que estiver em fase de `execução` será compelida a anular não-espontaneamente, a fim de liberar todos os bloqueios adquiridos e permitir a concessão destes por parte daquela em fase de `terminação`. Em consequência disto, é assegurada a progressão da transação que não tiver sido anulada.

2. **As fases de `execução` de T_1 e T_2 ocorreram na mesma réplica líder**: neste caso, se T_1 e T_2 estiverem em conflito e este seja caracterizado com um impasse, ele será detectado pela réplica e uma das transações será anulada de maneira compulsória – o próprio SGBD local daquela réplica dispõe de um mecanismo para detecção e resolução de impasses (HELLERSTEIN; STONEBRAKER; HAMILTON, 2007) –, o que permitirá o progresso daquela que não tiver sido compelida à anulação. De outro modo, se T_1 tiver obtido a posse dos bloqueios sobre os itens de dados, a concessão dos bloqueios para T_2 sobre os itens de dados em conflito com T_1 ocorrerá após a `terminação` de T_1; situação em que T_2 aguarda não de maneira indefinida, e tendo, portanto, seu progresso assegurado quando da `terminação` de T_1. Num caso em que T_1 está em fase de `execução` e detém a posse dos bloqueios, mas o cliente responsável por ela veio a falhar por parada, as réplicas líderes periodicamente enviam *heartbeats* aos clientes cujas transações se encontram em fase de `execução` naquelas réplicas (cfm. Seção 4.6). Se o cliente não responde ao *heartbeat*, é porque ele falhou por parada – eles não falham de maneira bizantina. Nesta situação, para assegurar a progressão das transações executadas localmente, as réplicas líderes anulam a(s) transação(ões) destes

clientes para liberar os bloqueios, e assegurar a progressão daquelas que estão em estado de espera devido ao(s) conflito(s).

Note que ambos os casos são analisados apenas quanto à fase de `execução` da transação. Isto porque a fase de `terminação` da transação é executada como uma máquina de estados (cfm. especificação no Algoritmo 3), e em conformidade com a semântica de linearização (HERLIHY; WING, 1990). Como a transição para a fase de `terminação` se dá apenas quando as réplicas entregam a mensagem `COMMIT` para a transação, pelas propriedades da difusão com ordem total (DÉFAGO; SCHIPER; URBÁN, 2004) – usada para o envio da mensagem `COMMIT` –, todas as réplicas corretas entregam a mensagem na mesma ordem. Deste modo, num cenário em que os clientes c_1 e c_2 enviam simultaneamente os pedidos de validação (i.e., a mensagem `COMMIT`) para T_1 e T_2, respectivamente, as réplicas corretas entregarão as mensagens uma após a outra, em uma mesma ordem, de modo a assegurar a linearização acerca da `terminação` de cada transação. Em consequência disto, apenas $T1$ ou T_2 estará em fase de `terminação`, e não ambas; o que, portanto, evita a indicação de prioridade na aquisição de bloqueios por mais de uma transação, simultaneamente (linha 21, Algoritmo 4). Tal argumentação evidencia que a única situação em que haveria a possibilidade de blocagem para este caso, seria aquela já demonstrada pelo cenário 1.

Como o controle de concorrência local de cada réplica é baseado no 2PL (vide Seção 4.2.2), em qualquer situação onde há a aquisição de bloqueio(s) por parte de uma transação T_i, ela o(s) mantém até a sua conclusão, o que ocorre: (i) em caso normal, pela sua fase de `terminação`; (ii) em caso de conflito ou de falha, pela sua anulação não-espontânea por preempção. Pelas argumentações apresentadas, as únicas hipóteses de T_1 e T_2 permanecerem bloqueadas de maneira indefinida seriam: (i) se as mensagens `COMMIT` não fossem entregues uma após a outra, e na mesma ordem por todas as réplicas corretas – o que violaria as propriedades da difusão com ordem total (DÉFAGO; SCHIPER; URBÁN, 2004); (ii) se históricos construídos a partir da execução de transações em conformidade com o 2PL não fossem históricos serializáveis – o que violaria não só as regras, mas também as propriedades especificadas para o 2PL (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Neste sentido, todas as argumentações apresentadas contradizem a afirmação de que ambas as transações permanecem bloqueadas indefinidamente e não terminam; o que, portanto, comprova o teorema em lide. □

4.8 ESPECIFICAÇÃO DE *MIDDLEWARE*

No intuito de verificar se o protocolo proposto é factível em termos de implementação, as especificações apresentadas pelos Algoritmos 1, 2, 3 e 4 foram implementadas seguindo a abordagem para replicação de bancos de dados baseada em *middleware* (CECCHET; CANDEA; AILAMAKI, 2008), isto é, aquela na qual o protocolo de replicação não é integrado ao núcleo do SGBD, mas atua como uma camada intermediária entre a aplicação e o ambiente replicado de banco de dados. Neste sentido, esta seção descreve alguns aspectos acerca da especificação de *middleware* que culminou pela implementação do protótipo, e o modo como se deu o desenvolvimento de alguns dos principais elementos que compõem o ambiente.

Esta especificação de arquitetura de *middleware* visa prover um suporte para o processamento de transações em ambientes de bancos de dados replicados sujeitos a faltas bizantinas (LUIZ; LUNG; CORREIA, 2014). Esta proposta de arquitetura de *middleware* é o resultado da integração de uma série de elementos/componentes desenvolvidos para a realização de um protótipo para o protocolo de replicação apresentado no presente capítulo. Neste sentido, a especificação apresentada por esta seção, se dá em termos dos componentes elaborados para a arquitetura de *middleware* proposta. Note que o objetivo não é apresentar de maneira detalhada e exaustiva a implementação desenvolvida, mas apenas descrever os aspectos essenciais para a realização da avaliação experimental da solução especificada no presente capítulo desta tese. A avaliação do protótipo é apresentada na Seção 6.1.2.

4.8.1 Aspectos de Implementação

O desenvolvimento do protótipo para o protocolo proposto no presente capítulo, o qual decorre da especificação da arquitetura de *middleware* em lide, foi realizado integralmente sobre a plataforma Java, o que foi motivado por algumas vantagens e facilidades oferecidas pela linguagem (p. ex.: portabilidade, segurança, etc.). No caso, o protótipo realizado encapsula os mecanismos requeridos para a tolerância a faltas bizantinas a partir da implementação da especificação padrão da API JDBC (FISHER; ELLIS; BRUCE, 2003). Isto significa que as chamadas às funcionalidades providas pelos algoritmos foram implementadas em conformidade com as especificações daquela API, o que se deu com vista para: (i) proporcionar um ambiente replicado e possivelmente he-

terogêneo, de modo a se apresentar aos clientes como um único SGBD virtual; (*ii*) oferecer uma interface padrão e transparente para as aplicações dos clientes. Note que o suporte à diversidade em termos de SGBDs é importante e essencial em ambientes sujeitos a faltas bizantinas, principalmente porque é pouco provável que diferentes versões de um mesmo sistema compartilhem os mesmos *bugs*, ou as mesmas vulnerabilidades.

A Figura 38 ilustra a arquitetura básica para o protótipo de *middleware* implementado. No caso, a aplicação cliente interage com o banco de dados através de um *proxy*, que consiste na parte do protocolo implementada para o cliente, o qual exporta a API JDBC implementada em substituição àquela que seria normalmente provida por um *driver* JDBC específico do SGBD (p. ex.: um *driver* do MySQL, DB2, Derby, etc.). Note que a parte essencial do *middleware* é aquela implementada no lado das réplicas, e portanto, sobre o SGBD local – o que indica que o protocolo é executado predominantemente pelas réplicas onde o banco de dados é replicado. A figura representa de maneira abstrata a camada de *middleware*, porém, há diversos processos em execução em cada uma das réplicas para executar as tarefas do protocolo, e alguns destes se comunicam uns com os outros por meio de passagem de mensagens. A implementação do *middleware* presente em cada uma das réplicas efetua chamadas para um *driver* nativo, e específico do SGBD de cada réplica, e no lado cliente ele é referenciado pelas aplicações como um *driver* JDBC do Tipo 3 (FISHER; ELLIS; BRUCE, 2003). É digno de nota que os *drivers* nativos estão disponíveis para uma ampla gama de SGBDs (http://www.oracle.com/technetwork/java/index-136695.html).

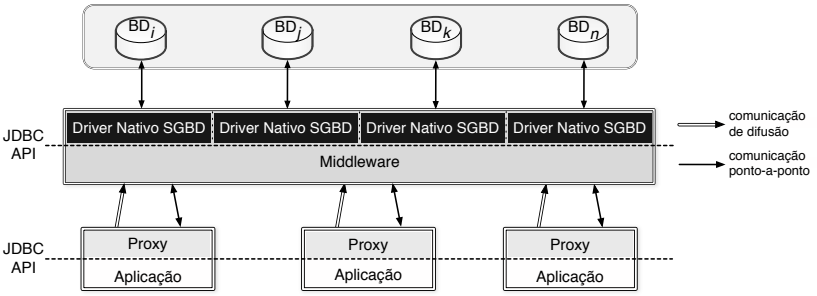


Figura 38 – Arquitetura básica do protótipo de *middleware*.

No caso, o *driver* JDBC implementado para o cliente recebe as operações emitidas pelo cliente no âmbito de uma transação (i.e., em formato SQL), e as encapsula como requisições para serem recebidas pelas réplicas, que posteriormente as executam sobre o banco de dados local. As réplicas por sua vez, se comunicam com os bancos de dados locais através de chamadas ao *driver* JDBC Tipo V, nativo daquele SGBD. Como as operações das extremidades da transação (i.e., `BEGIN`, `COMMIT` e `ABORT`) são enviadas via difusão com ordem total, como implementação deste protocolo utilizamos aquela disponível na biblioteca de replicação do BFT-SMaRt (BESSANI; SOUSA; ALCHIERI, 2014), por ser escrita em Java, e portanto, compatível com a implementação do protótipo, e também por ser estável e eficiente (disponível em http://code.google.com/p/bft-smart/). Um aspecto que merece ser elucidado, é que esta implementação do *middleware* foi realizada apenas com o propósito de servir como uma prova de conceito para o protocolo proposto, embora o código tenha sido implementado com cuidado. Todavia, no estágio atual nenhuma tentativa foi realizada no intuito de torná-lo eficiente a ponto de ser usado em ambientes comerciais.

Alguns aspectos que merecem destaque acerca do desenvolvimento do protótipo são: (*i*) a implementação dos canais de comunicação ponto-a-ponto foi realizada por meio dos *SocketChannels* (i.e., via conexão TCP) providas pela API NIO (HITCHENS, 2002); (*ii*) os protocolos criptográficos utilizados no protótipo quando apropriado (p. ex.: códigos de autenticação de mensagens – HMACs/SHA1, esquemas de assinatura – RSA) são aqueles disponíveis pela JCA (Java Cryptography Architecture); (*iii*) foi implementado um analisador/pré-processador/tradutor de SQL a partir dos pacotes Zql (http://zql.sourceforge.net/) e SwisSQL API (http://www.swissql.com/products/sql-one-apijava); (*iv*) para cada transação criada nas réplicas (i.e., durante a fase de `execução`) é criada uma *thread*; (*v*) o protocolo de `terminação` – que requer a execução com ordem total, é executado em cada réplica por uma única *thread*; (*vi*) o mecanismo para extração dos *write-set*s das transações foi implementado por meio de gatilhos de banco de dados (i.e., *triggers*). Embora o protocolo seja apto a executar em qualquer SGBD que proveja o isolamento serializável, neste estágio de desenvolvimento o protótipo foi preparado para suportar apenas o SGBD MySQL. Isso acontece porque, para implementação de alguns componentes do protótipo de *middleware*, é requerido o acesso às estruturas de dados que compõem o catálogo (i.e., dicionário de dados) do SGBD. Entretanto, isso não impede que ocorra a portabilidade/adaptação do mesmo para execução em outros SGBDs.

4.8.2 Componentes Arquiteturais do *Middleware*

A Figura 39 apresenta o detalhamento da proposta de arquitetura de *middleware*, que corresponde aos componentes especificados para o lado servidor/réplicas. É importante salientar que o *middleware* é executado em cada servidor/réplicas, portanto, esta representação do *middleware* pode ser vista como uma abstração do que ocorre na realidade. A parte que trata do lado dos clientes não é detalhada, devido à sua simplicidade de especificação, que apenas encapsula as chamadas ao banco de dados como mensagens para o protocolo de replicação. No caso, a ilustração da Figura 39 apresenta os componentes que fazem parte da proposta de *middleware*, os quais estão encapsulados em um Gerenciador de Réplica, que é uma entidade presente em cada réplica do sistema. Todavia, cabe ressaltar que a arquitetura em questão é distribuída e os diversos sistemas de gerência de bases de dados (DBMS) da figura denotam a capacidade do *middleware* em operar sobre algumas plataformas de SGBD.

A arquitetura é formada por uma série de componentes, que atuam em conjunto a fim de manter as propriedades, bem como prover as garantias propostas para o sistema. No que segue, são apresentados cada um dos componentes especificados pela arquitetura proposta. As setas e os respectivos números representam os fluxos de dados que transitam entres os componentes, e a ordem na qual eles ocorrem.

4.8.2.1 Gerenciador de Conexão

O **gerenciador de conexões cliente** consiste na interface do ambiente replicado com os clientes, ou seja, é o componente que recebe mensagens dos clientes e as encaminha para os componentes que farão o devido tratamento dos conteúdos destas. Este componente opera diretamente sobre as abstrações de comunicação de médio nível, tal como *sockets* e *channels*. O primeiro passo após o recebimento de uma mensagem de um cliente, consiste em encaminhá-la ao módulo **gerenciador autenticação** para a verificação da validade e autenticidade de seu conteúdo, e também para estabelecer o vínculo entre a mensagem do cliente com a respectiva transação em execução naquela réplica/servidor. Se o **gerenciador autenticação** retorna a informação de que a mensagem é valida, o **gerenciador de conexão** a encaminha para o **gerenciador de requisição** para que seja realizado o tratamento adequado daquela mensagem. Do contrário, a mensagem é descartada e uma excessão é

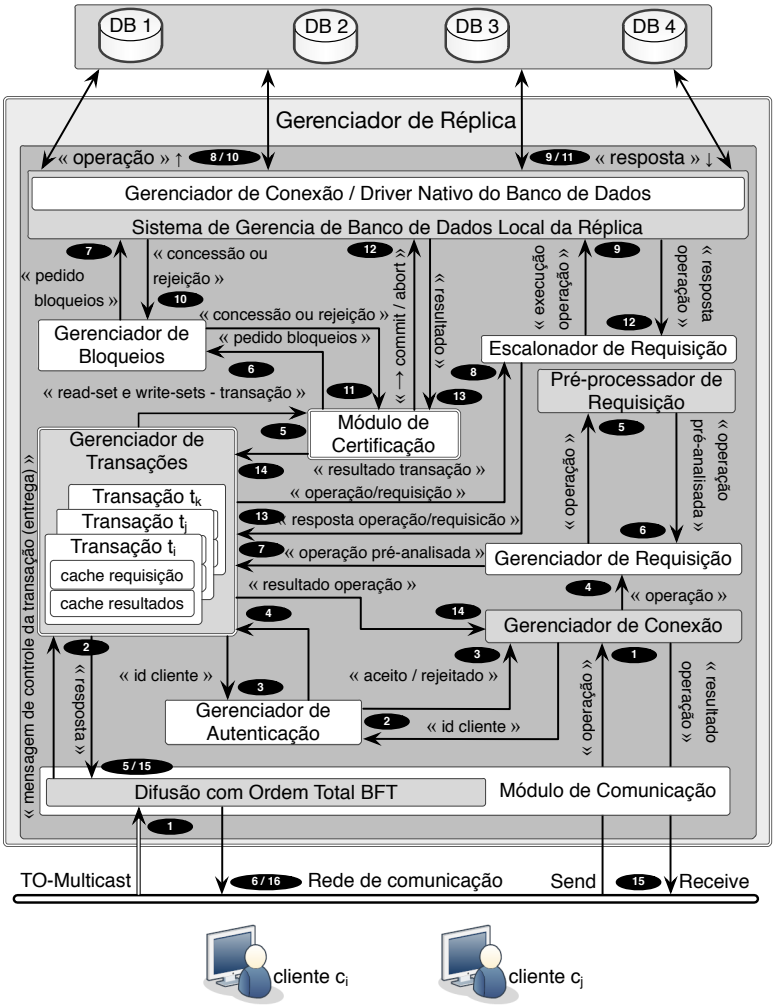


Figura 39 – Elementos que compõem a arquitetura de *middleware*.

enviada ao cliente emissor da mesma. Este componente também é o responsável por estabelecer um canal de comunicação entre a réplica líder de uma dada transação, e o respectivo cliente emissor. Note pela especificação do protocolo, que durante a fase de execução da transação os clientes interagem apenas com uma réplica, a líder.

4.8.2.2 Gerenciador de Requisição

O módulo **gerenciador de requisições** é o que estabelece o vínculo da requisição com o contexto da transação a que ela pertence. A cada requisição recebida em uma transação, o módulo primeiramente realiza a verificação acerca da sintaxe da requisição enviada para certificar se ela está bem formada, isto é, se ela segue os padrões de sintaxe definidos e aceitos para a execução pelo sistema – o que é feito pela invocação ao **pré-processador de requisição** . Uma vez que a requisição é aceita pela avaliação preliminar, ela é direcionada para a transação a que pertence, o que ocorre por meio da invocação ao **gerenciador de transações**, para posteriormente ser enviada para execução no sistema de banco de dados, pelo módulo **escalonador de requisição**.

4.8.2.3 Pré-processador de Requisição

Este componente é responsável por realizar um pré-processamento de uma requisição, no que tange à análise e verificação se a sintaxe está em conformidade com aquela suportada pela réplica local de banco de dados. Um dos propósitos acerca da arquitetura de *middleware* proposta é prover a interoperabilidade em um ambiente composto por réplicas possivelmente heterogêneas, a fim de permitir a diversidade em termos de replicação. Todavia, é lícito salientar que esta funcionalidade ainda não é suportada no estágio em que atualmente se encontra o desenvolvimento do protótipo. O suporte à diversidade/heterogeneidade em termos de réplicas resulta em uma série de problemas intermediários de compatibilidade, os quais devem ser previstos pela especificação do *middleware*. Neste sentido, um dos primeiros e principais problemas decorre do fato de os sistemas de gerência de bancos de dados implementarem diferentes dialetos acerca da declaração de comandos em linguagem SQL, apesar desta possuir uma padronização conhecida por ANSI SQL (ANSI, 1992). Uma solução imediata para o problema seria restringir apenas à declarações em conformidade com o padrão ANSI SQL, mas a experiência mostra que isso é algo limitativo.

Neste caso, o *middleware* resolve este problema através da uma tradução das instruções SQL emitidas pelos clientes para o dialeto SQL nativo da réplica do banco de dados local de cada réplica. Evidentemente que esta tradução consiste em uma tarefa bastante complexa, mas há pacotes de software que são capazes de fazê-lo para a maioria dos dialetos SQL, por exemplo, a API SwisSQL (`http://www.swissql.`

`com/products/sql-translator/sql-converter.html`) ou o SQL-Translator (`http://search.cpan.org/~frew/SQL-Translator-0.11018`). Ademais, como esta tarefa despende um custo sobre a execução da operação, no intuito de prover um melhor desempenho acerca deste processo, as réplicas mantém um *cache* de instruções pré-processadas e já traduzidas para execução, para evitar o re-processamento daquelas instruções usadas com maior frequência. A manutenção deste *cache* é realizada de acordo com a tradicional política de LRU (*least recently used*), onde as informações são removidas do *cache* de acordo com sua obsolescência.

4.8.2.4 Gerenciador de Transações

Este módulo é um dos principais componentes da arquitetura de *middleware*. Ele é responsável por controlar a execução da transação e manter alguns dados referentes às transações ativas e também para as já validadas. Este módulo é responsável pela execução das tarefas que envolvem as operações de controle da transação, com ordem total, em todas as réplicas. No caso, quando uma mensagem de controle de alguma transação é entregue ao *middleware* por meio do protocolo subjacente de difusão com ordem total, ele a encaminha ao módulo **gerenciador de transações** que, de acordo com o contexto presente naquela mensagem (p. ex.: tipo, operação, etc.) executará a ação apropriada. Pela especificação do protocolo de replicação, são três os tipos de mensagens de controle para uma transação: (*i*) mensagem de início de transação (i.e., `BEGIN`), que inicia uma nova transação no âmbito do sistema replicado, e; (*ii*) mensagem do pedido de validação (i.e., `COMMIT`), que modifica o estado da transação e inicia os testes de certificação e validação para aquela transação, a fim de validá-la ou anulá-la, e; (*iii*) mensagem de pedido de anulação espontânea para a transação (i.e., `ABORT`). Como todas as mensagens de controle do protocolo são enviadas pelos clientes para as réplicas por meio de um protocolo de difusão com ordem total, resulta que as ações efetuadas por este componente ocorrem em todas as réplicas na mesma ordem. Esta é uma premissa do sistema para que o protocolo de replicação seja capaz de prover as garantias de consistência das transações e manter o critério de serialização, não apenas em casos normais, mas também perante a ocorrência de faltas bizantinas.

4.8.2.5 Módulo de Certificação

O **módulo de certificação** basicamente implementa as tarefas especificadas pelo protocolo de terminação de transações, as quais essencialmente são: (*i*) verificar a validade de cada item de dados lido pela transação que se encontra em processo de terminação, que pediu sua validação; (*ii*) verificar se os itens de dados lidos através do processamento otimista da transação pela réplica líder, ainda estão atualizados no momento que precede à validação daquela transação; e (*iii*) verificar se os resultados obtidos para todas as operações executadas através do processamento otimista pela réplica líder, estão em consonância com os resultados obtidos pela execução local nas demais réplicas. Note que o módulo de certificação entra em ação apenas durante a fase de `terminação` da transação, cuja atuação é imperiosa e essencial para assegurar a consistência da transação, bem como para atender ao critério de serialização.

Neste sentido, a atuação do **módulo de certificação** assegura que uma transação é validada apenas se ela é consistente, se ela leu apenas itens de dados válidos, e também que as leituras realizadas por ela não entraram em conflito com escritas realizadas por outras transações concorrentes, cuja validação ocorreu no decurso da transação que está a ser certificada (vide Seção 4.5.3.3). É importante observar que todas as tarefas realizadas por este módulo são necessárias, porque uma réplica líder bizantina pode enviar itens de dados espúrios como resposta às operações executadas por um cliente, ou também, pode responder às declarações com dados obsoletos. Ademais, devido ao uso da primitiva de difusão com ordem total para o envio da mensagem que contém o pedido de validação para a transação – o que dá início à execução do módulo de certificação – todas as réplicas entregam a mensagem `COMMIT` para a mesma transação na mesma ordem, e por isso elas podem certificar e validar uma transação da mesma maneira. Por último, uma vez que o **módulo de certificação** realiza as ações cruciais atinentes à terminação da transação, tal como a certificação e a validação de maneira serializada – o que decorre da execução do protocolo de terminação nos termos do critério de linearização (HERLIHY; WING, 1990) – isso garante o determinismo e também a ausência de divergências no estado lógico das réplicas.

4.8.2.6 Gerenciador de Bloqueios

O módulo **gerenciador de bloqueios** é o mecanismo responsável por realizar a aquisição dos bloqueios de leitura e de escrita sobre os itens de dados afetados pela transação, em conformidade com as instruções executadas de maneira otimista pela réplica líder e enviadas pelo cliente junto à mensagem que contém o pedido de validação para a transação. Assim, o gerenciador de bloqueios atua como uma espécie de escalonador de requisições de bloqueio sobre os itens de dados, no banco de dados local de cada réplicas. Cabe ressaltar que como os pedidos de validação são propagados através da primitiva de difusão com ordem total, isto implica que eles ocorrem na mesma ordem em todas as réplicas corretas. E seguindo esta premissa, a ordem na qual ocorre a concessão dos bloqueios é a mesma na qual ocorre a entrega da mensagem do pedido de validação para a transação.

4.8.2.7 Escalonador de Requisição

Quando uma nova requisição é recebida, após a execução de todo o processo de validação e associação com a respectiva transação, ela é disponibilizada ao escalonador para execução pelo sistema de gerência de banco de dados local. O escalonador é responsável por lidar com os aspectos atinentes ao controle de concorrência da réplica em cooperação com o sistema de gerência de banco de dados local, bem como coordenar a execução das requisições de uma transação, de acordo com a respectiva ordem de envio das mesmas. Todas as operações são executadas de maneira síncrona, no sentido de que o escalonador aguarda o recebimento da resposta da réplica líder para a requisição enviada, para em seguida retornar/enviar o resultado ao respectivo cliente. Neste caso, se a resposta para uma requisição não é recebida dentro de um intervalo de tempo aceitável, o escalonador retorna uma exceção como resposta à requisição, através de uma notificação enviada ao cliente. Neste caso, o cliente pode reenviar o comando para execução, se assim desejar.

4.8.2.8 Gerenciador de Autenticação

O módulo **gerenciador de autenticação** estabelece o mapeamento entre o *login* e senha fornecidos pela aplicação cliente, e o *login* e senha a ser usado em cada uma das réplicas da base de dados. Assim,

o *login* e senha enviados pela aplicação não são, de fato, os dados de acesso à base de dados local, mas sim uma ponte de acesso ao sistema. A funcionalidade visa incrementar a segurança do ambiente, pois os dados de acesso à base são exclusivos para cada uma das réplicas. Outrossim, é permitido que cada uma das réplicas tenha diferentes *logins* e senhas para acesso a um mesmo banco de dados. Este módulo também verifica a autenticidade das mensagens recebidas dos clientes.

Embora a especificação deste componente, no estágio atual de desenvolvimento da arquitetura de *middleware* ocorra nestes moldes, em determinados cenários tal especificação pode não ser aceitável. O fato de haver exposição das credenciais dos clientes para acesso aos respectivos *schemas* presentes nos bancos de dados locais de cada réplica, pode incorrer na situação em que uma réplica bizantina venha a revelar os dados de acesso aos seus *schemas* locais, embora tais dados sejam apenas referentes àquela réplica. Uma possível solução seria utilizar um esquema de autenticação nos bancos de dados locais baseado em *single-sign-on*, já que este mecanismo suporta o uso de diferentes aplicações e recursos, com diferentes mecanismos de autenticação (MAGYARI; GENGE; HALLER, 2009). Acredita-se que uma alternativa viável seria estender e/ou melhorar a especificação deste módulo através do uso de um esquema de *single-sign-on* baseado em certificados (MAGYARI; GENGE; HALLER, 2009).

4.9 CONSIDERAÇÕES DO CAPÍTULO

Embora os primeiros indícios da ocorrência de faltas bizantinas em bancos de dados tenham surgido há aproximadamente trinta anos (GARCIA-MOLINA; PITTELLI; DAVIDSON, 1984, 1986), somente na última década é que surgiram na literatura as primeiras discussões acerca de aspectos de cunho mais práticos (GASHI; POPOV; STRIGINI, 2007; VANDIVER et al., 2007; PREGUIÇA et al., 2008). No entanto, dentre os existentes na literatura, apenas dois apresentam soluções algorítmicas (e suas respectivas implementações) para lidar com faltas bizantinas em bancos de dados relacionais, que são o HRDB (VANDIVER et al., 2007) e o Byzantium (PREGUIÇA et al., 2008; GARCIA; RODRIGUES; PREGUIÇA, 2011). No entanto, ambos apresentam algumas limitações, no sentido de que o protocolo especificado para o HRDB (i.e., o CBS – *Commit Barrier Scheduling*) requer uma entidade centralizada – que considera com confiável para conduzir o protocolo de replicação, uma hipótese bastante difícil de ser substanciada na prática num ambiente replicado

sujeito a faltas bizantinas; e o Byzantium, por ser especificado sobre a semântica do *snapshot isolation* (vide Seção 2.1.5), embora não necessite de uma entidade centralizada, tal como o HRDB, a semântica de consistência baseada no *snapshot isolation* é suscetível a algumas das anomalias de concorrência apresentadas na Seção 2.1.4.1, o que a torna não adequada para aquelas aplicações onde a consistência/integridade do banco de dados não pode ser violada.

A proposta discutida neste capítulo estende estes resultados, pela especificação de um protocolo que busca resolver as limitações encontradas nestes dois trabalhos. Neste sentido, é proposto um protocolo para tolerar faltas bizantinas em bancos de dados relacionais, a partir do uso de replicação. Por meio dos algoritmos apresentados, foi demonstrado que a solução opera de maneira distribuída e é capaz de obter uma serialização completa das transações, e portanto, é fortemente consistente. Para tanto, o protocolo se baseia parcialmente na ideia proposta pelo DBSM (PEDONE; GUERRAOUI; SCHIPER, 2003) – especificada para faltas de parada –, no sentido de que adota a mesma abordagem para a propagação da transação, em virtude dos bons resultados verificados para faltas por parada. Deste modo, o estudo apresentado nesta parte da tese tem sua originalidade pautada no fato de ser o primeiro (e até o momento o único) a prover um controle de concorrência que visa a corretude baseada na serialização, cuja operação do protocolo ocorre de maneira distribuída. Por esta razão, ele pode ser considerado um avanço em relação à literatura pertinente.

Em se tratando da implementação do protocolo proposto, dois aspectos foram observados. O primeiro é que a utilização da abordagem otimista para o processamento de transações sobre ambiente replicado explora um grau de paralelismo equivalente ao número de réplicas para a execução de transações concorrentes, o que em tese, poderia vir ser encarado como um fator para ganhos de desempenho sobre um ambiente centralizado. Porém, a despeito da exploração do paralelismo para a execução da transação, a terminação desta tem de ocorrer de maneira sequencial, com vista para a consistência do ambiente replicado, o que implica na perda de desempenho. O segundo aspecto observado, é que para o caso dos protocolos de replicação tolerantes a faltas de parada baseados na mesma abordagem, o tempo consumido pela fase de terminação da transação é geralmente negligenciável, uma característica que não ocorre em protocolos tolerantes a faltas bizantinas – cuja validação das operações executadas de maneira otimista é requerida a para atestar a validade dos resultados obtidos pelas mesmas.

5 VALIDAÇÃO ATÔMICA NÃO-BLOQUEANTE COM FALTAS BIZANTINAS

Conforme mencionado na Seção 2.2.1, um sistema de computação distribuída pode ser entendido como um conjunto de elementos que realizam ações locais e também trocam informações, a fim de executar, de maneira cooperativa, ações e tarefas globais. Para a realização cooperativa de tarefas/ações globais e de acordo com a natureza da computação, tipicamente é requerido algum tipo de sincronização entre os elementos que participam daquela computação. Esta sincronização requerida entre os elementos é conhecida na literatura como problemas de acordo (PEASE; SHOSTAK; LAMPORT, 1980).

A despeito da maturidade acerca da investigação de problemas de acordo (WENSLEY et al., 1978; PEASE; SHOSTAK; LAMPORT, 1980), ainda existem algumas restrições quanto à possibilidade de realizar algumas instâncias desta classe de problemas em determinados modelos de computação (RAYNAL; SINGHAL, 2001; CHARRON-BOST, 2001; CHARRON-BOST; SCHIPER, 2004), e principalmente no que tange aos aspectos práticos presentes na concepção dos algoritmos. Dentre estes problemas está a validação atômica não-bloqueante (ou NBAC) (BABAOGLU; TOUEG, 1993), cujo foco é centrado na preservação da atomicidade em presença de falhas, no modelo de transações distribuídas. Neste sentido, este capítulo apresenta a proposição de uma arquitetura de sistema distribuído, na qual é possível resolver o problema da validação atômica não-bloqueante a partir de um modelo híbrido de falhas, em que parte dos elementos pode estar sujeito à faltas bizantinas.

5.1 INTRODUÇÃO

A noção de transações é essencial para a especificação e implementação de sistemas computacionais, para os quais a consistência acerca da execução de um conjunto de operações sobre dados deve ser rigorosamente preservada. Não obstante, assegurar a terminação de uma transação de maneira consistente em um sistema distribuído não é uma tarefa trivial. Para tal, é requerido o estabelecimento de um acordo entre os participantes envolvidos na transação, a fim de validar (do inglês, *commit*) ou anular (do inglês, *abort*) as operações que a compõem (atomicidade), de modo a torná-las permanentes (durabilidade) ou descartá-las do sistema. O problema que define a validação atômica

de transações em sistemas distribuídos é conhecido na literatura como **Validação Atômica Não-Bloqueante** (do inglês, *Non-Blocking Atomic Commitment* – NBAC) (BABAOGLU; TOUEG, 1993).

O NBAC pode ser visto como um **problema de acordo** em sistemas distribuídos (PEASE; SHOSTAK; LAMPORT, 1980; HADZILACOS, 1990; GUERRAOUI, 1995), e como tal, está sujeito às mesmas restrições teóricas e aos aspectos práticos presentes na concepção de sistemas tolerantes a faltas. A essência da maioria dos problemas de acordo definidos pela literatura é capturada pelo consenso (RAYNAL; SINGHAL, 2001; CHARRON-BOST, 2001; CHARRON-BOST; SCHIPER, 2004), isto é, aquele onde todos os elementos que fazem parte de uma computação distribuída devem concordar acerca da proposição de algum valor, em que tal valor depende da natureza do problema em questão. De um modo geral, protocolos que resolvem o consenso são especificados através de algoritmos que procedem por meio de algumas fases e de sucessivas rodadas (i.e., *rounds*), que são repetidas até que todos os elementos corretos do sistemas cheguem à uma decisão (BEN-OR, 1983; BRACHA; TOUEG, 1985; CHANDRA; TOUEG, 1996; MOSTÉFAOUI; RAYNAL, 1999). Em ambientes puramente assíncronos, a ausência de sincronia pode impedir o progresso de um protocolo de consenso em meio à ocorrência de falhas (FISCHER; LYNCH; PATERSON, 1985), e neste caso, deve-se recorrer ao uso de **detectores de falhas** (CHANDRA; TOUEG, 1996) para que seja possível assegurar a terminação do consenso neste tipo de ambiente.

O problema da validação atômica de transações já fora resolvido tanto em cenários livres de falhas, como naqueles em que se espera a ocorrência de falhas, o que ocorreu pela introdução dos protocolos de validação em duas fases (ou *Two-Phase Commit* – 2PC) (SKEEN, 1981) e de validação em três fases (ou *Three-Phase Commit* – 3PC) (SKEEN, 1981), respectivamente. Apesar da existência de tais soluções de longa data, ao longo dos anos um grande número de soluções para o NBAC foram propostas na literatura, algumas no intuito de circunscrever restrições e também ampliar a resiliência a despeito de faltas por parada (p. ex. *crash*) (GUERRAOUI; LARREA; SCHIPER, 1995; GUERRAOUI, 1995; GUERRAOUI; LARREA; SCHIPER, 1996; PARK; LEE; YU, 2013), enquanto que outras foram desenvolvidas no intuito de prover otimizações quanto à terminação do protocolo (GUERRAOUI; SCHIPER, 1995a; ABDALLAH; PUCHERAL, 1999; GREVE; NARZUL, 2002). Um aspecto particularmente interessante presente nestes estudos, é que alguns deles demonstraram as condições necessárias para a resolução do NBAC (p. ex.: modelos de sistema, ambientes computacionais, modelos de falhas,

detectores de falhas, etc.), além de terem evidenciado as situações nas quais o problema é solúvel.

Embora o NBAC seja um problema bem estabelecido na literatura, tanto em cenários com ou sem faltas por parada, até o momento ele permanece sem uma solução no tocante a faltas bizantinas (LAMPORT; SHOSTAK; PEASE, 1982). A impossibilidade de tal solução advém de um importante resultado teórico, no qual fora demonstrado que nenhum problema de acordo baseado na propriedade de **acordo uniforme** pode ser resolvido em meio ao comportamento arbitrário oriundo de faltas bizantinas (CHARRON-BOST; SCHIPER, 2004). Isto decorre do simples fato de que o modelo bizantino não assegura a decisão para um processo faltoso, pois nada é previsível a respeito do comportamento arbitrário/bizantino. No caso, como um processo bizantino pode realizar as transições de estado de um algoritmo de maneira arbitrária, não é possível assegurar a propriedade de **acordo uniforme** para um processo faltoso. Este resultado é estendido para outros problemas de acordo especificados a partir da propriedade **uniforme**, como é o caso do NBAC. Particularmente, o NBAC não é solúvel porque mesmo que um participante bizantino reúna as condições necessárias para validar uma transação, ele pode optar por anulá-la propositadamente para deturpar o protocolo. Ou ainda, se o protocolo decidir pela validação, não se pode garantir que o processo bizantino realizará as ações pós-decisão (p. ex.: tornar as escritas permanentes).

Neste sentido, a contribuição deste capítulo consiste em uma abordagem prática para resolver a Validação Atômica Não-Bloqueante em um ambiente onde os participantes de uma transação podem falhar de maneira bizantina. Para isso, a solução explora um novo espaço de projeto, a partir da especificação de uma arquitetura de sistema distribuído baseada em um modelo de falhas híbrido e realista, na qual são admitidas algumas distinções em relação do modelo clássico especificado para o NBAC (BABAOGLU; TOUEG, 1993). Não obstante, um protocolo de validação atômica não-bloqueante original e diferente de todos os trabalhos existentes na literatura também é especificado. Ao nosso conhecimento, trata-se de um trabalho pioneiro da literatura no que concerne a abordagem do problema NBAC com um modelo híbrido de falhas, de tal maneira que seja possível tolerar faltas de natureza arbitrária durante a execução do protocolo de validação da transação. Isto posto, pode-se dizer que esta proposição representa uma contribuição no âmbito das áreas de algoritmos distribuídos tolerantes a faltas, e de arquiteturas e ambientes para a concepção de sistemas distribuídos tolerante a faltas.

5.2 CARACTERIZAÇÃO DO PROBLEMA

A validação atômica pode ser entendida como um problema fundamental acerca do gerenciamento de transações em sistemas de computação distribuída tolerantes a faltas. Tipicamente, uma transação distribuída consiste em um conjunto parcialmente ordenado de operações de leitura e de escrita, que são realizadas sobre dados residentes em diversos locais – estes conhecidos por sítios (BABAOGLU; TOUEG, 1993). A execução de uma transação distribuída envolve um conjunto de atores – os participantes da transação, que cooperam entre si para a execução das operações que compõem a unidade lógica de processamento transacional, nos diversos sítios engajados naquela transação (BABAOGLU; TOUEG, 1993). Em cada sítio engajado na transação distribuída reside um participante daquela transação, e este por sua vez, pode desempenhar dois papéis durante a execução da transação no sítio a que pertence; o de **gestor de transação** (*Transaction Manager* – TM) e o de **gestor de dados** (*Data Manager* – DM) – também denominado/conhecido por **gestor de recursos** (*Resource Manager* – RM) (BABAOGLU; TOUEG, 1993).

Na especificação clássica do problema NBAC (BERNSTEIN; HADZILACOS; GOODMAN, 1987; BABAOGLU; TOUEG, 1993), o TM tem como atribuições receber a(s) operação(ões) que lhe foi(ram) solicitada(s), e realizar a cooperação com os demais sítios através dos outros TMs, no decurso da transação. De outro modo, o DM é o responsável por executar a(s) operação(ões) sobre o(s) dado(s)/recurso(s) ao(s) qual(is) a(s) transação(ões) faz(em) referência naquele sítio (BABAOGLU; TOUEG, 1993). Ao término da execução de todas as operações que compõem a transação distribuída, o TM de um sítio solicita aos demais TMs residentes noutros sítios, o encerramento da transação. Deste modo, como a transação distribuída é executada em múltiplos sítios, a atomicidade acerca das operações realizadas no contexto de uma transação depende de um acordo entre os participantes engajados naquela transação. Não obstante, os participantes, por meio de seus TMs, devem decidir de maneira única e irrevogável sobre o resultado final da transação, o qual será pela sua validação (`COMMIT`) – se tudo ocorreu conforme o esperado, ou pela sua anulação (`ABORT`) – caso algo anormal tenha ocorrido.

Este acordo envolvendo o encerramento da transação é realizado através de um protocolo de validação atômica, o qual tem por finalidade assegurar a concordância de todos os participantes quanto ao resultado final da transação, sendo este resultado determinado de acordo com as condições locais de cada participante. Com isso, ao concluir o processa-

mento da transação, cada participante, por meio de seus DMs emite um voto (`SIM` ou `NÃO`), que declara a sua capacidade em garantir a consistência e a durabilidade da parte que lhe coube da transação. E a partir dos votos recebidos é calculado um resultado, que normalmente é `COMMIT` se todos votaram SIM ou, do contrário, o resultado é `ABORT`. Formalmente, o problema da validação atômica não-bloqueante (NBAC) é definido pelas propriedades de **Acordo Uniforme**, **Validade Uniforme**, **Integridade Uniforme**, **Terminação** e **Não-Trivialidade**, todas já definidas e apresentadas na Seção 2.2.3.2.

Uma observação importante acerca do NBAC decorre da propriedade **Não-Trivialidade**, cuja definição original desta propriedade considerava a resolução do problema no contexto de sistemas síncronos (BERNSTEIN; HADZILACOS; GOODMAN, 1987), e que mais tarde fora estendida para sistemas parcialmente síncronos (COAN; WELCH, 1990). Todavia, tendo em consideração os ambientes assíncronos, esta propriedade não pode ser plenamente satisfeita em sistemas de computação distribuída sob condições de falha, mesmo com um suporte de detecção de falhas (GUERRAOUI, 1995). Esta afirmação tem sua fundamentação no fato de que os detectores de falhas são especificados em termos de suspeição de falhas, e não da detecção de fato (CHANDRA; TOUEG, 1996). Deste modo, como as falhas não podem ser verificadas de maneira segura a partir dos detectores de falhas, os resultados verificados para o consenso em ambientes assíncronos não são extensíveis ao NBAC. No caso, apenas a **Validação Atômica Fraca Não-Bloqueante** (*Non-Blocking Weak Atomic Commitment* – NBWAC) (GUERRAOUI, 1995) é que pode ser resolvida em ambientes assíncronos. A única diferença entre o NBAC e o NBWAC está justamente na propriedade **Não-Trivialidade**, que no segundo é enfraquecida para levar em conta **suspeição de falhas** em vez de **ocorrência de falhas**.

Por outro lado, a especificação do NBWAC considera apenas a suspeição de faltas por parada (i.e., *crash*), de modo que faltas bizantinas não são toleradas. No que se refere a tolerância a faltas arbitrárias/bizantinas no contexto de transações distribuídas, alguns trabalhos propuseram soluções para serem empregadas como protocolos de terminação/validação distribuída de transações (MOHAN; STRONG; FINKELSTEIN, 1983; ROTHERMEL; PAPPE, 1993; ZHAO, 2007), porém, nenhum destes foi especificado sobre as propriedades definidas para o NBAC. Portanto, eles são sensivelmente diferentes dos protocolos de validação atômica não-bloqueante, razão pela qual eles não podem ser considerados como soluções para a validação atômica, mas apenas como soluções para a validação distribuída. Por assim dizer, ao conhecimento

do proponente desta tese, tanto o NBAC como o NBWAC permanecem sem solução para cenários onde os participantes podem falhar de maneira arbitrária/bizantina. O que, portanto, justifica a proposição de um protocolo que seja capaz de lidar com faltas bizantinas. No entanto, é importante ressaltar que já fora discutido na literatura sobre a impossibilidade de resolução do NBAC/NBWAC com faltas bizantinas (CHARRON-BOST; SCHIPER, 2004). Portanto, o que se pretende como contribuição desta parte da tese, é circunscrever tal impossibilidade, a partir da introdução de um novo ambiente de computação distribuída pela hibridização do modelo de falhas, de tal forma que ele permita obter as condições necessárias para resolver o problema com a admissão de faltas bizantinas. Esta solução será apresentada nas próximas seções deste capítulo.

5.3 UM BREVE PANORAMA SOBRE SISTEMAS HÍBRIDOS

5.3.1 Definição de Ambiente Híbrido

Uma noção bastante clara de modelo híbrido de falhas para sistemas de computação distribuída foi apresentada por Veríssimo (VERÍSSIMO, 2006). Esta definição se deu em face aos desafios encontrados à época, quanto à especificação de modelos e arquiteturas para sistemas de computação distribuída tolerantes a faltas. Os modelos/sistemas híbridos são uma alternativa para lidar com impossibilidades e restrições teóricas em face a algumas barreiras encontradas na resolução de problemas de acordo em sistemas distribuídos. A hibridização parte da conjectura de que em determinados ambientes, alguns componentes são mais simples de se controlar do que outros, e por esta razão eles podem estar sujeitos a comportamentos distintos daqueles verificados para os demais componentes do sistema. Na literatura são encontradas duas propostas para a hibridização de sistemas.

A **hibridização quanto ao modelo**, que consiste na abordagem onde tanto os elementos que compõem o sistema (p. ex.: os nós), como os canais que ligam estes elementos uns aos outros podem estar sujeitos à diferentes tipos de falhas e aspectos de sincronismo. Esta proposta admite que uma parte dos elementos se comuniquem de maneira síncrona, enquanto que os demais o fazem de maneira assíncrona (BALDONI; MARCHETTI; TERMINI, 2002), ou ainda, que alguns elementos falhem de maneira bizantina e outros falhem apenas por parada (MEYER; PRADHAN, 1991).

Por outro lado, a **hibridização em nível de arquitetura** é aquela onde cada elemento do sistema é constituído por um conjunto de componentes, de modo que cada componente pode operar sob diferentes perspectivas e pressupostos em termos de falhas e de sincronismo. No caso, os componentes que operam sob hipóteses mais fortes são conhecidos na literatura por *wormholes* (CORREIA et al., 2002; CORREIA; VERÍSSIMO; NEVES, 2002). O ambiente TTCB (*Trusted Timely Computing Base*) é um exemplo de sistema que faz uso de *wormholes*, no qual cada nó possui um componente seguro e inviolável que não admite faltas bizantinas (um núcleo), sendo este núcleo ligado aos núcleos dos demais nós através do *wormhole*. No caso, os algoritmos especificados sobre o ambiente TTCB admitem faltas bizantinas e ambientes assíncronos, mas quando apropriado e necessário, fazem uso do serviço provido pelo TTCB para executar os passos considerados como críticos.

5.3.2 Hibridização através de Componentes Confiáveis

Alguns trabalhos também exploram ambas as abordagens de hibridização e de elementos confiáveis para a proposição de soluções para tolerar faltas bizantinas. É o caso do trabalho de Correia *et al.* (2004), que baseado nos *wormholes* propôs um protocolo para RME tolerante a faltas bizantinas capaz de tolerar até f faltas dentre $2f + 1$ réplicas. Para tanto, o protocolo faz uso de um serviço confiável de difusão ordenada de mensagens (*Trusted Multicast Ordering* – TMO), o qual é executado entre os demais componentes confiáveis e só pode falhar por parada, além de ter o relógio sincronizado com os demais componentes confiáveis. No caso, o TMO é que fornece o subsídio para o protocolo de difusão com ordem total, por meio da atribuição de um número de ordem/sequência para as mensagens. Em suma, quando um processo do sistema deseja difundir uma mensagem com ordem total, ele envia aquela mensagem por meio de uma rede assíncrona aos demais processos, e entrega ao TMO um resumo criptográfico (*hash*) daquela mensagem. Quando outro processo recebe a mensagem, ele também entrega ao TMO um *hash* desta. Deste modo, quando o TMO recebe um número suficiente de *hashes* para uma mesma mensagem, ele atribui um número que indica a ordem daquela mensagem e envia este número aos processos, que por conseguinte, entregam a mensagem de acordo com o TMO. Note que os componentes TMO locais de cada processo se comunicam uns com os outros através de uma rede controlada e síncrona.

Embora o componente confiável introduzido por Correia *et al.* (2004) tenha dois grandes feitos, que foram circunscrever a condição FLP e reduzir o número de réplicas de $3f + 1$ para $2f + 1$, tal componente pode ser visto como consideravelmente complexo. E neste ensejo Veronese *et al.* (2010) propuseram um protocolo de nome EBAWA, que ao contrário do TMO, utiliza de um componente confiável de simplicidade considerável. Este componente denominado USIG (*Unique Sequential Identifier Generator*), fornece apenas um contador para as operações executadas em uma réplica, e um verificador para atestar a validade e autenticidade dos contadores das outras réplicas. Para isso, o USIG que é executado em todas as réplicas, atribui um identificador incremental único, monotônico e sequencial para cada mensagem, onde também gera um certificado de corretude para o contador gerado, o qual pode ser verificado pelos USIGs das outras réplicas. No caso, o EBAWA se baseia em alguns aspectos desenvolvidos para o protocolo de nome *Spinning* (VERONESE et al., 2009) dos mesmos autores, porém, neste último, devido ao uso do componente confiável há uma redução do número de réplicas necessárias para tolerar as faltas bizantinas para $2f + 1$.

5.3.3 Hibridização através de Virtualização

Um dos trabalhos pioneiros a discorrer sobre hibridização por meio de máquinas virtuais, com vista para tolerar faltas bizantinas, foi o VM-FIT (REISER; KAPITZA, 2007). Neste trabalho, os autores utilizaram da tecnologia de virtualização (vide Seção 2.3) para implementar serviços tolerantes a faltas bizantinas através de replicação de máquina de estados. A ideia consiste no uso de uma única máquina física, que oferece um serviço replicado e confiável através das máquinas virtuais, as quais tem com única tarefa a execução das requisições que chegam dos clientes do serviço. No caso, é introduzido na arquitetura um domínio denominado NV, sendo este o responsável por efetuar a ordenação das requisições, e portanto, executar o consenso. Por esta razão, são necessárias apenas $2f + 1$ réplicas do serviço, já que elas não executam nenhum consenso bizantino. Deste modo, em decorrência das propriedades inerentes ao *HyperVisor* (cfm. Seção 2.3), o domínio NV passa a ser considerado como um componente inviolável. De outro modo, a despeito da capacidade de tolerar faltas bizantinas nos domínios de aplicação, o domínio NV só é sujeito à faltas por parada.

Outro trabalho particularmente interessante quanto ao uso de

tecnologia de virtualização para prover serviços BFT é o sistema SMIT (*Shared Memory Based Intrusion Tolerance*) (STUMM JR. et al., 2010). O SMIT também é especificado para fins de otimização do custo de replicação BFT, porém, diferente do VM-FIT, ele não realiza o acordo/ordenação de requisições a partir de um domínio isolado, mas por meio de uma abstração de memória compartilhada provida pelo *HyperVisor* para as máquinas virtuais. No caso, as réplicas, que são executadas a partir de $2f+1$ máquinas virtuais, executam um algoritmo de consenso cujas mensagens são trocadas por meio da memória compartilha – a *PostBox* –, o que simplifica em muito a especificação e implementação do algoritmo de consenso bizantino. Como a abordagem adotada pelo SMIT provê um serviço replicado a partir das máquinas virtuais, a corretude do sistema se baseia na propriedade de isolamento do *HyperVisor*. Deste modo, o acesso ao ambiente hospedeiro através das máquinas virtuais é impedido pelo próprio VMM, enquanto que o acesso direto e externo ao ambiente hospedeiro é bloqueado a partir do uso de um mecanismo de *firewall*.

Por fim, embora possa ser considerado como um *position paper*, o trabalho de Chun *et al.* (2008) apresenta uma ampla discussão e encadeamento de ideias sobre o projeto de protocolos tolerantes a faltas bizantinas, através do uso de tecnologia de virtualização. Neste trabalho os autores demonstram uma abordagem que visa a utilização de protocolos de replicação em um único *hardware*, a partir da execução de diversas instâncias do serviço replicado tolerante a faltas bizantinas sobre uma mesma máquina física. Como já dito, o trabalho não apresenta proposições, mas apenas discussões sobre os desafios encontrados à época sobre a especificação de protocolos BFT, e como estes poderiam ser resolvidos com um suporte de virtualização. Um aspecto de particular interesse decorre do fato de os autores terem arguido sobre o escalonamento realizado pelo *HyperVisor* para o processamento das tarefas executadas pelas máquinas virtuais, o qual ocorre de maneira justa para evitar a inanição, e portanto, assegura a progressão (*liveness*) por parte dos ambientes virtuais. O trabalho também discorre sobre os desafios acerca da replicação BFT a partir de um ambiente virtualizado, quanto a diversidade e independência de faltas. Para isso, os autores propõem algumas alternativas baseadas na tecnologia de virtualização, com vista aos aspectos práticos de concepção, no intuito de obter soluções satisfatórias para alguns dos problemas apontados.

5.3.4 Outras Abordagens para Hibridização

Um abordagem bastante interessante para a hibridização do sistema, porém, totalmente diferente das demais apresentadas é aquela baseada na ideia de duas réplicas monitorando uma à outra, conhecida na literatura por *signal-on-failure* (MPOELENG; EZHILCHELVAN; SPEIRS, 2003). A ideia na qual a abordagem *signal-on-failure* é baseada surgiu na literatura a partir da especificação de outra abordagem denominada *fail-stop* (SCHLICHTING; SCHNEIDER, 1983). No entanto, a diferença entre eles é que a implementação da *signal-on-failure* requer apenas um par de processos/réplicas, enquanto que a *fail-stop* demanda por, pelo menos, três processos/réplicas. A abordagem *signal-on-failure* permite implementar processos em sistemas de computação distribuída, partindo do pressuposto que: (*i*) há um subconjunto de processos sujeitos a faltas bizantinas, de modo que cada processo deste conjunto possui um correspondente (i.e., o seu par) para permitir o monitoramento da cada saída por ele(s) produzida(s)[1]; (*ii*) os processos organizados em pares não falham simultaneamente.

A implementação desta abordagem requer, pelo menos, $4f + 2$ máquinas físicas, além de demandar por um canal de comunicação síncrono e confiável entre cada par de réplicas – o que pode ser um pressuposto difícil de se garantir em termos práticos. A despeito do custo verificado, mais precisamente pela incidência de $f + 1$ máquinas a mais do que as usualmente empregadas para as soluções de replicação BFT, os autores justificam o custo computacional pelo custo moderado do *hardware*, verificado na atualidade. A partir da abordagem *signal-on-failure*, Inayat e Ezhilchevan apresentam um protocolo de difusão com ordem total otimista e tolerante a faltas bizantinas (INAYAT; EZHILCHELVAN, 2006). Os autores demonstram que em uma execução livre de falhas, a abordagem tende a apresentar melhor desempenho em relação à outras abordagens utilizadas para replicação BFT (INAYAT; EZHILCHELVAN, 2006). No melhor caso, a latência verificada para a solução implementada pela abordagem demonstra uma redução de três rodadas assíncronas para apenas duas rodadas assíncronas, porém, acrescido de um pequeno atraso para a comparação entre os pares de processos sobre os canais síncronos e confiáveis.

[1] Note que os processos estão organizados em pares, de tal maneira que em cada par de processos um inspeciona/monitora o outro

5.4 CONCEITOS PRELIMINARES

Nesta seção se apresenta os conceitos essenciais para o entendimento da contribuição proposta no presente capítulo desta tese. No mesmo ensejo, são discutidos os aspectos teóricos que amparam e fundamentam a proposição da solução para o problema da validação atômica não-bloqueante com faltas bizantinas. É importante recapitular que a especificação original do NBAC (BABAOGLU; TOUEG, 1993) não admite solução nem mesmo em ambientes sujeitos a mais simples das faltas, que são aquelas por parada (GUERRAOUI, 1995). De outro modo, diferente do que ocorre com o NBAC, o NBWAC (GUERRAOUI, 1995) pode ser resolvido em ambientes sujeitos a faltas por paradas. Neste caso, o NBWAC pode ser reduzido ao consenso, já que os resultados apresentados na literatura para o consenso com detectores de falhas não-confiáveis, também podem ser estendidos ao NBWAC.

Neste sentido, dada as restrições e impossibilidades teóricas verificadas no âmbito do NBAC (GUERRAOUI, 1995), a contribuição apresentada neste capítulo consiste em uma abordagem inédita para resolver o problema NBWAC, isto é, aquele que admite falhas, porém, em um cenário onde alguns dos participantes da transação podem falhar de maneira bizantina. Um aspecto que é digno de nota acerca da contribuição, é que o protocolo proposto neste capítulo tem sua originalidade pautada principalmente pela especificação plena do protocolo, isto é, sem a dependência de um módulo de consenso subjacente, tal como ocorre na maioria das soluções encontradas na literatura, para a resolução do NBAC/NBWAC com faltas por parada (GUERRAOUI; SCHIPER, 1995a; GUERRAOUI; LARREA; SCHIPER, 1996; HURFIN; TRONEL, 1997; ABDALLAH; PUCHERAL, 1999; GREVE; NARZUL, 2002).

Note que os problemas NBAC e NBWAC versam sobre o acordo com vista para a terminação de transação. Neste caso, a contribuição apresentada no presente capítulo não fornece um mecanismo para prover o processamento de transações tolerante a faltas bizantinas, mas sim um protocolo capaz de realizar a validação da transação de maneira segura e confiável, a despeito de faltas bizantinas.

5.4.1 Tecnologia de Virtualização

O uso da tecnologia de virtualização (SMITH; NAIR, 2005) em *hardware commodity* moderno tem se tornado algo bastante comum, atrativo, e também popular ao longo dos últimos anos. A literatura

tem demonstrado que a despeito de declarações de que a tecnologia subjacente já atingiu seus limites (CHUN; MANIATIS; SHENKER, 2008), a tecnologia de virtualização tem tido grande ascensão, principalmente porque ela se mostra bastante útil para a execução racional de aplicações no âmbito da computação verde (HARRIS, 2008). Não obstante, a tecnologia de virtualização também pode ser considerada como o alicerce de grande parte dos ambientes de computação em nuvem (NIST, 2011), a partir da consolidação de serviços e servidores (INTEL, 2013). Aspectos estes, que corroboram para a que tecnologia de virtualização seja encarada como uma alternativa viável quanto à especificação de sistemas de computação.

Do mesmo modo, no contexto de sistemas distribuídos tolerantes a faltas, a tecnologia de virtualização tem se mostrado como um mecanismo bastante atrativo, não apenas para especificar, mas também para implementar soluções computacionais tolerantes a falhas (BRESSOUD; SCHNEIDER, 1995; REISER; KAPITZA, 2007; STUMM JR. et al., 2010; WOOD et al., 2011; DETTONI et al., 2013a). Neste ensejo, a contribuição apresentada no presente capítulo desta tese emprega a tecnologia de virtualização como base para a especificação de uma arquitetura de sistema distribuído. A especificação desta arquitetura se dá com vista para obter a **hibridização em nível de arquitetura** (cfm. Seção 5.3), de modo que seja possível construir um ambiente onde o problema da validação atômica fraca não-bloqueante com participantes que podem falhar de maneira bizantina seja resolvido.

5.4.2 Assinaturas de Limiar

A definição do conceito de assinatura de limiar foi introduzido na literatura por Yvo Desmedt (DESMEDT, 1987). O propósito de um esquema de assinatura (digital) de limiar é permitir que um conjunto de processos utilize uma chave privada (e secreta) compartilhada, de tal maneira que nenhum dos processos, individualmente, tenham o conhecimento daquela chave. Em um esquema de assinaturas de limiar (k, n), qualquer conjunto de processos de tamanho k, dentre um universo de n processos pode efetuar a geração de uma assinatura digital válida, de modo, que qualquer conjunto de processos de tamanho inferior a k é incapaz de produzir uma assinatura válida. No caso, a chave privada é dividida em n chaves compartilhadas, onde cada processo tem a posse de apenas uma chave compartilhada. Para assinar uma mensagem m, cada processo usa sua chave compartilhada para gerar uma assinatura

parcial de m, e então coleta k assinaturas (incluindo a sua) parciais e as combina para produzir uma assinatura de limiar em m.

Note que qualquer processo pode coletar as k assinaturas e combiná-las, com o propósito de gerar uma assinatura de limiar em m. Do mesmo modo, a assinatura de limiar produzida sobre m corresponde a mesma assinatura que seria produzida a partir de uma única entidade que tem o conhecimento da chave privada compartilhada. Em se tratando de ambientes sujeitos a faltas bizantinas, um esquema de assinaturas de limiar é um componente bastante útil, pois, se f é o limite de processos que podem falhar de maneira bizantina, tendo $k \geq f + 1$ para a geração de uma assinatura de limiar em uma mensagem m implica que, pelo menos, um fragmento de assinatura (i.e., assinatura parcial) é gerado por um processo correto, que também atestou a validade do conteúdo daquela mensagem.

Posteriormente à definição do conceito de assinatura de limiar, uma importante propriedade foi adicionada visando prover maior robustez ao esquema, a qual foi dada o nome de **compartilhamento verificável de segredo** (FELDMAN, 1987). Em esquemas de assinaturas de limiar que admitem tal propriedade, a chave compartilhada distribuída para cada processo pode também ser usada para criar uma prova de que uma assinatura parcial foi gerada corretamente. Neste caso, os fragmentos de assinatura gerados incorretamente podem ser descartados antes mesmo de serem combinados, pois se houver um fragmento inválido/incorreto dentre os que foram recolhidos para combinação, o procedimento de combinação tornar-se-á falho e a assinatura produzida será inválida. A partir desta premissa, os fragmentos de assinatura inválidos podem constituir uma prova de que processo que a gerou pode ter sofrido alguma corrupção ou ter manifestado o comportamento bizantino. Deste modo, fragmentos oriundos daquele processo podem ser descartados/ignorados nos procedimentos de combinação subsequentes.

Em um trabalho não tão recente, Shoup (SHOUP, 2000) propôs um esquema de assinaturas de limiar capaz de produzir assinaturas de acordo com o padrão RSA (RIVEST; SHAMIR; ADLEMAN, 1978). Portanto, uma assinatura gerada a partir do esquema de Shoup pode ser verificada pela chave pública correspondente à chave privada dividida e compartilhada, pois a assinatura gerada pelo esquema de limiar é equivalente àquela gerada somente pelo uso da chave privada. Para isso, o esquema que também provê a verificação de segredo, pressupõe que existe um negociador confiável que divide a chave privada e as distribui de maneira segura aos processos.

5.4.3 Detecção de Falhas

A detecção de falhas em sistemas de computação distribuída foi introduzida na literatura por Chandra e Toueg (CHANDRA; TOUEG, 1996), com a finalidade de circunscrever a condição FLP (FISCHER; LYNCH; PATERSON, 1985), a partir do fornecimento de informações acerca do padrão de falhas ocorrido em uma execução do sistema distribuído. Para tanto, os autores especificaram abstrações denominadas **detectores de falhas** ($\mathcal{FD}$) (CHANDRA; TOUEG, 1996), as quais podem ser vistas como oráculos distribuídos, que periodicamente fornecem informações a respeito dos processos suspeitos de terem falhado em uma execução. Formalmente, os detectores de falhas são definidos em termos das propriedades de **completude** e **exatidão** que eles se propõem a exibir. Para tanto, os autores estabeleceram duas propriedades para a **completude** e quatro para a **exatidão**, e a partir da combinação destas classificaram os detectores de falhas em oitos classes, de acordo com as propriedades que eles podem exibir (CHANDRA; TOUEG, 1996).

Uma classe de detectores de grande interesse é classe $\diamond\mathcal{S}$, em que os detectores são conhecidos como *eventually strong* (ou forte terminal), e cujo interesse decorre do fato que é a classe mais fraca e capaz de resolver o problema do consenso (CHANDRA; TOUEG, 1996). Um detector de falhas da classe $\diamond\mathcal{S}$ satisfaz às seguintes propriedades:

1. **Completude Forte (*Strong Completness*):** a partir de algum instante no tempo, cada processo faltoso é permanentemente suspeito por processos corretos;

2. **Exatidão Terminal Fraca (*Eventual Weak Acurracy*):** existe um instante no tempo após o qual, algum processo correto não mais é suspeito por qualquer processo correto.

Note que a propriedade de *completude* visa assegurar que os processos faltosos serão de alguma forma suspeitos, enquanto que a *exatidão* tenta impor um limite quanto aos equívocos que podem ser cometidos pelo detector $\mathcal{FD}$. Dentre as oito classes definidas por Chandra e Toueg (CHANDRA; TOUEG, 1996), os extremos são as classes $\mathcal{P}$ e $\diamond\mathcal{W}$. A classe $\mathcal{P}$ representa os detectores mais fortes, isto é, aqueles que não cometem erros de nenhuma natureza. Já a classe $\diamond\mathcal{W}$ denota os detectores mais fracos, os quais podem cometer erros tanto de completude como de exatidão, o que significa que eles podem levantar suspeitas sobre processos corretos, mas não suspeitar de processos faltosos.

5.4.4 Memória Compartilhada

Um aspecto imperioso quanto a especificação da solução para o NBWAC tolerante a faltas bizantinas, decorre do fato de que o protocolo requer uma abstração de memória compartilhada para a interação entre os agentes presentes em um mesmo sítio, que é parte do protocolo NBWAC e denota um participante da transação distribuída (GUERRAOUI, 1995). Esta abstração de memória compartilhada, conhecida por *PostBox* (STUMM JR. et al., 2010), é um componente em que as escritas ocorrem de maneira atômica e em modo *append-only*. No entanto, diferente da especificação original de tal componente (STUMM JR. et al., 2010), e para os fins requeridos para esta tese, a *PostBox* denota uma espécie de registrador com semântica 1W/nR que modela uma forma de comunicação persistente entre dois ou mais agentes, sendo que um emissor é sempre considerado com o escritor (LAMPORT, 1986). No caso, cada agente presente num sítio possui sua própria *PostBox*, na qual apenas ele pode atuar como escritor e os demais atuam apenas como leitores. Os agentes presentes num sítio desconhecem os aspectos de implementação de tal componente, de modo que este fornece apenas uma interface básica de acesso, na qual duas primitivas estão disponíveis:

- $append(valor)$
- $read() : \{valor, \perp\}$

A primitiva *append* concretiza uma operação de escrita do *valor* fornecido como argumento, de forma atômica, na respectiva *PostBox*. Por se tratar de um componente especificado a partir da semântica *append-only*, o *valor* escrito é armazenado na posição imediatamente posterior àquela alimentada pela invocação da operação de escrita realizada anteriormente. De outro modo, a primitiva *read* retorna o valor imediatamente posterior ao último valor lido daquela *PostBox* para o agente que invocou a operação, ou na ausência de valores disponíveis para leitura naquela posição, ela retorna um valor nulo ($\perp$). Note que, como diversos agentes podem realizar operações de leitura sobre uma mesma *PostBox*, para cada agente é mantido um descritor que indica a sua posição corrente, sendo este descritor incrementado a cada operação de leitura realizada por aquele agente. Em suma, tanto os valores escritos como os lidos de uma *PostBox*, seguem o critério FIFO.

Em referência ao uso de abstrações de memória compartilhada no âmbito da algorítmica distribuída, Guerraoui e Raynal (GUERRAOUI;

RAYNAL, 2006) estabeleceram uma hipótese através da especificação de uma propriedade para o sistema subjacente de memória compartilhada. Esta propriedade visa assegurar a progressão (*liveness*) quanto aos acessos à memória compartilhada, realizados pela execução das operações básicas permitidas.

- **Memória Compartilhada Síncrona Terminal (*Eventually Synchronous Shared Memory*):** a partir de algum instante no tempo, há um limite inferior positivo e um limite superior para um processo executar um passo local, uma leitura ou uma escrita na memória compartilhada.

Mais precisamente, a propriedade em questão estabelece a hipótese de que, em algum momento, um processo para de se portar de maneira completamente assíncrona, a passa a se portar de maneira parcialmente síncrona (DWORK; LYNCH; STOCKMEYER, 1988), de modo a possibilitar o acesso à memória compartilhada dentro de um tempo limitado. Note que os limites de tempo para execução de passos existem, porém são desconhecidos, ou de outro modo, se os limites de tempo são conhecidos, eles somente são cumpridos após um Tempo de Estabilização Global (*Global Stabilization Time* – GST). No contexto da especificação do trabalho do presente capítulo desta tese, também é admitida tal hipótese, no que concerne ao acesso à memória compartilhada por parte dos agentes de um sítio.

Uma nota importante a respeito da especificação desta abstração de memória compartilhada, é que o fato de ela adotar a semântica *append-only* para as escritas, uma vez executada a operação *append*, o valor armazenado não pode ser alterado. Esta restrição tem como finalidade evitar que um agente faltoso que manifesta o comportamento bizantino, escreva um valor e posteriormente altere-o para tentar divergir as ações dos demais agentes. Ademais, como a memória provida por tal abstração tem capacidade finita, para evitar o transbordamento da mesma, quando todos os agentes de um sítio tiverem lido um dado valor, em tese este valor pode ser admitido para descarte. Porém, na implementação proposta, um valor só é elegível para descarte após a conclusão de uma instância do protocolo de acordo. Deste modo, todos os valores usados no decurso daquela instância, são descartados após a constatação de que todos os agentes corretos concluíram suas operações.

Para a formalização deste trabalho, a abstração de memória compartilhada provida pela *PostBox* é especificada em nível de ambiente hospedeiro, em que o VMM/*HyperVisor* fornece-a como um serviço

para as aplicações que executam em nível de convidados, isto é, através do ambiente virtualizado (máquinas virtuais).

5.5 ASPECTOS GERAIS DA SOLUÇÃO PROPOSTA

A proposta do presente capítulo desta tese visa especificar uma solução algorítmica para o problema da Validação Atômica Fraca Não-bloqueante (NBWAC), em um ambiente sujeito à faltas bizantinas. Todavia, já fora demonstrado pela literatura, que o modelo de faltas bizantinas não admite solução para nenhum problema pertencente à classe de problemas acordo, especificados pela propriedade de **acordo uniforme** – que é o caso do NBAC e do NBWAC (CHARRON-BOST; SCHIPER, 2004). Neste sentido, o que se busca em primeira instância, é verificar quais as condições necessárias para resolver o NBWAC num cenário com faltas bizantinas. Para tanto, se tem como objetivo principal descrever a especificação de uma arquitetura de sistema distribuído, baseada em virtualização, que tome proveito de recursos/características oferecidos pela tecnologia de virtualização, para possibilitar a construção de um protocolo capaz de resolver o problema NBWAC em cenários com sujeição a faltas bizantinas.

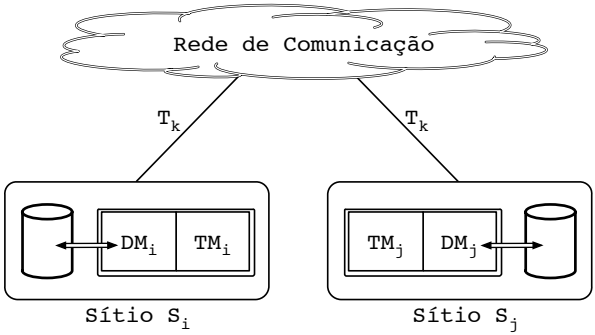


Figura 40 – Modelo típico de cenário de implementação de um protocolo NBAC.

Uma inspeção sobre as especificações de trabalhos clássicos da literatura desenvolvidos no âmbito no NBAC (GRAY, 1978; BERNSTEIN; HADZILACOS; GOODMAN, 1987; GRAY; REUTER, 1992; BABAOGLU; TOUEG, 1993), demonstra que um cenário recorrente em termos

da implementação de um protocolo que resolve tal problema é aquele ilustrado pela Figura 40, em que um sítio que participa da transação, tipicamente atua em dois papéis durante a execução da transação, o de DM/RM e o de TM. Nos termos das especificações clássicas definidas para o problema NBAC (BERNSTEIN; HADZILACOS; GOODMAN, 1987; GRAY; REUTER, 1992; BABAOGLU; TOUEG, 1993), o DM provê a interface de acesso ao banco de dados local do sítio, enquanto que o TM atua como uma espécie de monitor quanto ao processamento de transações locais (GRAY; REUTER, 1992), e também opera em conjunto com os TMs dos demais sítios para o processamento da transação distribuída (p. ex.: a transação T_k da Figura 40). Embora os papéis destes processos sejam complementares, com relação ao acordo requerido para a terminação da transação – que compreende ao cerne do problema NBAC (BABAOGLU; TOUEG, 1993) – esta é uma tarefa delegada aos TMs de cada sítio. Neste caso, devido às atribuições inerentes aos papéis de cada um deles, o TM pode ser definido como um agente **ativo** do protocolo de terminação, enquanto que o DM pode ser considerado como um agente **passivo** na etapa que compreende ao passo de terminação da transação.

E por assim dizer, a partir deste pressuposto verificado quanto à especificação do NBAC, é que a solução proposta por esta tese é inspirada em termos de sua especificação e implementação. Para um melhor entendimento a respeito da especificação da solução proposta, em vez de assumir o modelo tradicional que denota um participante do NBAC, o TM e o DM/RM passam a ser considerados como agentes de um sítio que participa da transação. A taxonomia definida no âmbito desta tese, em termos dos agentes que participam do NBAC em **ativos** e **passivos**, é lícita por duas razões, a primeira porque que é através do TM que um sítio executa o protocolo de terminação da transação; e a segunda é porque o voto de um sítio depende das condições locais de execução da transação, o que só pode ser determinado pelo DM. À vista disso, durante a execução de um protocolo NBAC, o TM deve consultar o DM quanto à declaração do voto daquele sítio, para a realização do acordo com os demais sítios envolvidos no protocolo. Neste sentido, diferente de todas as propostas de algoritmos para o NBAC tradicionalmente encontradas na literatura (GUERRAOUI; LARREA; SCHIPER, 1995; GUERRAOUI; SCHIPER, 1995a; GUERRAOUI, 1995; GUERRAOUI; LARREA; SCHIPER, 1996; HURFIN; TRONEL, 1997; ABDALLAH; PUCHERAL, 1999; GREVE; NARZUL, 2002; PARK; LEE; YU, 2013), o que se propõe no âmbito desta tese é uma **separação** dos agentes **ativo** e **passivo**, de tal maneira que eles sejam executados isoladamente um do

outro, conforme o modelo ilustrado pela Figura 41.

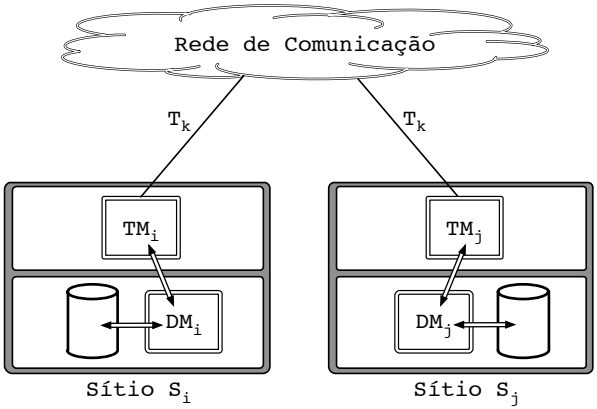


Figura 41 – Modelo de cenário para implementação da solução NBWAC proposta.

É importante notar que o modelo proposto apresenta uma nova abordagem para representar os sítios que participam de um protocolo NBWAC, sendo esta representação definida em consonância com a especificação tradicional/clássica do NBAC. Note que cada sítio permanece a abrigar os processos/agentes que atuam nos papeis de TM e DM/RM, porém, a partir de especificações e implementações distintas. Não obstante, em face da impossibilidade de resolução do NBAC/NBWAC em ambientes puramente bizantinos, esta separação pode ser vista como um meio factível para se prover os requisitos e as condições necessárias para assegurar as propriedades do NBWAC, a despeito de faltas dos sítios/participantes. Isto posto, para que seja possível resolver o problema da NBWAC num ambiente com faltas bizantinas, esta tese recorre à hibridização do modelo, a fim de compatibilizar a especificação do protocolo com o cenário desejado. A partir dai, para realizar a especificação da solução proposta, duas hipóteses são presumidas: (*i*) os agentes ativos (i.e., os TMs) são os que conduzem o protocolo de terminação da transação nos moldes do NBWAC, e; (*ii*) os agentes passivos, representados pelos DMs, são envolvidos no protocolo apenas quando solicitado pelos TMs dos respectivos sítios.

Um aspecto de capital relevância decorre do fato de que a terminação consiste em um dos passos mais críticos quanto ao gerenciamento de uma transação distribuída, pois a partir da terminação é que

o(s) resultado(s) da(s) operação(ões) passa(m) da condição de transiente(s) para permanente(s) no sistema, a fim de tornar a transação válida. Deste modo, torna-se lícito presumir um modelo de falhas híbrido para a especificação de um protocolo de validação atômica fraca não-bloqueante, modelo este em que os TMs, que denotam os agentes ativos do protocolo, e portanto, que conduzem a terminação possam falhar de maneira arbitrária/bizantina, enquanto que os DMs (i.e., agentes passivos) possam falhar apenas por parada. Entretanto, a separação dos agentes do sítio por meio de especificações/implementações distintas não impede um agente TM bizantino de sempre declarar seu voto como não, a fim de deturpar o protocolo e impedir que nenhuma transação consiga sua validação – uma exigência das propriedades **Validade Uniforme** e **Não-Trivialidade** definidas para o NBAC/NBWAC (vide Seção 2.2.3.2).

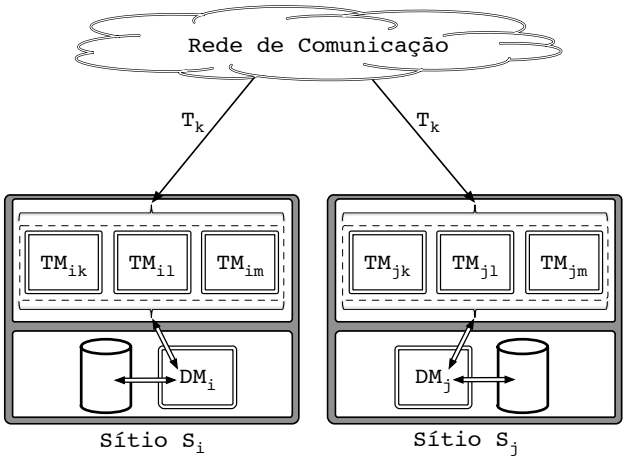


Figura 42 – Modelo de cenário para implementação da solução NBWAC pelo conceito de CTM.

Em face ao problema apontado no tocante a um TM que exibe faltas bizantinas, esta tese introduz o conceito de CTM (do inglês, *Collaborative Transaction Manager* – Gerenciador de Transações Colaborativo). O CTM consiste em uma entidade abstrata residente em cada sítio participante da transação, sendo concretizada a partir de um conjunto de réplicas de TM em cada sítio – conforme ilustrado pela Figura 42, através do retângulo pontilhado. As atividades do CTM ocorrem através da atuação conjunta e colaborativa das réplicas de TM de um

sítio, com o propósito de atestar/certificas as ações realizadas no âmbito daquele sítio. Do mesmo modo, o CTM pode ser visto como uma espécie de agente que atua de maneira pró-ativa em relação à detecção de faltas bizantinas ocorridas nos agentes TMs daquele sítio, ao mesmo tempo que permite prover um meio factível e eficiente para mascarar a(s) falta(s) bizantina(s) ocorridas durante a execução do protocolo de terminação baseado no NBWAC.

A introdução do CTM, se dá com vista para evitar que o comportamento arbitrário/bizantino exibido de maneira unilateral por um TM faltoso, cause perturbação à execução do protocolo de terminação da transação. Como exemplo para tal afirmação, considere a execução da fase de preparação no protocolo 3PC (SKEEN, 1981); caso um TM manifeste comportamento bizantino, ele pode simplesmente indicar/votar `não` de maneira ilegítima, apenas com a finalidade de deturpar o acordo acerca da terminação da transação – i.e., para invalidá-la. Em tal situação, nenhuma transação terá êxito na consecução de sua validação, uma vez que um voto `não` implica na anulação da mesma. Note que não se trata de apenas uma particularidade do protocolo 3PC, mas de qualquer protocolo que atende às propriedades especificadas para o NBAC (BABAOGLU; TOUEG, 1993).

Para facilitar a compreensão em referência a ideia da abordagem empregada para proposição da solução para o NBWAC no cenário com faltas bizantinas, a Figura 43 ilustra de maneira mais detalhada, o modelo especificado para a arquitetura de sistema distribuído, com vista a prover os requisitos necessários à resolução da validação atômica fraca não-bloqueante com faltas bizantinas. Conforme já mencionado, a especificação da arquitetura recorre ao uso de tecnologia de virtualização, mais precisamente a partir de uma abordagem de nossa autoria denominada por **máquinas virtuais gêmeas** (DETTONI et al., 2013a, 2013b). Por meio desta abordagem, a especificação de um sítio participante do sistema passa a ser modelada a partir de uma máquina física, em que dois ambientes são permitidos: o anfitrião, que opera diretamente sobre nível físico; e o virtual, também conhecido na literatura por convidado (vide Seção 2.3), que é constituído pelo conjunto de máquinas virtuais presentes na especificação da arquitetura.

A arquitetura de sistema proposta recorre à tecnologia de virtualização em sua especificação, pela razão de que a virtualização permite encapsular e isolar falhas em níveis distintos da arquitetura (BRESSOUD; SCHNEIDER, 1995); ao passo que a separação dos componentes de um sistema, em máquinas virtuais isoladas, reduz substancialmente o impacto decorrente de falhas verificadas em parte dos componentes

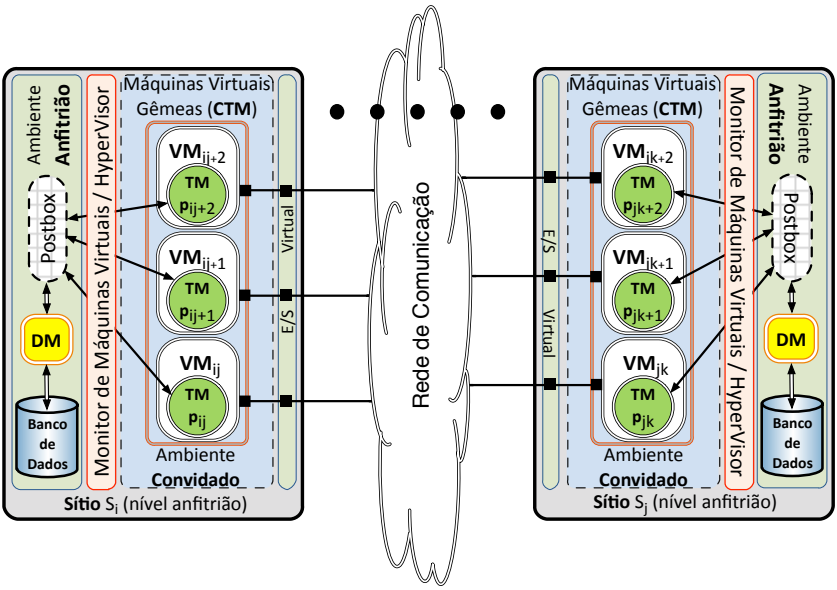


Figura 43 – Detalhamento do modelo de arquitetura proposta para resolver o NBWAC com faltas bizantinas.

sobre o restante do sistema (LEVASSEUR et al., 2004). Com base nesta concepção, é possível notar que a separação dos agentes TM e DM ocorrem em níveis distintos da arquitetura proposta, de acordo com a Figura 43. No caso, as réplicas que formam o CTM são executadas por um conjunto de máquinas virtuais, enquanto que o sistema anfitrião é usado para implementar o agente passivo do protocolo, o DM. Ademais, como a especificação presume que apenas o CTM é quem executa o protocolo de acordo para a terminação da transação (i.e., o NBWAC), tal abordagem previne que o comportamento bizantino exibido por um participante (i.e., TM) faltoso do protocolo afete a execução do mesmo, de modo que apenas as faltas de parada dos sítios sejam percebidas no âmbito do NBWAC. Assim, a especificação de um algoritmo que resolve o NBWAC em um cenário com faltas bizantinas torna-se muito próxima/semelhante daqueles que toleram apenas faltas por paradas. A formalização do protocolo proposto, através de seus algoritmos será apresenta na seção seguinte.

Note que a partir da atuação de um CTM em cada sítio, o comportamento de falha exibido por um TM comprometido fica confinado

ao seu espaço local, e, se manifestado externamente, torna-se facilmente detectável, não apenas em seu contexto de atuação, mas também pelos demais sítios participantes do protocolo. Em termos práticos, a atividade do CTM ocorre pela atuação colaborativa e conjunta dos TMs de um mesmo sítio, e com a ajuda de um esquema de assinaturas de limiar (SHOUP, 2000), com o propósito de assegurar não apenas a integridade das mensagens, mas também a autenticidade do CTM de um sítio emissor. Portanto, sempre que houver a necessidade de interação/cooperação de um sítio com os demais, a mensagem que for enviada deve ser atestada pelo CTM daquele sítio, o que significa que, pelo menos, $f + 1$ réplicas de TM do sítio emissor devem assinar parcialmente a mensagem, para posteriormente produzir uma assinatura de limiar válida para aquela mensagem. E se porventura um agente TM de um sítio tentar deturpar o protocolo, ele o fará isoladamente de maneira unilateral, e por isso será detectado e desconsiderado pelos CTMs dos demais sítios envolvidos no protocolo.

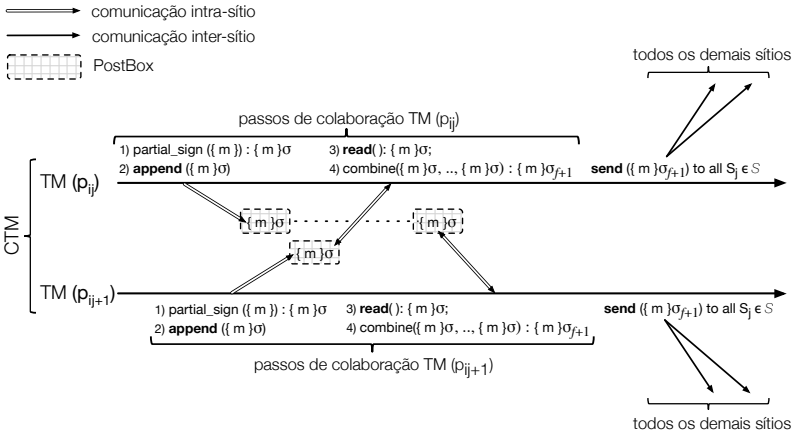


Figura 44 – Colaboração entre os TMs de um mesmo sítio (CTM).

Para exemplificar a atuação do CTM em um sítio nos termos da Figura 43, note que cada TM de um sítio S_i é especificado e implementado a partir de um processo p_{ij}. Cada processo p_{ij}, tal que $1 \leq j \leq |\mathcal{P}|$ ($\mathcal{P}$ denota o conjunto dos processos TM de um sítio), executa os passos do protocolo como um agente TM do NBAC, a fim de colaborar com sua parte na atuação do CTM. A atuação do CTM pela colaboração dos TMs de um sítio é ilustrada na Figura 44, e se dá da seguinte maneira,

(*i*) a cada mensagem recebida através da rede de comunicação, o TM executado pelo processo p_{ij} assina a mensagem com a parte/segredo da chave privada que lhe cabe, e insere a mensagem assinada em sua *PostBox*, e; (*ii*) verifica através das *PostBoxes* dos demais TMs executados pelos processos p_{ik} (tal que $k \neq j$), as mensagens assinadas e compartilhadas por aqueles processos, até obter, pelo menos, $f + 1$ fragmentos de assinatura válidos para a mesma mensagem; (*iii*) após obter o número de fragmentos suficientes, cada TM os combina e gera uma assinatura de limiar para aquela mensagem, cuja validade será atestada pela chave pública do sítio S_i – a qual é de conhecimento de todos os sítios S_j (tal que $j \neq i$). Note que para a mensagem ser considerada válida, os fragmentos de assinaturas gerados pelos TMs devem ser mutuamente consistentes. Este processo pode ser visto como uma espécie de endosso com relação às ações realizadas por parte dos TMs corretos de um sítio, de modo a atestar cada mensagem enviada por aquele sítio.

Por fim, um aspecto que também é digno de nota a respeito da especificação da arquitetura de suporte à execução do NBWAC, é que ela dispõe de um mecanismo para prover a confiabilidade na recepção de mensagens enviadas entre os sítios, o qual é baseado no conceito de interceptador, nos mesmos moldes do trabalho de Narasimhan *et al.* (1997). Este mecanismo é executado no ambiente hospedeiro de cada sítio, como uma aplicação em nível de usuário (i.e., não em nível de núcleo), e sua funcionalidade única é interceptar as mensagens endereçadas aos TMs que chegam através das interfaces virtuais de rede, e encaminhá-las a todos os TMs daquele sítio como uma espécie de difusão confiável. Na realidade não se trata de um protocolo de difusão confiável, mas apenas de um mecanismo para assegurar que uma mensagem destinada a um TM de um sítio será recebida por todas as réplicas de TM daquele sítio, a fim de evitar problemas que possam causar divergências/desacordos quanto à atuação do CTM. Note que uma eventual falha de parada em um sítio que acabara de enviar uma mensagem, pode acarretar em problemas na recepção de mensagens por parte dos TMs de um sítio, o que comprometeria a atuação do CTM.

5.6 ESPECIFICAÇÃO DO PROTOCOLO

Esta seção descreve a integração da arquitetura apresentada na Seção 5.5 (vide Figura 43) e sua relação com a proposição do protocolo que resolve o problema da NBWAC com um modelo híbrido

de faltas, onde é permitido que parte dos TMs sejam sujeitos a faltas arbitrárias/bizantinas. Doravante, o protocolo proposto passa a ser referenciado por HB-NBWAC – acrônimo para *Hybrid/Byzantine Non-Blocking Weak Atomic Commitment* (i.e., validação atômica fraca não-bloqueante com faltas híbridas/bizantinas).

5.6.1 Modelo de Sistema

A arquitetura da Figura 43 é especificada a partir de um modelo híbrido, onde são admitidas diferentes hipóteses acerca do modelo de falhas das entidades que a compõem. O sistema é composto por um universo de processos $\mathcal{U}$, sendo este dividido em alguns subconjuntos. O subconjunto $\mathcal{S} = \{s_1, s_2, ..., s_n\}$ denota os sítios que compõem o sistema distribuído. Cada sítio s_i é composto por um sistema **anfitrião** (*host*), sobre o qual é executado um Monitor de Máquinas Virtuais (ou *Virtual Machine Monitor* – VMM). Para cada sítio s_i, o VMM mantém um conjunto de **máquinas virtuais gêmeas**, que são também conhecidas como sistemas **convidados** (*guests*), tal que $\forall s_i \in \mathcal{S} : \exists\ \mathcal{G}_{s_i} = \{g_1, g_2, ..., g_n\}$. O subconjunto $\mathcal{S}$ tem cardinalidade superior a f_s, logo $|\mathcal{S}| \geq f_s + 1$; e os subconjuntos $\mathcal{G}_{s_i}$ têm cardinalidade $\forall s_i \in \mathcal{S} : |\mathcal{G}_{s_i}| \geq 2f_v + 1$. Todavia, como o modelo assume o uso de um detector de falhas da classe $\diamond\mathcal{S}$ como mecanismo subjacente para a resolução do acordo, este requer uma maioria dos sítios corretos $\lceil (n+1)/2 \rceil$ (CHANDRA; TOUEG, 1996), o que, portanto, impõe que a cardinalidade de $|\mathcal{S}| \geq 2f_s + 1$. Os termos f_s e f_v são utilizados para denotar os limites de faltas dos sítios e das máquinas virtuais presentes em cada sítio, respectivamente.

Cada sistema **anfitrião** mantém uma base de dados, a qual é utilizada nas transações, sendo também o responsável por executar o processo que atua como gestor de dados (DM) no protocolo HB-NBWAC. Não obstante, o anfitrião mantém um monitor de máquinas virtuais, sobre o qual são executadas como sistemas **convidados** as máquinas virtuais gêmeas (DETTONI et al., 2013a). Cada máquina virtual gêmea executa apenas uma réplica do processo que atua no papel de gestor da transação (TM) daquele sítio, de modo que a atuação destas réplicas se dá de maneira conjunta, para representação destas através do CTM em cada sítio. Devido a separação das entidades em níveis distintos da arquitetura, é assumido que: (i) até $f_v \leq \lceil \frac{n-1}{2} \rceil$ máquinas virtuais gêmeas, **por sistema anfitrião** (no caso, os TMs), podem desviar arbitrariamente de suas especificações e então falhar de maneira ar-

bitrária/bizantina, nos termos do modelo bizantino com autenticação (DOLEV; STRONG, 1983); (*ii*) o sistema anfitrião pode falhar apenas por parada (*crash*), sendo que não mais de f_s anfitriões (i.e., sítios) falham simultaneamente durante uma janela de vulnerabilidade do sistema. Além disso, embora as réplicas de TM executadas num mesmo sítio operem de maneira assíncrona, é presumido que os desvios entre as corretas são limitados, no que concerne as suas velocidades relativas – uma premissa bastante razoável, pois, uma vez que elas são baseadas em um mesmo relógio físico, as derivas são pequenas e limitadas (VMWARE, 2011).

Também é presumido que o sistema anfitrião pode apresentar vulnerabilidades, porém, o VMM fornece isolamento entre os sistemas anfitrião e convidado(s), de tal forma que vulnerabilidades não podem ser exploradas através das máquinas virtuais, nem entre máquinas virtuais (POPEK; GOLDBERG, 1974). Embora o termo **máquinas virtuais gêmeas** soe de aparência problemática tendo em consideração um ambiente sujeito a faltas bizantinas, o modelo proposto presume que há a diversidade em termos de software (AVIZIENIS et al., 2004). Neste sentido, as réplicas de TM podem ser construídas/implementada sobre sistemas operacionais distintos, e também através de diferentes ambientes de programação, ambos com vista para evitar que eles compartilhem os mesmos *bugs* ou vulnerabilidades em termos software. Estas duas premissas reforçam a verificação da independência de faltas (OBELHEIRO; BESSANI; LUNG, 2005) no âmbito do protocolo proposto, de modo que a ocorrência de uma falta sobre uma das réplicas de TM não implica na ocorrência da mesma falta em outra réplica de TM, o que, portanto, não afeta a execução/atuação do CTM daquele sítio.

É assumida a existência de um esquema criptográfico de assinaturas de limiar (n, k) (SHOUP, 2000) baseado no algoritmo RSA (RIVEST; SHAMIR; ADLEMAN, 1978), onde existem n chaves parciais (ou segredos) e n chaves de verificação, de tal forma que pela combinação de, pelo menos, $k = f_v + 1$ segredos é possível gerar uma assinatura RSA, verificável a partir da chave pública correspondente, do sítio que assinou a mensagem. Também é assumido que o sistema anfitrião – que não é suscetível a faltas bizantinas, atua como distribuidor dos segredos às réplicas de TMs daquele sítio, sendo que ele gera uma chave privada, a qual fica em sua posse e é dividida em n segredos que são distribuídos a cada uma das máquinas gêmeas, e uma chave pública. A chave pública é usada para verificar a autenticidade e integridade das mensagens assinadas, tanto por meio da chave privada usada pelo sistema anfitrião (p. ex. pelo DM), como daquelas geradas a partir do

esquema de limiar.

A respeito das hipóteses temporais, não há nenhuma restrição temporal no que se refere as ações executadas pelos processos, de modo que eles as realizam em conformidade com suas respectivas velocidades. Não obstante, dada a impossibilidade de resolução do acordo em ambientes onde não há restrições temporais (FISCHER; LYNCH; PATERSON, 1985), o sistema se baseia no uso de abstrações de detectores de falhas da classe $\diamond\mathcal{S}$, o qual é utilizado apenas para estimar as possíveis falhas por **parada** que tenham ocorrido em nível de sítio. Os relógios dos sítios não são sincronizados, embora os desvios entre eles sejam pequenos – uma suposição válida, considerando as tecnologias atuais. O sistema anfitrião fornece abstrações de memória compartilhada para comunicação entre os processos de um sítio, estas denominadas por *PostBox* (STUMM JR. et al., 2010), em que as escritas acontecem apenas em modo *append-only* e as leituras seguem a ordem *FIFO*. Um processo pode escrever apenas em sua *PostBox*, tendo somente o direito de leitura nas *PostBoxes* dos demais processos do mesmo sítio. A comunicação entre os processos é realizada de duas maneiras: (*i*) entre as máquinas virtuais gêmeas de um sítio (*intra-sítio*); e (*ii*) entre os sítios (*inter-sítio*). As comunicações *inter-sítio* ocorrem via canais ponto-a-ponto confiáveis e autenticados, e a comunicação *intra-sítio* ocorre através das *PostBoxes*.

5.6.2 Princípio de Funcionamento do HB-NBWAC

Em conformidade com a literatura, a especificação dos problemas da NBAC e NBWAC (BABAOGLU; TOUEG, 1993; GUERRAOUI, 1995) frisam a terminação, e não a execução a transação. Logo, o protocolo apresentado nesta seção é um protocolo para a terminação de transações distribuídas. Por esta razão, apenas presume-se a existência de um suporte de nível superior para a execução da transação, de tal forma que a terminação da transação é realizada por meio do protocolo HB-NBWAC. A Figura 45 ilustra as fases e os passos de comunicação realizados pelo protocolo HB-NBWAC, onde num cenário mais favorável, o problema é resolvido a partir de três fases e quatro passos de comunicação *inter-sítio*. Note que toda comunicação *inter-sítio* é precedida por uma comunicação *intra-sítio*, sendo que esta última ocorre devido à atuação do CTM no protocolo a partir das *PostBoxes*, isto é, pela colaboração entre as réplicas de TM de um sítio.

No caso da Figura 45, o sítio S_1 atua como o líder da transação

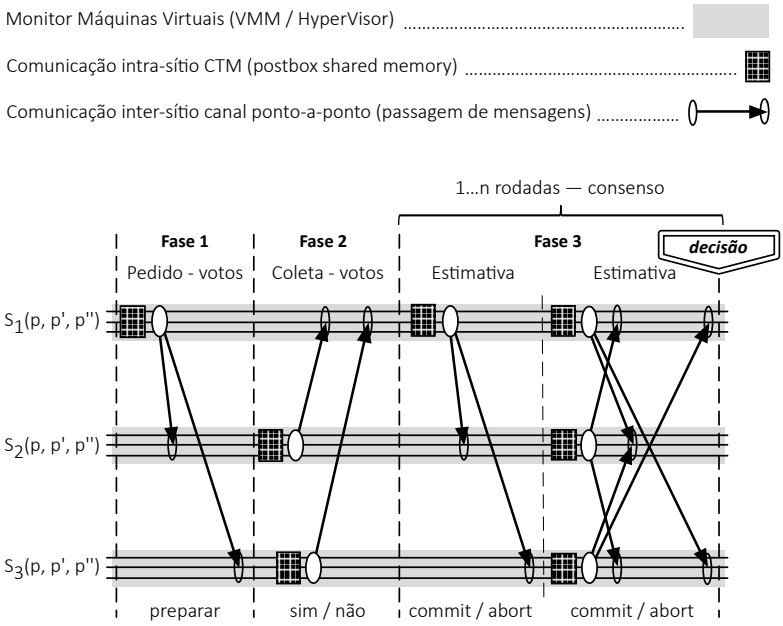


Figura 45 – Diagrama de execução e fases do HB-NBWAC.

e é o responsável por conduzir a primeira rodada do protocolo de terminação. Na mesma figura, $f_v = 1$, e por isso cada sítio mantém $2f_v + 1$ réplicas de TM – ali ilustrados por p, p' e p''. Na primeira fase, o líder solicita uma declaração de voto aos demais participantes, no intuito de verificar se todos estão preparados para proceder com a terminação daquela transação, e portanto, validar as operações realizadas naquela transação. Na fase seguinte, cada sítio não-líder vota de acordo com a sua condição local, a partir da qual é declarado seu voto que é então enviado ao sítio líder. O voto de um sítio é: `SIM`, se ele está pronto para validar as atualizações; ou `NÃO`, caso não esteja pronto ou suspeite que houve falha do líder. Ainda nesta fase, o líder coleta os votos recebidos, avança para a fase três e calcula uma estimativa de resultado para a transação, o que é realizado a partir das declarações recebidas dos sítios incluindo a sua. Determinada a estimativa, o protocolo entra na fase de decisão, a qual é realizada a partir de um algoritmo de consenso baseado no mesmo princípio do algoritmo de Mostéfaoui e Raynal (1999). Durante a fase de decisão, com vista para assegurar a concordância uniforme por parte de todos os sítios participantes da transação, quando

apropriado e necessário a realização do consenso ocorre através de sucessivas rodadas, sendo que para cada rodada é adotado o paradigma do coordenador rotativo.

No que concerne a resolução do NBWAC através da solução proposta, todas as fases apresentadas pela Figura 45 são necessárias para respeitar as condições e propriedades especificadas para o problema (vide Seção 2.2.3.2), de modo que uma decisão pela validação é legítima apenas se **todos** os sítios votaram `SIM`. E para não permitir equívocos no que se refere a tomada de decisão, o protocolo se orienta da seguinte maneira: (*i*) na fase 1, se um sítio não puder validar a transação devido às suas condições locais ou se não tiver recebido o pedido de voto do líder devido à uma suspeita de falha daquele sítio, ele vota `NÃO`; do contrário, se ambas as condições não forem verificadas ele vota `SIM`; (*ii*) na fase 2, se porventura o líder não tiver recebido **todos** os votos dos sítios não-líderes, ele suspeita que houve falha em algum sítio e por isso calcula a estimativa de decisão como `abort`; ou do contrário, a estimativa de decisão será `commit`; (*ii*) na fase 3, o resultado obtido para a decisão será subsidiado, em primeira instância, pela estimativa recebida do líder, onde o protocolo decide por meio de rodadas até obter uma concordância uniforme por parte de todos os sítios não faltosos. Note que, para cada rodada de decisão, um novo sítio assume o posto de líder. Isto posto, se durante alguma rodadas algum sítio não receba a estimativa e suspeite de falha no líder daquela rodada, ele determinará sua estimativa local como `abort`, o que incorrerá na anulação da transação pelo protocolo. À vista disso, uma decisão pela validação só poderá ocorrer na primeira rodada de decisão e apenas se todos os sítios votaram `SIM`, e tanto nas fases predecessoras com na fase de decisão, não houve suspeita(s) de falha(s).

5.6.3 Base Algorítmica do Protocolo HB-NBWAC

Esta seção apresenta os detalhes quanto à especificação algorítmica do protocolo HB-NBWAC, o que é realizado a partir dos Algoritmos 5, 6 e 7. É importante ressaltar que os algoritmos em questão formalizam apenas a especificação dos agentes **ativos** do HB-NBWAC, as quais denotam os TMs e que compreendem as entidades responsáveis pela execução do protocolo de terminação (BABAOGLU; TOUEG, 1993) – e realizam a representação do CTM do sítio. À vista disso, pela simplicidade de especificação do agente **passivo** denotado no protocolo pelo DM, este é apenas consultado pelos TMs para verificar se as

operações da transação foram completamente executadas, e sem erros naquele sítio.

A apresentação do protocolo é iniciada pelo Algoritmo 5, o qual descreve a declaração das variáveis utilizadas em todos os algoritmos, a inicialização das estruturas de dados e também duas funções cruciais para a concretização do protocolo. Note que o protocolo opera a partir de um conjunto de sítios ($\mathcal{S}$ – linha 1), e cada sítio é composto por um processo que implementa a entidade DM e por um conjunto de $2f_v + 1$ réplicas (ou máquinas virtuais gêmeas) que representam as entidades TMs do protocolo ($\mathcal{P}$ – linha 2), e que, portanto, dão origem ao CTM daquele sítio. A comunicação *intra-sítio* entre as réplicas de TMs e também com o DM ocorre por meio da abstração de memória compartilhada denominada *PostBox*, que é provida pelo sistema anfitrião de um sítio, para cada uma das entidades envolvida no protocolo, e neste caso, uma por processo/réplica de máquina virtual gêmea (TM) e uma para o DM. O código entre as linhas 10 e 12 compreende à especificação da função *endorse_ message*, usada para atestar as ações a serem exibidas para os demais sítios através das mensagens produzida por um dado sítio. Esta primeira função denominada por *endorse_ message*, é referenciada nos Algoritmos 6 e 7, onde ela é chamada sempre que há a necessidade de se enviar uma mensagem para os demais sítios participantes do protocolo de terminação da transação, nas fases em que ocorrem a evolução do protocolo.

Neste sentido, esta função retorna para uma variável, a mesma estrutura de dados que foi fornecida como argumento, e que representa a mensagem que se pretende validar/atestar via esta função – a variável $\langle message \rangle$ no caso dos Algoritmos 6 e 7. Esta função tem o único propósito de simplificar o processo de validação das mensagens produzidas pelos TMs de um sítio, em que a especificação (linhas 10 a 12) realiza uma espécie de endosso da mensagem produzida por um TM para os outros TMs do mesmo sítio – o que constitui um processo essencial para a atuação do CTM dos sítios, a fim de prover a validação das ações ocorridas nos sítios por meio das mensagens de saída dos mesmos. Ela efetua basicamente duas tarefas, a primeira é a assinatura parcial da mensagem fornecida como argumento para ela, com a parte do segredo do processo TM que a invocou (linha 10), em que o termo σ denota a assinatura parcial efetuada sobre a mensagem. Em seguida é feita uma chamada para a função *intrasite_ validate* (linha 11) – especificada nas linhas 13 – 35 do mesmo algoritmo, que é onde ocorre de fato, a interação com os demais TMs do mesmo sítio (via comunicação *intra-sítio*) para a coleta das demais assinaturas parciais para a mesma

Algoritmo 5 Especificações para o protocolo – sítio S_i – processo p_j

Variáveis:

(1) $\mathcal{S} = \{S_1, S_2, ..., S_n\}$ /* Sítios que participam das transações distribuídas */
(2) $\mathcal{P}[\ ] = \varnothing$ /* Processos TM gêmeos de cada sítio S_i - um arranjo */
(3) $\mathcal{H}[\ ]\langle\rangle = \varnothing$ /* Buffer de mensagens oriundas da postbox - *hashtable* */
(4) $\texttt{outcome}\langle\rangle = \varnothing$ /* Estrutura que contém a decisão de cada transação - *hashtable* */
(5) $vote : \{yes, no\}$ /* Únicos votos possíveis para declaração pelos sítios */
(6) $decision : \{commit, abort\}$ /* Resultados possíveis para a terminação do protocolo */
(7) $estimate : \{commit, abort\}$ /* Estimativas possíveis para uma decisão */

Inicialização:

(8) $\forall S_j \in \mathcal{S}, \mathcal{P}[S_j] := \{p_1, p_2, ..., p_n\}$ /* $n = 2f_v + 1$ */
(9) $\forall p_k \in \mathcal{P}[S_i], 1 \leq k \leq |\mathcal{P}|, \mathcal{H}[p_k]\langle\rangle := \perp$

function $endorse\_message(\langle message\rangle)$ /* assinatura parcial da mensagem */
(10) $\langle message\rangle_\sigma := partial\_sign(p_j, \langle message\rangle)$
(11) $\langle message\rangle_{\sigma_{f+1}} := intrasite\_validate(\langle message\rangle_\sigma)$
(12) **return** $\langle message\rangle_{\sigma_{f+1}}$

function $intrasite\_validate(\langle message\rangle_\sigma)$ /* comunicação intra-sítio / validação da mensagem */
(13) $postbox[p_j].append(\langle message\rangle_\sigma)$
(14) $\forall p_k \in \mathcal{P}[S_i], 1 \leq k \leq |\mathcal{P}|, buffer[p_k] := \varnothing$
(15) $certified \leftarrow \varnothing$
(16) $read \leftarrow \varnothing$
/* 1º verifica se as mensagens parcialmente assinadas já foram recuperada das *PostBoxes* */
(17) **for each** $p_k \in \mathcal{P}[S_i]$ **do**
(18) **if** $\mathcal{H}[p_k]\langle message.key\rangle \neq \perp$ **then**
(19) $buffer[p_k] := \mathcal{H}[p_k]\langle message.key\rangle$
(20) **for each** $msg \in buffer[p_k] : msg.type = message.type$ **do**
(21) **if** $\nexists p_k \in read$ **and** $verifypart(p_k, \langle msg\rangle_\sigma)$ **then**
(22) $certified := certified \cup \{\langle msg\rangle_\sigma\}$
(23) $read := read \cup \{p_k\}$
(24) $\mathcal{H}[p_k]\langle message.key\rangle := buffer[p_k] \setminus certified$
/* ainda não há, pelo menos, $f_v + 1$ assinaturas parciais p/ a mensagem, verifica as *PostBoxes* até obtê-las */
(25) **while** ($\nexists$ at least $f_v + 1$ matching $msg \in certified$) **do**
(26) **for each** $p_k \in (\mathcal{P}[S_i] \setminus read)$ **do**
(27) $msg := postbox[p_k].read()$
(28) **if** $msg \neq \perp$ **and** $verifypart(p_k, \langle msg\rangle_\sigma)$ **then**
(29) **if** $msg.type = message.type$ **and** $msg.key = message.key$ **then**
(30) $certified := certified \cup \{\langle msg\rangle_\sigma\}$
(31) $read := read \cup \{p_k\}$
(32) **else**
(33) $\mathcal{H}[p_k]\langle msg.key\rangle := \mathcal{H}[p_k]\langle msg.key\rangle \cup \{msg\}$
(34) $\sigma \leftarrow combine(certified)$ /* combina as mensagens/assinaturas parciais em uma assinatura de limiar */
(35) **return** $\langle message\rangle_{\sigma_{f+1}}$

mensagem e posteriormente a combinação destas em uma única assinatura de limiar válida. Note que o retorno da função *intrasite_ validate* (linha 11) é a mensagem fornecida como argumento, assinada por, pelo menos, $f + 1$ TMs distintos do mesmo sítio (o termo σ_{f+1} é usado para representar a assinatura de limiar da mensagem).

O código entre as linhas 13 e 35 diz respeito à implementação da função *intrasite_ validate*, que é onde ocorrem as comunicações *intra-*

sítio entre as máquinas virtuais gêmeas (ou TMs) de um sítio, para a validação das mensagens do protocolo antes do envio aos outros sítios participantes (*inter-sítio*), pelo canal que o liga à rede de comunicação. Na função *intrasite_ validate* é também onde ocorre a implementação do esquema de assinaturas de limiar (SHOUP, 2000), usado para prover a integridade e autenticidade das mensagens dos agentes ativos dos sítios, um aspecto essencial para o funcionamento do protocolo. Além de garantir a integridade e autenticidade das mensagem, seu uso impede um processo faltoso de forjar assinaturas para mensagens espúrias, sem ser detectado. E se porventura um processo faltoso insistir no envio de mensagens inválidas, elas serão descartadas no receptor, pois toda a recepção de mensagens no protocolo está condicionada à verificação da assinatura, a partir da chave pública do sítio emissor. Esta função é referenciada pelos Algoritmos 6 e 7 através da função *endorse_ message*, e consiste em um dos principais mecanismos utilizados no protocolo para tolerar o comportamento bizantino dos agentes implementados pelos participantes da transação, denotados pelos TMs.

O funcionamento do protocolo, por meio dos TMs apresenta a seguinte dinâmica: cada réplica de TM do sítio recebe as mesmas mensagens, para que a evolução do algoritmo ocorra da mesma maneira em cada um deles. Neste caso, quando eles se deparam como a necessidade de exibir alguma ação através do envio de uma mensagem a outros sítios participantes (i.e., uma comunicação *inter-sítio*), cada réplica de TM gera a mensagem e chama a função *endorse_ message* para assiná-la parcialmente com sua parte da chave secreta, e disponibilizá-la às demais réplicas de TMs do mesmo sítio através de sua *PostBox*. Isto ocorre pela invocação da função *intrasite_ validate*, onde o argumento fornecido na chamada é a mensagem parcialmente assinada. Num primeiro momento, a mensagem é escrita na *PostBox* do processo TM que a invocou e é inicializado um *buffer* para recuperar as mensagens que porventura já haviam sido lidas das *PostBoxes* noutras invocações à função, mas que fazem referência a esta rodada do protocolo. Note que as escritas na *PostBox* são *append-only* e as leituras ocorrem pelo critério *FIFO*, segundo a ordem das escritas. Se uma mensagem é lida da *PostBox*, ela não estará mais disponível para aquele TM e por isso os processos guardam em uma *hashtable* (uma para cada *PostBox*) as mensagens que se deseja fazer uso posterior (linha 3 – $\mathcal{H}[]\langle\rangle$).

A partir dai, a validação da mensagem se dá em duas etapas. Na primeira, que compreende as linhas 17-24, cada processo $p_j \in \mathcal{P}[S_i]$ verifica se nas *hashtables* há mensagens que correspondem àquela que se deseja validar e, em caso afirmativo, o processo faz uma inspeção

nas tabelas para obter apenas as mensagens desejadas, de modo que as mensagens lidas mas não utilizadas naquela rodada do protocolo, retornam para as respectivas tabelas, para uso posterior (linha 24). No caso, a variável *message_key* denota o identificador da transação para a qual a rodada do protocolo está sendo executada. Na etapa que segue (linhas 25 a 34), cada processo $p_j \in \mathcal{P}[S_i]$ entra em um laço que se repete enquanto o processo TM não encontrar pelo menos $f_v + 1$ mensagens iguais de processos TMs distintos (incluindo a sua), para concluir a combinação das assinaturas e então gerar uma assinatura de limiar válida para a mensagem em questão. Note que, se até f_v processos podem falhar de um total de $2f_v + 1$, sempre haverá pelo menos $f_v + 1$ processos corretos. Note também, que em ambas as etapas, os processos consideram apenas as mensagens obtidas, para as quais foi possível verificar a validade da assinatura parcial a partir da respectiva chave de verificação do esquema de limiar (SHOUP, 2000) (linhas 21 e 28). Por fim, obtidas as assinaturas parciais necessárias e corretas (conjunto *certified* - linhas 22 e 30), o laço é encerrado e elas são combinadas para gerar uma assinatura que identifica aquele sítio, e retorna ao processo, a mensagem completamente assinada e preparada para o envio (linhas 34 e 35).

No que segue, o Algoritmo 6 é onde ocorre a especificação do procedimento que implementa o protocolo HB-NBWAC, o qual é invocado pelos TMs dos respectivos sítios que participam da transação. Este procedimento é desenvolvido por meio das tarefas **T1** e **T2**. A tarefa **T1** consiste na implementação do protocolo que coordena a terminação da transação, com o propósito de obter um resultado final único e uniforme para a transação, a despeito da ocorrência de faltas bizantinas em alguns dos agentes TMs dos sítios. O Algoritmo 6 é iniciado quando o sítio participante por meio de seus TMs, conclui sua parte no processamento da transação e invoca o procedimento *AtomicCommit*() para realizar a validação atômica da transação em lide. No caso, o primeiro passo realizado por um processo p_j que representa uma réplica de TM do sítio S_i, consiste na inicialização das variáveis que serão utilizadas naquela rodada do protocolo (linhas 1 a 7), e cujos valores serão modificados no decurso do protocolo de terminação. Na sequência, na linha 8, os TMs definem o líder daquela transação, o que é realizado a partir dos identificadores da transação em questão e também da rodada em que o protocolo se encontra (p. ex.: inicialmente 0 – linha 5). Se o processo p_j que representa um TM pertence ao sítio definido como líder, ele executa os passos entre as linhas 9 e 20, os quais se referem à especificação das atividades pertinentes as **fases** 1 e 2 de um sítio

Algoritmo 6 Tarefa principal do protocolo – sítio S_i – processo p_j.

procedure $AtomicCommit(T_{id})$ /* Procedimento que inicia o protocolo */
Task T1:
(1) $early\_decision$:= `true` /* antecipação da decisão - inicialmente sim */
(2) $decision$:=$\perp$ /* valor de decisão - inicialmente nulo */
(3) $estimate$:=$\perp$ /* estimativa para a decisão - inicialmente nulo */
(4) $proof$:=$\perp$ /* prova de que o CTM do sítio particitou da decisão */
(5) $round$:= 0 /* rodada do protocolo - inicialmente zero */
(6) $votes[\,]$:= $\varnothing$ /* votos recebidos - um arranjo */
(7) $estms[\,]$:= $\varnothing$ /* estimativas recebidas - um arranjo */
(8) `leader` := $S_k : k = ((T_{id} + round)$ **mod** $|\mathcal{S}|)$ /* define o coordenador da transação */
(9) **if** $p_j \in$ `leader` **then** /* Fase 1: somente os TMs do sítio líder */
(10) $\langle message \rangle_{\sigma_{f+1}} := endorse\_message(\langle S_i,$ `REQUEST-VOTE`$, T_{id}\rangle)$
(11) $send(p_j, \langle message \rangle_{\sigma_{f+1}})$ to $\forall p_k \in \{\bigcup_{j=1}^{n=|\mathcal{S}|} \mathcal{P}[S_j] \setminus \mathcal{P}[S_i]\}$
/* Fase 2 */
(12) **wait until** $[(received(\langle S_j,$ `VOTE`$, T_{id}, vote\rangle_{\sigma_{f+1}})$ from some p_k of every $S_j \in (\mathcal{S} \setminus \{S_i\}) : verifysig(S_j, \langle received\_msg \rangle_\sigma))$ **or** $(S_j \in \mathcal{FD}_i)]$
(13) **for each** $S_j \in (\mathcal{S} \setminus \{S_i\})$ **do**
(14) **if** received $\langle S_j,$ `VOTE`$, T_{id}, vote\rangle_{\sigma_{f+1}}$ from $p_k \in S_j$ **then**
(15) $votes[S_j] := \{vote$ **from** msg $\langle S_j,$ `VOTE`$, T_{id}, vote\rangle\}$
/* o líder também determina seu voto */
(16) $my\_vote := vote\_from\_DM(T_{id})$ /* verifica se o DM pode validar a transação */
(17) $votes[S_i] := my\_vote$
(18) **if** $(|votes[\,]| = |\mathcal{S}|)$ **and** $(\forall i, j; 1 \le i, j \le |\mathcal{S}|; i \ne j : votes[i] = votes[j])$ **then**
(19) $estimate$:= `commit` /* todos os votos recebidos e iguais, logo, a estimativa é commit */
(20) $early\_decision$ = `false`
(21) **else** /* Fase 1: todos os TMs dos sítios não-líderes */
(22) **wait until** $[(received(\langle S_j,$ `REQUEST-VOTE`$, T_{id}\rangle_{\sigma_{f+1}})$ from some p_k of `leader` : $verifysig($`leader`$, \langle received\_msg \rangle_\sigma))$ **or** (`leader` $\in \mathcal{FD}_i)]$
/* Fase 2 */
(23) **if** $leader \in \mathcal{FD}_i$ **then**
(24) $my\_vote := no$ /* suspeita de falha do líder e vota pela anulação da transação */
(25) **else**
(26) $my\_vote := vote\_from\_DM(T_{id})$ /* verifica se o DM pode validar a transação */
(27) **if** $my\_vote = yes$ **then**
(28) $\langle message \rangle_{\sigma_{f+1}} := endorse\_message(\langle S_i,$ `VOTE`$, T_{id}, my\_vote\rangle)$
(29) $send(p_j, \langle message \rangle_{\sigma_{f+1}})$ to $\forall p_k \in \{\mathcal{P}[S_j] : S_j =$ `leader`$\}$
(30) $early\_decision$ = `false`
(31) **if** $early\_decision$ = `true` **then** /* o TM votou não e por isso antecipa a decisão */
(32) $\langle message \rangle_{\sigma_{f+1}} := endorse\_message(\langle S_i,$ `DECISION`$, T_{id}, round, abort\rangle_{\sigma_{f+1}})$
(33) $send(p_j, \langle message \rangle_{\sigma_{f+1}})$ to $\forall p_k \in \{\bigcup_{j=1}^{n=|\mathcal{S}|} \mathcal{P}[S_j] \setminus \mathcal{P}[S_i]\}$
(34) **return** $\{abort, \langle message \rangle_{\sigma_{f+1}}\}$
(35) **else** /* Fase 3 : início da rodada de decisão pelo consenso - estimativa */
(36) $\{decision, \langle proof \rangle\} := take\_decision(T_{id}, estimate)$ /* usa a estimativa na decisão */
(37) $send(p_j, \langle proof \rangle)$ to $\forall p_k \in \{\bigcup_{j=1}^{n=|\mathcal{S}|} \mathcal{P}[S_j] \setminus \mathcal{P}[S_i]\}$
(38) **return** $\{decision, \langle proof \rangle\}$

Task T2:
upon $received(\langle S_j,$ `DECISION`$, T_{id}, round, decision\rangle_{\sigma_{f+1}})$ from some p_k of $S_j \in \mathcal{S}$: $verifysig(S_j, \langle received\_msg \rangle_\sigma)$
(39) $decision := \{decision$ **from** msg $\langle S_i,$ `DECISION`$, T_{id}, round, decision\rangle\}$
(40) $round := \{round$ **from** msg $\langle S_i,$ `DECISION`$, T_{id}, round, decision\rangle\}$
(41) $T_{id} := \{T_{id}$ **from** msg $\langle S_i,$ `DECISION`$, T_{id}, round, decision\rangle\}$
(42) $\langle message \rangle_{\sigma_{f+1}} := endorse\_message(\langle S_i,$ `DECISION`$, T_{id}, round, decision\rangle)$
(43) $send(p_j, \langle message \rangle_{\sigma_{f+1}})$ to $\forall p_k \in \{\bigcup_{j=1}^{n=|\mathcal{S}|} \mathcal{P}[S_j] \setminus \mathcal{P}[S_i]\}$
(44) **return** $\{decision, \langle message \rangle_{\sigma_{f+1}}\}$

líder do protocolo HB-NBWAC. De outro modo, se o processo p_j que representa o TM for apenas um participante, ele executa o código das linhas 21 a 30 para realizar as atividades de um sítio não-líder para o protocolo.

Algoritmo 7 Etapa de decisão do protocolo – sítio S_i – processo p_j

```
procedure take_decision(T_id, estimate)                    /* Fase 3 : Decisão pelo Consenso Uniforme */
(1)  while decision =⊥ do
(2)     leader := S_k : k = ((T_id + round) mod |S|)       /* define o coordenador da rodada de decisão */
(3)     round := round + 1
(4)     estms[] := ∅
(5)     if p_j ∈ leader then
(6)        ⟨message⟩_σf+1 := endorse_message(⟨S_i, ESTIMATE, T_id, round, estimate⟩)
(7)     else if p_j ∉ leader then
(8)        wait until [(received(⟨S_j, ESTIMATE, T_id, round, estimate⟩_σf+1) from some p_k
                    of leader : verifysig(leader, ⟨received_msg⟩_σ)) or (leader ∈ FD_i)]
(9)        if received ⟨S_j, ESTIMATE, T_id, round, estimate⟩_σf+1 then
(10)          estimate := {estimate from msg ⟨S_j, ESTIMATE, T_id, round, estimate⟩}
(11)       else
(12)          estimate :=⊥
(13)       ⟨message⟩_σf+1 := endorse_message(⟨S_i, ESTIMATE, T_id, round, estimate⟩)
                                                           /* todos os TMs enviam suas estimativas ao demais sítios */
(14)    send(p_j, ⟨message⟩_σf+1) to ∀p_k ∈ {⋃_{j=1}^{n=|S|} P[S_j] \ P[S_i]}
                                                           /* todos os TMs aguardam pelo recebimento de todas as estimativas dos demais sítios */
(15)    wait until [(received(⟨S_j, ESTIMATE, T_id, round, *⟩_σf+1) from some p_k of every
                    S_j ∈ (S \ {S_i}) : verifysig(S_j, ⟨received_msg⟩_σ)) or (S_j ∈ FD_i)]
(16)    for each S_j ∈ (S \ {S_i}) do
(17)       if received ⟨S_j, ESTIMATE, T_id, round, estimate⟩_σf+1 from p_k ∈ S_j then
(18)          estms[S_j] := {estimate from msg ⟨S_j, ESTIMATE, T_id, round, estimate⟩}
                                                           /* estimativa do próprio sítio - linha 6 para o líder - linha 10 ou 12, e 13 para os não-líderes */
(19)    estms[S_i] := {estimate from my_msg ⟨S_i, ESTIMATE, T_id, round, estimate⟩}
                                                           /* uma decisão pela validação só é legítima na 1ª rodada do consenso */
(20)    if round = 1 then
(21)       if (|estms[]| = |S|) and (∀i, j; 1 ≤ i, j ≤ |S|; i ≠ j : estms[i] = estms[j]) then
(22)          decision := estimate     /* a estimativa inicial é commit, portanto, decision = commit */
(23)          ⟨message⟩_σf+1 := endorse_message(⟨S_i, DECISION, T_id, round, decision⟩)
(24)       else
(25)          estimate := abort
                                                           /* a partir da 2ª rodada do consenso só é possível decidir pela anulação */
(26)    else if round > 1 then
(27)       if (|estms[]| ≥ (|S| − f)) and (∀i, 1 ≤ i ≤ |S| : estms[i] ≠⊥) then
(28)          decision := abort
(29)          ⟨message⟩_σf+1 := endorse_message(⟨S_i, DECISION, T_id, round, abort⟩)
(30)       else
(31)          estimate := abort
(32) return {decision, ⟨message⟩_σf+1}
```

O código das linhas 36 a 38 do Algoritmo 6 compreende ao início da etapa de decisão do protocolo, o que é realizado na 3ª **fase** – especifi-

cada pela função *take_decision*() como parte do Algoritmo 7. Trata-se da última fase do protocolo, a qual é realizada a partir de uma instância de um protocolo de consenso uniforme, e portanto, comum a todos os participantes do protocolo. Todavia, a decisão pelo consenso só é realizada no cenário mais favorável, isto é, aquele em que nas fases que precede a decisão não foram constatadas nenhuma suspeita de falha de algum sítio, e por isso, todos os votos dos CTMs dos sítios não-líderes foram recebidos pelo CTM do sítio líder e tendo em seus respectivos conteúdos uma declaração `SIM`. Neste sentido, após a conclusão das **fases** 1 e 2 por parte dos TMs de cada sítio, independentemente do papél exercido no protocolo HB-NBWAC, se o cenário verificado é favorável, cada sítio entra na **fase** 3 do protocolo de terminação, o que ocorre a partir da chamada à função *take_decision*() (linha 36). Esta função retorna duas importantes variáveis, a *decision* – que contém o valor decidido pelo consenso e, a $\langle proof \rangle$ que contém a mensagem `DECISION` que foi atestada/certificada/assinada pelo CTM daquele sítio, e portanto, serve como prova de que os TMs não faltosos decidiram por um mesmo valor. Note que após a atribuição das variáveis em questão pelo retorno da função *take_decision*(), os TMs dos sítios enviam a mensagem $\langle proof \rangle$ já devidamente atestada/assinada/certificada para todos os TMs dos demais sítios quando da obtenção do valor de decisão. Na sequência, os TMs retornam esta duas variáveis e dão por concluída aquela rodada do protocolo. Por sua vez, o DM de um sítio, ao receber a 2-tupla $\{decision, \langle proof \rangle\}$ de qualquer TM, se constatar a validade desta pela assinatura da mensagem enviada como prova da decisão, pode efetivar a validação ou o descarte das operações (de acordo com o que foi decidido) no banco de dados daquele sítio.

E finalmente, a parte formalizada pelas linhas 39 a 44 do Algoritmo 6 consiste na especificação da tarefa **T2**, a qual diz respeito à implementação de uma primitiva de difusão confiável (DÉFAGO; SCHIPER; URBÁN, 2004). A implementação desta primitiva é necessária para evitar que um processo p_j que representa um TM fique bloqueado para sempre durante alguma rodada da fase de decisão (Algoritmo 7), à espera de uma estimativa advinda de um sítio que já obteve sua decisão, ou mesmo de um valor decidido – isso devido ao comportamento assíncrono do protocolo. Em tal caso, quando os processos TMs de um sítio concluírem sua fase de decisão, eles certificam/atestam o valor de decisão obtido através do processo já mencionado, e na sequência difundem a mensagem contendo tal valor a todos os TMs dos demais sítios, a fim de garantir a recepção daquela mensagem por todos os processos TMs corretos dos sítios não faltosos. Note que esta difusão confiável é impe-

riosa para assegurar tanto a concordância uniforme, como a terminação do protocolo. No que segue, o protocolo será explicado em detalhes, a partir das especificações dos Algoritmos 6 e 7, e de acordo com o diagrama de execução e respectivas fases ilustradas pela Figura 45.

Fase 1 Ao adentrar no procedimento *AtomicCommit*(), o CTM do sítio líder é quem dá início ao protocolo, de modo que os CTMs dos demais sítios aguardam a manifestação do líder para iniciá-lo. Cada processo réplica de TM do sítio líder, quando inicia a primeira fase do protocolo solicita aos participantes dos demais sítios, através de seus respectivos TMs, uma declaração de intenção em validar as operações da transação. Para tanto, cada processo TM do líder envia uma mensagem `REQUEST-VOTE` assinada através do esquema de limiar, que é certificada pelo CTM daquele sítio e conta com a colaboração de, pelo menos, $f_v + 1$ TMs daquele sítio (linhas 10 e 11 - Algoritmo 6).

Os processos TMs dos sítios não-líderes – que estão à espera da manifestação do líder (linha 22 - Algoritmo 6) –, quando recebem uma mensagem `REQUEST-VOTE` certificada pelo CTM do sítio líder, e portanto, contendo uma assinatura válida daquele sítio; dão inicio a fase 2 do protocolo. O mesmo ocorre quando os TMs de um sítio não-líder não recebem a manifestação inicial do sítio líder, devido à suspeita de falha daquele sítio. Para tanto, um único detector de falhas $\mathcal{FD}_i$ é instanciado por sítio e é executado em nível de anfitrião, de modo a fornecer uma mesma visão em termos de suspeitas de falhas dos demais sítios participantes do protocolo, para todos os TMs de um mesmo sítio. Do mesmo modo, cabe ressaltar que o mecanismo de interceptação especificado/implementado em nível de sítio (cfm. Seção 5.5), garante a recepção de uma mensagem por todas as réplicas de TM de um sítio. Neste caso, quando uma mensagem `REQUEST-VOTE` chega em qualquer um dos sítios não-líderes, ele garante a recepção da mesma por parte de todos os TMs não bizantinos executados em nível de convidados, através das respectivas máquinas virtuais gêmeas.

Fase 2 Para os TMs do sítio líder, esta fase tem início no momento em que eles passam a aguardar pelos votos solicitados aos TMs não-líderes na fase anterior (linha 12 - Algoritmo 6). De outro modo, os TMs não-líderes, ao iniciar esta fase do protocolo fazem o seguinte: (*i*) se alguma suspeita de falha recaiu sobre o sítio líder e eles não receberam o pedido de voto oriundo daquele sítio, eles declaram seu voto como `NÃO` (linhas 22 a 24, Algoritmo 6). Note que isso ocorrerá apenas se o sítio líder vier a falhar por parada, pois a atuação do CTM

do sítio líder prevê o envio da mensagem `REQUEST-VOTE` por todos os TMs corretos daquele sítio (i.e., pelo menos $f_v + 1$), o que corrobora para a prevenção de falsas suspeitas pelo detector $\mathcal{FD}_i$; (*ii*) se recebem uma mensagem com o pedido de voto válida e assinada corretamente pelo CTM do sítio líder, eles consultam ao DM local de seus respectivos sítios, a fim de verificar se sua parte da transação logrou êxito (linhas 25 e 26 - Algoritmo 6). Esta consulta ocorre através de uma comunicação *intra-sítio* entre os TMs e o DM de um mesmo sítio, em que a função $vote\_from\_DM(T_{id})$ retorna `SIM` se houve êxito no processamento da transação T_{id}; ou `NÃO` caso algo impeça aquela transação de ser validada. Com isso, baseado no retorno do DM os TMs dos sítios não-líderes preparam suas declarações de voto através da mensagem `VOTE`, com os identificadores do sítio e da transação para a qual o voto está sendo emitido, bem como o voto propriamente dito. A mensagem é submetida ao processo de endosso e validação *intra-sítio* já mencionado e, uma vez assinada ela é enviada aos TMs do sítio líder (linhas 28 e 29, Algoritmo 6), e o TM modifica o valor da variável $early\_decision$ para `false` para que ele execute a fase de decisão (especificada pelo Algoritmo 7). A partir daí, os TMs não-líderes iniciam a fase de decisão pela chamada da função $take\_decision()$ (linha 36), e ficam à espera dos valores que serão fornecidos pelo retorno da chamada desta função – eles serão atribuídos no decurso da execução do Algoritmo 7. Note que os sítios não líderes fornecem como argumento para a chamada da função o identificador da transação T_{id} e suas estimativas para decisão (i.e., valores nulos $\perp$).

Com relação à execução desta fase pelo sítio líder, um TM daquele sítio ao receber as mensagens contendo a declaração de voto emitida pelo CTM de cada um dos sítios não-lideres, armazena as mensagens recebidas e computa os respectivos votos (linhas 11 a 14 - Algoritmo 6). Nessa situação, como o TM líder também é um participante da transação, nos mesmo moldes dos TMs não-líderes ele consulta seu DM local para subsidiar sua estimativa para o resultado da transação (linha 16, Algoritmo 6). No passo seguinte, os TMs do sítio líder calculam uma estimativa de resultado para a transação, baseado nos votos recebidos e possíveis suspeitas recaídas sobre os sítio não-líderes. Duas condições são observadas para o cálculo da estimativa, 1) se o tamanho do arranjo/conjunto que contém os votos é igual ao tamanho do conjunto que denota os sítios participantes do protocolo, o que indica que todos os votos foram recebidos; 2) se todos os valores contidos no conjunto dos votos são iguais; estas especificadas na linha 18 do Algoritmo 6. Se ambas as condições forem satisfeitas, os TMs líderes calculam

a estimativa a partir da sua declaração de voto, o qual também está presente no arranjo/conjunto *votes*[]. Portanto, a estimativa será declarada como `commit` se o seu voto foi `SIM`, ou `abort` caso o seu voto tenha sido `NÃO` – note pela segunda condição da linha 18, que o voto do sítio líder é igual a todos os votos dos demais sítios. Isto encerra a fase atual do protocolo, e os TMs do sítio líder dão início à fase de decisão do protocolo, o que ocorre pela chamada da função *take_decision*(), em que eles fornecem o T_{id} da transação e a estimativa recém-calculada, cujos valores serão utilizados na primeira rodada do consenso (linha 36, Algoritmo 6).

É importante salientar que uma decisão antecipada, tal como que é previsto na especificação original dos protocolos NBAC e NBWAC (BABAOGLU; TOUEG, 1993; GUERRAOUI; LARREA; SCHIPER, 1995), é atendida pela especificação HB-NBWAC nas linhas 31 a 34 do Algoritmo 6. A decisão antecipada é algo plausível em situações onde o CTM de algum sítio declara seu voto como `NÃO`, já que esta declaração de voto induzirá o protocolo espontaneamente a decidir pela anulação da transação. De acordo com as propriedades definidas para o problema (BABAOGLU; TOUEG, 1993; GUERRAOUI; LARREA; SCHIPER, 1995), a anulação pode ser vista como a única decisão aceitável e legítima, e por esta razão o protocolo deve admitir este tipo de decisão. Tal condição é especificada no protocolo pela variável *early_decision* (linha 1, Algoritmo 6), onde o protocolo parte do pressuposto que é prevista a decisão antecipada em todas as execuções. Note que esta condição/variável será alterada, quando um cenário favorável a uma decisão `COMMIT` for verificada pelos CTMs dos sítios. A variável *early_decision* pode ser alterada em dois momentos da execução do protocolo: (*i*) nas linhas 18 a 20, pelo CTM do sítio líder, ao verificar que não houve nenhuma suspeição de falha e que todos os votos recebidos são `SIM`; (*ii*) nas linhas 27 a 30, pelos CTMs dos sítios não-líderes, após eles declararem seus respectivos votos com `SIM`.

Estas duas situações revelam a ocorrência de cenários favoráveis para uma decisão `COMMIT`, e por esta razão, se elas são verificadas pelos respectivos CTMs ao término da fase 2, estes são conduzidos à decisão pelo consenso uniforme através da função *take_decision*() (linhas 35 a 38). De outro modo, se tais situações não forem verificadas, resulta que os CTMs não alteram os valores de sua variável local *early_decision* e por isso eles executam o código entre as linhas 31 e 34, onde são conduzidos a uma decisão antecipada pela anulação da transação (i.e., `ABORT`). Neste caso, o CTM que decide antecipadamente envia aos CTMs dos demais sítios uma mensagem `DECISION` devidamente as-

sinada/certificada/atestada (linhas 32 e 33) e com isso dá por encerrada a execução do protocolo pelo retorno da 2-tupla $\{abort, \langle message \rangle_{\sigma_{f+1}}\}$ na linha 34. A 2-tupla retorna o valor de decisão, que no caso é `abort`, e a mensagem que fora enviada aos CTMs dos demais sítios. Esta serve como prova de que, pelo menos, $f_v + 1$ TMs daquele sítio chegaram àquela decisão e por isso atestaram, certificaram e assinaram a mensagem. Tal situação impede um TM bizantino de forjá-la ou mesmo de inventar ou alterar uma decisão obtida no âmbito sítio em que ele atua.

Fase 3 Se o CTM não decidiu de maneira antecipada, ele prossegue para a 3ª fase do protocolo, a qual tem início quando o código da função *take_decision*() é instanciado pela chamada a ela na linha 36 do Algoritmo 6, onde o fluxo de execução passa para o Algoritmo 7. A partir daí, é iniciada a primeira rodada do consenso para tentar obter uma decisão acerca da estimativa calculada pelos TMs do sítio líder. Neste sentido, note que o primeiro passo da função é um laço de repetição, em que os TMs de todos os sítios permanecem em iteração sobre ele até que um valor seja atribuído à variável *decision*, o que ocorre apenas ao término de cada rodada do consenso pelas linhas 22 e 28 (i.e., uma ou outra, não ambas). O algoritmo formalizado para a tarefa de decisão segue a mesma dinâmica de funcionamento do consenso de Mostéfaoui e Raynal (1999), com algumas modificações para atender à semântica das propriedades do NBWAC. Informalmente, o passo de decisão ocorre por meio de um ciclo de rodadas, de modo que cada rodada opera da seguinte maneira:

1. o sítio líder da rodada envia sua estimativa aos demais sítios, no intuito de chegar a uma decisão unânime a partir desta;

2. os sítios não-líderes na mesma rodada esperam pela estimativa advinda do sítio líder, e determinam suas estimativas locais baseada naquela recebida do líder – se eles não tiverem recebido a estimativa e suspeitarem que o líder falhou, uma estimativa sem valor é definida (p. ex.: $\perp$);

3. os sítios não-líderes enviam suas estimativas aos demais sítios, inclusive para o líder;

4. se todas as estimativas foram recebidas na primeira rodada, e os sítios verificam que elas são todas iguais e o valor delas é `commit`, eles decidem pela validação da transação; se todas as estimativas foram recebidas mas em uma rodada posterior à primeira, o

único valor decidido será `abort`; em ambos os casos, após a tomada da decisão os CTMs enviam uns aos outros uma mensagem informando a decisão, e a rodada na qual ela foi obtida;

5. se não foram recebidas as estimativas de todos os sítios, ou se dentre as recebidas houver alguma sem valor (i.e., $\perp$), a estimativa é calculada como `abort`, o ciclo se repete e uma nova rodada de decisão é iniciada com um outro sítio definido como líder.

Ao iniciar o laço de repetição, os CTMs definem o líder daquela rodada do consenso, que na primeira rodada será sempre o mesmo sítio atribuído como líder para o protocolo de terminação (linha 2), o que decorre do fato deste ser o único sítio a ter recebido os votos dos demais sítios e calculado a estimativa de decisão na fase anterior. Na sequência o número da rodada, que fora inicializado na fase 1 como zero (0), é incrementado em uma unidade, e o arranjo que será utilizado para armazenar as estimativas recebidas dos sítios é inicializada – note que estes passos se repetem a cada interação que ocorre no laço, quando uma nova rodada para o consenso é iniciada. Nos passos subsequentes, os TMs líderes preparam uma mensagem contendo a estimativa com base nos votos recebidos na fase anterior, que após certificada pelo CTM é enviada aos demais, isto ocorre nas linhas 6 e 14. Após enviarem a estimativa, os TMs líderes ficam à espera das estimativas advindas dos demais sítios para subsidiar uma decisão para a transação (linha 15).

Os TMs não-líderes executam os passos entre as linhas 7 e 13, onde primeiramente aguardam pela chegada da mensagem contendo a estimativa enviada pelo líder daquela rodada, para então dar continuidade ao processo de decisão. A continuidade do protocolo acontece quando eles recebem, na mesma rodada de decisão, a estimativa do líder; ou quando não recebem a estimativa em decorrência de uma suspeita de falha do líder daquela rodada (linhas 9 e 11). Se a estimativa foi recebida, ele executa as linhas 9 e 10, onde calcula sua estimativa de acordo com aquela recebida do líder – contida na mensagem $\langle S_i, \texttt{ESTIMATE}, T_{id}, round, estimate\rangle$. Por outro lado, se recaiu uma suspeita sobre o sítio líder daquela rodada, os TMs não líderes definem suas estimativas como um valor nulo ($\perp$ – linhas 11 e 12). Calculada a estimativa, os TMs não-líderes a encapsulam numa mensagem, que é certificada pelo CTM de cada sítio não-líder e posteriormente enviada a todos os demais sítios (linhas 13 e 14). Um aspecto importante decorre do fato de que nenhum sítio pode enviar diferentes estimativas para cada sítio, pois uma mensagem só é considerada válida se ela for certificada/atestada/assinada pelo CTM do sítio emissor. E se assim

não o for, a ação arbitrária/bizantina quanto ao envio da mensagem culminará na rejeição desta por parte dos sítios receptores.

Note que os passos subsequentes à linha 15 do Algoritmo 7, são comuns a todos os TMs participantes do protocolo de terminação (i.e., líderes e não-líderes). Neste sentido, a linha 15 é onde os TMs de todos os sítios aguardam pelas estimativas enviadas de uns para os outros, no intuito de subsidiar a decisão daquela rodada do protocolo de consenso. Assim, a sequência do algoritmo se dá pelo recebimento de todas as estimativas dos demais sítios, ou pelo recebimento algumas estimativas e suspeição de falha daqueles sítios para os quais as estimativas não foram recebidas (linha 15). Os passos das linhas 16 a 18 é onde os sítios obtém as estimativas recebidas naquela rodada e armazenam-nas no arranjo *estms*[] (p. ex.: uma entrada/índice por sítio). Em seguida os CTMs de cada sítio armazenam suas próprias estimativas no arranjo *estms*[], pois eles não as enviaram para si mesmos, mas elas também devem ser usadas para subsidiar uma decisão. Uma vez que são computadas as estimativas recebidas, os TMs realizam as verificações acerca do conteúdo do arranjo *estms*[] para determinar se é possível chegar a um valor de decisão, a partir das estimativas recebidas.

Neste sentido, numa etapa antes da decisão é necessário verificar se aquela é a primeira rodada de decisão do protocolo (i.e., $round = 1$ – linha 20). Pois, pela interpretação estrita das propriedades do NBWAC, uma decisão `commit` só é legítima num cenário mais favorável onde todos votam `sim` e não há suspeitas de falhas, o que é verificado apenas na primeira rodada de decisão[2]. Se aquela for a primeira rodada de decisão, os TMs verificam se as estimativas de todos os sítios foram recebidas e se elas são todas iguais (linha 21) – a verificação é feita sobre o tamanho e conteúdo do arranjo *estms*[] – e em sendo, o TM determinada a decisão baseado em sua estimativa, já que ela é igual à todas as demais recebidas. No que segue, o valor da variável *decision* será `commit`, em consonância com a estimativa inicial enviada pelo CTM do sítio líder, e adotada como estimativa local do CTM de cada sítio nas linhas 8 e 9, e após definida, a variável *decision* é encapsulada numa mensagem `DECISION` que passa pelo processo de validação/certificação do CTM daquele sítio. E uma vez assinada, esta mensagem passa a ser usada como prova de que a decisão daquele sítio foi tomada pelo CTM, sendo então retornada à tarefa **T1** acompanhada do valor de decisão, na linha 32.

De outro modo, se pelo menos, uma das condições da linha 21

[2]O algoritmo só segue para a próxima rodada, caso uma suspeita de falha recaiu sobre algum(ns) dos participantes.

da tarefa de decisão não for atendida (no Algoritmo 7), é porque não foram recebidas todas as estimativas dos sítios ou dentre elas havia uma estimativa de valor nulo ($\perp$). Ambos os casos indicam que houve suspeita de falha na linha 15, o que resultou no não recebimento de todas as estimativas, ou algum outro sítio suspeitou da falha do líder e por isso enviou uma estimativa nula (linha 12). Neste cenário não é mais possível validar a transação, e por isso, os TMs são desviados para a condição da linha 24 da tarefa de decisão, onde determinam suas estimativas locais como `abort`. Por conseguinte, o ciclo do laço se repete, uma nova rodada de decisão é iniciada tendo `abort` como estimativa dos CTMs de todos os sítios, um novo líder é atribuído para aquela rodada e eles novamente executam o passo de decisão. Por sua vez, este ciclo se repete até que os sítios obtenham uma decisão numa rodada do consenso.

Em ambos os casos, isto é, numa decisão em qualquer rodada $i \geq 1$, o laço do Algoritmo 7 é encerrado e os TMs do sítio que acabara de decidir, retornam o valor de decisão e a sua respectiva prova, a partir de uma 2-tupla contendo a variável *decision* e a mensagem `DECISION` certificada pelo CTM daquele sítio. E pelo retorno da linha 32, o fluxo de execução volta para a linha 36 do Algoritmo 6, onde a 2-tupla é recebida e a mensagem de prova (i.e., `DECISION` gerada na fase de decisão) é enviada a todos os CTMs dos demais sítios para subsidiar a decisão uniforme daqueles que não lograram êxito na mesma rodada de decisão que resultou no valor para o CTM emissor, e por isso podem estar ainda executando alguma rodada posterior da fase de decisão, e aguardando pela(s) estimativa(s) oriunda(s) do(s) CTM(s) que já decidiu(ram). Enviada a mensagem de prova, a 2-tupla é então retornada para o processo que instanciou a função *AtomiCommit*(). E conforme já mencionado, quando do recebimento da 2-tupla pelo retorno da função *AtomiCommit*(), o processo fornece-a ao DM daquele sítio, que ao recebê-la verifica a validade da decisão através da assinatura da mensagem usada como prova – que corresponde à mesma assinatura construída a partir da chave privada do sítio que se encontra em sua posse. Se o DM verifica a validade da prova de decisão, ele confirma ou descarta as operações executadas para a transação, de acordo com a decisão tomada, no banco de dados daquele sítio. Ao realizar as ações pós-decisão, o DM daquele sítio coloca o resultado obtido para aquela transação no formato $\langle T_{id}, \texttt{outcome} \rangle_{\sigma}$ (vide linha 4 do Algoritmo 5) numa *PostBox* especificamente usada para registrar o histórico das transações realizadas – note que o valor de `outcome` será `commit` ou `abort`.

5.7 CORREÇÃO DO PROTOCOLO

Nesta seção demonstramos que o protocolo proposto satisfaz as propriedades especificadas para o problema da Validação Atômica Fraca Não-Bloqueante (GUERRAOUI, 1995), nomeadamente:

- **Validade Uniforme**: se um participante decide `COMMIT`, então todos os participantes votaram `SIM`;
- **Acordo Uniforme**: todos os processos que decidem obtém um mesmo valor de decisão;
- **Integridade**: um participante decide no máximo uma vez, e não pode reverter sua decisão após ela ter sido tomada;
- **Terminação**: todos os participantes corretos terminam por obter um valor de decisão;
- **Não-Trivialidade**: se todos os participantes votaram `SIM` e não há suspeitas de falhas, então a decisão é `COMMIT`.

Para realizarmos esta prova, algumas hipóteses são presumidas no âmbito da especificação do protocolo, são elas:

1. cada mensagem direcionada a qualquer TM de um sítio é interceptada em nível de hospedeiro, a fim de prover um mecanismo de recepção confiável daquela mensagem a todos os TMs daquele sítio. Uma mensagem interceptada é disseminada a todos os TMs daquele sítio, de modo que ela é incluída no *buffer* de recepção de cada TM, na respectiva ordem em que ocorreu a interceptação pelo ambiente hospedeiro (i.e., FIFO);
2. tomando em conta que a *PostBox* é uma abstração de memória compartilhada *append-only* e cuja capacidade é finita, periodicamente os TMs efetuam a limpeza das mensagens contidas em suas respectivas *PostBoxes*, a fim de evitar o esgotamento/transbordamento da capacidade. Esta limpeza ocorre baseada na obsolescência das mensagens contidas nas *PostBoxes* (p. ex.: mensagens das rodadas já concluídas pelo protocolo), por meio de um *checkpointing* que é realizado através de um acordo entre os TMs do sítio;
3. embora as derivas entre os TMs de um mesmo sítio sejam limitadas (vide Seção 5.6.1), se um TM que está um pouco atrasado

em relação aos demais percebe que alguma mensagem requerida por ele para a conclusão da instância atual do acordo já foi removida das *PostBoxes* de outros TMs, este TM atualiza seu estado a partir da verificação na *PostBox* que contém o histórico das transações. E neste caso, ele dá por concluídas as rodadas pendentes de execução, pois elas já foram decididas e atestada pelo CTM daquele sítio;

4. tendo em consideração que: (*i*) os TMs de um sítio são réplicas uns dos outros; (*ii*) eles executam os mesmos passos/etapas do protocolo; (*iii*) eles recebem as mensagens numa mesma ordem devido ao mecanismo de intercepção descrito no item 1; estas asserções implicam que os TMs corretos de um sítio sempre evoluem para as mesmas etapas do protocolo. Portanto, a atuação do CTM de um sítio através de $f + 1$ TMs sempre resultará em ações consistentes e corretas no âmbito do protocolo.

Com essas premissas é possível provar a corretude do protocolo, tendo em conta, inclusive, a realização de um procedimento de coleta de lixo para uma gestão de memória mais racional das *PostBoxes*. Ademais, como as *PostBoxes* são abstrações de memória compartilhada especificadas sobre memória secundária (TANENBAUM, 2007), elas são baseadas em mecanismos de armazenamento persistente (i.e., não volátil), que são tecnologias abundantes e baratas na atualidade.

Antes de demonstrar que o protocolo satisfaz as propriedades definidas para o NB-WAC, primeiro demonstraremos uma invariante do algoritmo através de um lema intermediário, e útil para a prova das propriedades do HB-NBWAC.

Lema 5.1 *Se os CTMs de dois sítios i e j ($i \neq j$) decidem na linha 22 ou na linha 28 do Algoritmo 7 durante a mesma rodada r, eles decidem pelo mesmo valor.*

Prova Se ambos os CTMs do sítio i e j decidiram durante a rodada $r = 1$, eles o fizeram na linha 22 por consequência de não ter havido nenhuma suspeita de falha por parte de nenhum sítio, durante aquela rodada. Além disso, a decisão só foi possível porque durante as fases que precederam a decisão, também não ocorreu nenhuma suspeita de falha, e o CTM do sítio líder calculou a estimativa como `commit`, em razão de ter recebido todos os votos dos demais sítios declarados como `sim`. De outro modo, se os CTMs do sítio i e j decidiram numa rodada $r > 1$, é porque nas rodadas anteriores alguma suspeita de falha recaiu sobre algum sítio participante do protocolo, o que por conseguinte,

resultou na alteração das estimativas de todo os sítios para `abort`. No caso, a decisão numa rodada $r > 1$ ocorre apenas na linha 28, se durante aquela rodada a estimativa enviada inicialmente pelo CTM do sítio líder alcançou a todos os CTMs dos sítios corretos (i.e., $n - f$), e as estimativas enviadas posteriormente por todos os CTMs dos sítios não-líderes corretos foram recebidas por todos os CTMs corretos. Em ambos os casos, seja *decision* o valor de decisão da rodada $r \geq 1$, temos $CTM_i = decision_i$ e $CTM_j = decision_j$, em que $decision_i = decision_j = estimativa_r$. Logo, o Lema é satisfeito. □

Teorema 5.1 (Acordo Uniforme) *O protocolo satisfaz o acordo uniforme, em que todos os participantes que decidem obtém um mesmo valor de decisão.*

Prova Esta prova é demonstrada por contradição. Suponha, por contradição, que dois sítios participantes i e j (tal que $i \neq j$) chegaram a diferentes decisões, onde um decidiu `COMMIT` e o outro `ABORT` em alguma rodada do protocolo. Para tanto, considere sem perda de generalidade uma rodada do protocolo $r \geq 0$, onde j vota `não` e i vota `sim`. Pelo Lema 5.1, temos que, se $r_i = r_j$, então $decision_i = decision_j$. Neste caso, considere o cenário onde o sítio i decide na rodada r_i e o sítio j decide na rodada r_j, em que $r_i \neq r_j$. Analisaremos os casos em que esta situação pode ocorrer:

Caso 1: Supondo que j é o CTM de um sítio não-líder que atestou/certificou um voto `não` para aquele sítio. Para que isso tenha ocorrido, ele executou o código das linhas 21 a 30 do Algoritmo 6, cuja declaração de voto ocorreu porque ele suspeitou de falha do líder (linhas 23 e 24 - Algoritmo 6) ou porque o DM daquele sítio não estava apto a validar a transação (linha 26 - Algoritmo 6). E como consequência daquele voto, ele não executou a linha 30 do Algoritmo 6, o que o conduziu diretamente para uma decisão antecipada pela anulação da transação (linhas 31 a 34). Por isso, ele enviou a mensagem `DECISION` contendo a decisão tomada, devidamente certificada/assinada pelo CTM daquele sítio a todos os demais sítios participantes (linhas 32 e 33 - Algoritmo 6). Ao completar o envio da mensagem de decisão, os TMs daquele sítio, que já atestaram a mensagem antes do envio, executaram a linha 34 do Algoritmo 6 e deram por concluída aquela execução do protocolo. Do mesmo modo, cada CTM dos demais sítios participantes $i \neq j$, ao terem recebido a mensagem `DECISION` pelas suas respectivas tarefas **T2** do Algoritmo 6, obtiveram os valores

recebidos pela mensagem, prepararam uma mensagem idêntica, porém, de sua autoria, certificaram-na e a assinaram (linha 42 - Algoritmo 6). E uma vez que a mensagem foi atestada/assinada pelo CTM daquele sítio, cada TM correto enviou a mensagem `DECISION` idêntica àquela recebida do CTM do sítio j, aos demais sítios. E por fim, após o envio eles executaram a linha 44, onde também deram por concluída a execução do protocolo. Neste cenário, o sítio j que votou `não` decidiu na linha 34, enquanto que o sítio $i \neq j$ decidiu a partir da recepção da mensagem `DECISION` enviada pelo sítio j, o que ocorreu na linha 44 e pelo mesmo valor decido por j, e enviado a ele. Portanto, uma contradição.

Caso 2: Presumindo que i é o CTM de um sítio não-líder que atestou/certificou um voto `sim` para aquele sítio, e que j é o sítio líder. À vista disso, CTM do sítio i executou o código entre as linhas 21 a 30, onde determinou seu voto na linha 26 pelo retorno da função $vote\_from\_DM()$. Como consequência da atuação do CTM, pelo menos $f + 1$ TMs do sítio i certificaram/assinaram a mensagem `VOTE` (linha 28) e enviaram-na ao CTM j do sítio líder. A partir daí, como o CTM i votou `sim`, ele modificou a variável $early\_decision$ para `false` e por isso iniciou sua fase de decisão (linha 36), onde passou a aguardar a estimativa do líder (linha 8 - Algoritmo 7). Em decorrência do recebimento das mensagens `VOTE` de todos os CTMs i ($i \neq j$), o CTM j do sítio líder determinou seu voto (linha 16) e calculou sua estimativa subsidiado pelas mensagens recebidas e também pelo seu voto (linhas 18 a 20). Em seguida, o CTM j do sítio líder também entrou na fase de decisão (linha 36), onde foi iniciada a primeira rodada de decisão/consenso (i.e., $round = 1$) com o envio de sua estimativa (i.e., aquela fornecida como parâmetro para a fase de decisão) na mensagem `ESTIMATE` devidamente certificada/atestada/assinada aos CTMs dos demais sítios (linhas 6 e 14 do Algoritmo 7) – note que o líder determinado no início do protocolo é o mesmo para a primeira rodada de decisão (linha 2 do Algoritmo 7 – $round = 1$). Se o CTM i de cada sítio não-líder recebeu a estimativa do líder na primeira rodada, ele a adota como sua estimativa, ou se tiver suspeitado do CTM j do sítio líder ele define sua estimativa como nula ($\perp$) – em ambos os casos, o CTM de cada sítio envia sua estimativas devidamente certificadas/assinadas a todos os demais sítios (linhas 9, 10 e 14 do Algoritmo 7). Se algum CTM i de um sítio não faltoso recebeu as estimativas de todos os demais CTMs j ($i \neq j$) na primeira rodada do protocolo, é por-

que não houve suspeitas de falhas (linhas 15 a 19), e por isso ele inspecionou os valores recebidos nas estimativas para verificar se todas eram iguais. Se todas estas condições foram satisfeitas na primeira rodada do protocolo (linha 21, Algoritmo 7), ele adotou sua estimativa como valor de decisão (`commit`) e elaborou uma mensagem de prova daquela decisão (i.e., `DECISION`), a qual foi certificada/assinada pelo CTM daquele sítio (linhas 22 e 23, Algoritmo 7). A partir daí o CTM retornou a 2-tupla contendo o valor de decisão e a mensagem de prova para a linha 36 do Algoritmo 6, cuja mensagem de prova foi enviada a todos os CTMs dos demais sítios e então a execução do protocolo se deu por encerrada na linha imediatamente posterior ao envio. De outro modo, os CTMs que não receberam todas as estimativas na primeira rodada devido a alguma suspeita de falha (linha 25, Algoritmo 7), eles modificaram suas estimativas para `abort` e iniciaram uma nova rodada de decisão (i.e., $round = 2$). Todavia, os CTMs que estavam numa rodada posterior àquela em que o CTM decidiu e enviou a mensagem de prova `DECISION`, receberam tal mensagem pelas suas respectivas tarefas **T2** do Algoritmo 6. E em decorrência deste recebimento, eles decidiram pelo mesmo valor decidido por aquele CTM e também enviaram suas respectivas decisões e deram por concluída aquela execução do protocolo (linhas 43 e 44). Uma situação que também contradiz a demonstração em prova.

Nas situações enumeradas por 1 e 2 se verifica contradições à demonstração em prova. Logo, o acordo uniforme é válido para o protocolo e o teorema segue. □

Teorema 5.2 (Validade Uniforme) *Se um participante decidiu `COMMIT` para uma transação, então todos os participantes votaram `SIM`.*

Prova Para esta prova, iremos demonstrar que uma decisão pela validação (ou `COMMIT`) só é possível quando todos os participantes votam `sim`. Pela especificação do protocolo, as decisões ocorrem nas linhas 34, 38 e 44 do Algoritmo 6. O único valor decido pela linha 34 é `abort`, como consequência do CTM de algum sítio ter votado `não`. De outro modo, uma decisão pela linha 44 só ocorre pelo recebimento de um valor que já foi decidido pelo CTM de algum outro sítio, cujo valor pode ser `commit` ou `abort`. No caso da linha 38, a decisão é o resultado da execução da fase de decisão do protocolo, a qual é baseada em um consenso que é formalizado pelo Algoritmo 7. Pelo Algoritmo 7, um

valor de decisão `COMMIT` só é possível na 1ª rodada de decisão (linha 20), desde que as estimativas de todos os CTMs tenham sido recebidas e seus respectivos conteúdos eram `commit` (linhas 21 e 22). Para que isso tenha ocorrido, o CTM líder da 1ª rodada enviou sua estimativa inicial com o valor `commit`, que fora calculada como tal, em decorrência do recebimento de declarações de voto `sim` de todos os CTMs, inclusive a sua (linhas 18 e 19, Algoritmo 6). E uma vez que o valor de decisão foi atribuído na linha 22 do Algoritmo 7 (i.e., para a variável *decision*), ele foi retornado acompanhado da prova de validade daquele resultado (i.e., atestado/certificado/assinado pelo CTM daquele sítio) para a tarefa **T1** do Algoritmo 6, que decidiu por tal valor na linha 38. Isto comprova o teorema em lide. □

Teorema 5.3 (Integridade) *Um participante decide no máximo uma vez, e não pode reverter sua decisão após ela ter sido tomada.*

Prova Pela especificação do protocolo, a decisão é realizada em um dos três pontos do Algoritmo 6, isto é, na linha 34 ou 38 ou 44. Se o CTM de um sítio decide de maneira unilateral pela anulação da transação (i.e., `abort`), ele o faz pela execução das linhas 32 e 33, o que resulta na terminação imediata do protocolo para aquele sítio na linha 34. Se porventura o CTM de um sítio decide pela linha 38, é porque ele invocou a função *take_decision*() para que a fase de decisão fosse iniciada, e em alguma rodada da fase de decisão houve um consenso por parte de todos os CTMs quanto à estimativa do CTM líder daquela rodada (vide Algoritmo 7). Diante disso, o consenso foi concluído pela linha 22 ou pela linha 28, sendo que para ambos os casos, a rodada do consenso se deu por encerrada (i.e., $decision \neq \perp$) e o valor decidido foi retornado na linha 32 do Algoritmo 7, para a efetivação da decisão pela tarefa **T1** (linha 36 - Algoritmo 6). Por sua vez, a tarefa **T1** ao receber o resultado da decisão, deu por encerrada aquela execução do protocolo ao retornar o valor de decisão e a prova recebida para esta (linha 38, Algoritmo 6). E finalmente, se um CTM decide pela linha 44, é porque ele não concluiu sua fase de decisão e ainda não decidiu, mas algum CTM de outro sítio que já decidiu enviou seu o valor de decisão a todos os CTMs dos demais sítios (i.e., `DECISION`). Em tal caso, o CTM que ainda não decidiu mas recebeu a mensagem `DECISION`, adotou o valor recebido na mensagem como seu valor de decisão, elaborou uma mensagem de prova baseada nos mesmos parâmetros recebidos, enviou esta aos demais CTMs, onde posteriormente encerrou aquela execução do protocolo na linha 44. Nos três cenários em que uma decisão é possível, a 2-tupla com o valor decidido e a prova é retornado imediatamente,

o que portanto, conclui aquela execução do protocolo e as respectivas tarefas que porventura estavam em execução. Esta asserção demonstra que cada CTM decide no máximo uma vez, e não reverte sua decisão em nenhum ponto de execução. Portanto, o teorema segue. □

Teorema 5.4 (Terminação) *Todos os participantes corretos chegam a um valor de decisão.*

Prova Para a construção desta prova, iremos analisar os pontos do protocolo onde há a possibilidade de blocagem, os quais podem impedir a continuidade da execução do protocolo, e consequentemente sua terminação:

Caso 1: A função *endorse_message* é invocada por um TM para certificar, atestar e assinar todas as mensagens que circundam cada ambiente local do protocolo. Esta função realiza a comunicação *intra-sítio* com os outros TMs do mesmo sítio, através de uma chamada à função *intrasite_validate*(), cuja especificação tem um ponto de blocagem na condição do laço de repetição da linha 25 (Algoritmo 5). Por construção, a função *postbox.read*() é não-bloqueante, e isto posto, se não houver mensagens na *PostBox* no momento da invocação da mesma (linha 27), ela retorna um valor nulo (i.e., $\perp$). Pela condição da linha 28, para a agregação de uma mensagem ao esquema de assinaturas de limiar, além de ter sido retornada da *PostBox* ela deve ser integra e válida quanto às suas assinaturas parciais. Portanto, como cada sítio mantém $2f_v + 1$ réplicas de TM, das quais até f_v podem falhar de maneira bizantina, sempre haverá um quórum formado por, pelo menos, $f_v + 1$ TMs corretos, os quais finalizam suas respectivas chamadas à função (i.e. eles não ficam bloqueados indefinidamente no laço de repetição);

Caso 2: As linhas 12 e 22 do Algoritmo 6, bem como as linhas 8 e 15 do Algoritmo 7 caracterizam pontos de blocagem no protocolo. Como é presumido que as comunicações só ocorrem por meio de canais ponto-a-ponto confiáveis, todas as mensagens enviadas através dos TMs de um sítio correto finarão por serem recebidas. De outro modo, se um sítio participante do protocolo vier a falhar antes do envio de uma mensagem, pela propriedade de **completude forte** do detector $\diamond\mathcal{S}$ utilizado em nível de sítio, todo sítio faltoso finará por ser suspeito. Portanto, os TMs dos sítios não ficam bloqueados indefinidamente à espera de mensagens;

Caso 3: Num cenário onde um CTM_i decidiu numa rodada r', ele deu por encerrada aquela execução do protocolo, e por isso parou de enviar estimativas aos CTMs que ainda não haviam decido naquela mesma rodada. Como estes CTMs possivelmente estão em rodadas de decisão r'' posteriores àquela em que o CTM_i decidiu, eles permanecerão à espera das estimativas oriundas do CTM_i, as quais nunca chegarão. E se porventura o sítio que abriga o CTM_i jamais vier a falhar, pela propriedade de **exatidão terminal fraca**, a partir de algum momento ele nunca virá a ser suspeito por nenhum outro sítio correto. O que induzirá os CTMs que ainda estão em fase de decisão, a aguardar pela estimativa de CTM_i para sempre. Todavia, no momento em que o CTM_i decidiu, ele enviou uma mensagem `DECISION` a todos os CTMs dos demais sítios, que quando a receberam, decidiram pelo mesmo valor de CTM_i, enviaram a mesma mensagem `DECISION` a todos os CTMs dos demais sítios e encerram suas respectivas execuções do protocolo. Isto implica que todo CTM_j ($\forall j \neq i$) finará por receber a mensagem `DECISION` e acabará por decidir, e consequentemente, por encerrar sua execução do protocolo.

Pelas demonstrações realizadas, em qualquer ponto passível de blocagem no protocolo, os CTMs dos sítios não permanecem em estado de espera indefinida e por isso completam a execução do protocolo, que é finalizada pela decisão. Sendo assim, o teorema é comprovado. □

Teorema 5.5 (Não-Trivialidade) *Se todos os participantes votaram `SIM` e não houve suspeitas de falhas, então a decisão é `COMMIT`.*

Prova Esta prova é realizada pela demonstração de que o protocolo só decide `ABORT` se algum CTM emitiu e atestou um votou `não` ou se em alguma fase do protocolo houve a suspeita de falha de outro sítio. Pela especificação do protocolo, todas as decisões ocorrem no Algoritmo 6, e um CTM só decide pela anulação (i.e., `ABORT`) na linha 34, ou então pela execução da fase de decisão (linhas 36-38), ou ainda pela recepção de um valor de decisão na linha 44. Portanto, temos que analisar apenas os casos em que os CTMs decidem pelas condições verificadas na execução do protocolo e não pela decisão de outrem (i.e., na linha 44), vejamos:

Caso 1: Para que ele tenha decidido na linha 34 do Algoritmo 6, ele era o CTM de um sítio que votou `não`, e por consequência de seu voto decidiu de maneira unilateral pela anulação (linhas 24, 26

e 31 a 34 - Algoritmo 6), onde enviou sua decisão na mensagem `DECISION` a todos os CTMs dos demais sítios. Portanto, tal ação culminou em uma decisão antecipada pelo protocolo, em que todos os CTMs dos demais sítios também decidiram pela anulação, quando da recepção da mensagem `DECISION` – tarefa **T2** (linhas 39 a 44, Algoritmo 6);

Caso 2: Se a decisão ocorreu na linha 38, é porque ele iniciou a fase de decisão pela invocação à função $take\_decision()$ e enviou sua estimativa como parâmetro para o consenso. E para que o valor de decisão tenha sido `ABORT`, a variável *decision* só foi atribuída na linha 28 do Algoritmo 7, cuja rodada de decisão era superior à 1 (i.e., $round > 1$). Se a rodada é maior que 1, significa que em rodadas anteriores o valor não pôde ser decidido em virtude da suspeição de falha(s) nas rodadas que precederam à rodada em que o valor foi decido.

Pela especificação da fase de decisão no Algoritmo 7, o valor `COMMIT` só é decidido na primeira rodada de decisão (p. ex.: $round = 1$), desde que não tenha havido suspeitas de falhas e todos os votos recebidos foram `sim` – um cenário favorável especificado nas condições da linha 21. Se a decisão não ocorre na primeira rodada, é porque houve alguma suspeita de falhas e a estimativa de algum CTM não foi recebida. Ademais, a evidência de que o valor `COMMIT` só pode ser decidido se todos votaram `sim` decorre da prova demonstrada para o Teorema 5.2. Logo, o teorema em questão segue comprovado. □

5.8 CONSIDERAÇÕES DO CAPÍTULO

As contribuições que deram origem a este capítulo foram oriundas da observação de que, tecnologias modernas encontradas na atualidade podem prover um suporte factível para a tolerância a faltas, principalmente para aplicações distribuídas com semântica transacional. E com base neste pressuposto, duas contribuições foram introduzidas para área de sistemas de computação distribuída.

A primeira consiste na proposição de um modelo arquitetural especificado sob uma abordagem pragmática e realista, baseada em tecnologia de virtualização, pelo qual é possível especificar soluções baseadas em um modelo híbrido de faltas, com a admissão de diferentes hipóteses para cada nível do modelo arquitetural. À primeira vista, acredita-se que a proposição deste modelo arquitetural representa um

passo inicial para que soluções algorítmicas de cunhos práticos e voltadas para ambientes reais, possam vir a ser desenvolvidas. Não obstante, por se basear numa abordagem realista e empregar o uso de tecnologia moderna, a ideia proposta pode ser vista como um avanço para a área de modelos de computação para a especificação de algoritmos distribuídos tolerante a faltas. Do mesmo modo, o modelo é flexível ao ponto de ser passível de generalização e adaptação, em termos de sua especificação e implementação para uma arquitetura modular. O que, por conseguinte, viria a possibilitar o desenvolvimento de soluções para resolver diversos problemas práticos e teóricos verificados na concepção de sistemas de computação distribuída tolerantes a faltas.

A segunda contribuição trazida pelo presente capítulo foi a proposição de uma solução inédita na literatura, através da abordagem do problema da validação atômica fraca não-bloqueante sobre um modelo híbrido de falhas com vista para tolerar faltas bizantinas nos participantes do protocolo de terminação. Os poucos trabalhos que envolvem a terminação de transações e faltas bizantinas em um sistema de computação distribuída, não seguem a especificação do NBAC, e portanto, não atendem aos requisitos estipulados para o problema (MOHAN; STRONG; FINKELSTEIN, 1983; ZHAO, 2007). A separação dos elementos presentes nos sítios participantes de um protocolo NBWAC através da noção de agentes diferentes (i.e., TM e DM), que passam a operar em níveis distintos do modelo arquitetural proposto (p. ex.: anfitrião e convidado), se mostrou o meio factível para especificar a solução para o NBWAC, de modo a permitir a sujeição a faltas bizantinas sobre os agentes abrigados pelos sítios participantes do protocolo de terminação.

Além disso, diferente de todos os trabalhos que tratam de mesmo problema na literatura (HADZILACOS, 1990; BABAOGLU; TOUEG, 1993; GUERRAOUI; LARREA; SCHIPER, 1995; GUERRAOUI; SCHIPER, 1995a; GUERRAOUI, 1995; GUERRAOUI; LARREA; SCHIPER, 1996; ABDALLAH; PUCHERAL, 1999; GREVE; NARZUL, 2002; PARK; LEE; YU, 2013), a solução introduziu um novo agente para a resolução do NBWAC com faltas bizantinas, o CTM (*Collaborative Transaction Manager*), a fim de eliminar a dependência sobre o TM, por parte de um sítio participante do protocolo. Para tanto, o protocolo distribui a responsabilidade de um TM entre um conjunto de réplicas, que passam a atuar de maneira colaborativa no que diz respeito às ações realizadas pelo TM de um sítio, o que, portanto, dá origem ao agente CTM. O CTM atua de maneira pró-ativa na detecção e confinamento de faltas bizantinas ocorridas no âmbito dos TMs de um sítio, onde ele atesta e certifica todas as ações que são externalizadas pelo sítio para DM do mesmo sítio e para os

CTMs dos demais sítios. Com isso, as faltas bizantinas ocorridas no âmbito dos TMs de um sítio ficam confinadas apenas a eles, de tal maneira que eles não conseguem forjar ações e tampouco deturpar a execução do protocolo de terminação de transações.

E finalmente, devido a especificação e generalidade do modelo arquitetural proposto, tem-se a convicção de que este pode vir a ser adotado como base para diversos trabalhos futuros envolvendo problemas de acordo em sistemas distribuídos.

6 AVALIAÇÃO DAS SOLUÇÕES PROPOSTAS

Este capítulo descreve os resultados e discussões acerca das contribuições introduzidas por esta tese. Neste sentido, é realizada a validação das propostas apresentadas e formalizadas nos capítulos anteriores do documento, cuja verificação ocorre de duas maneiras, isto é, por meio das avaliações analítica e experimental. Nas seções a seguir, são descritos os cenários em que foram realizados os experimentos, bem como os parâmetros e configurações utilizadas em cada ambiente de execução, para as respectivas soluções.

6.1 AVALIAÇÃO DO PROTOCOLO DE REPLICAÇÃO DE BANCOS DE DADOS TOLERANTE A FALTAS BIZANTINAS

Esta seção descreve os resultados verificados através de avaliações efetuadas sobre o protocolo apresentado no Capítulo 4 desta tese. A avaliação do protocolo é realizada de duas maneiras, onde a primeira ocorre a partir da abordagem analítica, no intuito de verificar a eficiência do protocolo proposto em relação aos trabalhos existentes na literatura; e a segunda se dá de maneira experimental, a fim de avaliar se o protocolo é factível em termos de implementação. É digno de nota que as avaliações foram realizadas nestes moldes, por estarem em consonância com a literatura que trata da avaliação de desempenho de sistemas computacionais, a qual destaca os métodos analítico e empírico/experimental como os mais usuais a serem empregados para a avaliação destes sistemas (JAIN, 1991). Para ambas as avaliações, em razão de não ter sido sugerido um nome para o protocolo proposto, o mesmo é designado pelo acrônimo PP – de **P**rotocolo **P**roposto. No que segue, são apresentados os detalhes envolvidos em cada processo de avaliação, bem como os respectivos resultados obtidos em cada uma delas.

6.1.1 Avaliação Analítica

A abordagem analítica para a avaliação de sistemas computacionais é fundamentada em aspectos teóricos, cujo propósito é medir a eficiência de um sistema através do estudo das complexidades algorítmicas verificadas para este (JAIN, 1991). Neste caso, a avaliação apre-

sentada nesta seção visa demonstrar a eficiência do protocolo proposto, em termos das complexidades envolvidas e das propriedades requeridas para o funcionamento do mesmo. Para tanto, a análise é realizada acerca dos custos envolvidos no processamento de uma transação, os quais são verificados a partir de execuções normais do protocolo, isto é, aquelas onde não há a ocorrência de faltas. Isto porque, segundo a literatura, o caso normal é o mais comum e recorrente na execução de um protocolo tolerante a faltas (CASTRO; LISKOV, 1999; YIN et al., 2003; KOTLA et al., 2007; GUERRAOUI et al., 2010).

Para a avaliação analítica, além do protocolo proposto nesta tese (i.e., PP), são analisados os protocolos de replicação do HRDB (VANDIVER et al., 2007), do Byzantium (GARCIA; RODRIGUES; PREGUIÇA, 2011) e do BFT-DUR (PEDONE; SCHIPER; ARMENDÁRIZ-IÑIGO, 2011; PEDONE; SCHIPER, 2012), em razão deles compartilharem os mesmos princípios de funcionamento, e terem sido especificados para resolver problemas envolvendo transações e faltas bizantinas. Neste sentido, a tabela 6 enumera alguns dos resultados observados para cada um dos protocolos avaliados, tendo em conta todo o processamento da transação, desde o início até o término destas em cada protocolo. Note que além da avaliação acerca dos custos envolvidos no processamento da transação, é de interesse avaliar algumas características presentes nos protocolos BFT para replicação de bancos de dados. As expressões apresentadas como resultados para a latência compreendem o número de mensagens requeridas durante todo o processamento da transação, nas quais o termo δ denota o número de operações declaradas numa transação.

Tabela 6 – Avaliação acerca do processamento da transação.

Aspecto \ Protocolo	**PP**	**HRDB**	**Byzantium**	**BFT-DUR**
Número – Réplicas	$3f+1$	$2f+1+$ *controlador*	$3f+1$	$3f+1$
Latência – Comunicação	$2\delta+2(TOMcast)+2$	$6\delta+3$	$2\delta+2(TOMcast)+2$	$2\delta^{*}+(TOMCast)+1$
Número – Mensagens	$2\delta+2(TOMcast)+2n$	$\delta(2n+2)+2+n$	$\delta(n+1)+2(TOMcast)+2n$	$2\delta^{*}+(TOMCast)+n$
Complexidade	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n^2)$
Consistência	serialização	serialização	*snapshot isolation*	serialização
Controle	distribuído	centralizado	distribuído	distribuído
Banco de Dados	relacional	relacional	relacional	chave/valor

A partir dos resultados reportados na Tabela 6, se verifica que, em termos de número de réplicas necessárias para a execução dos protocolos, o HRDB apresenta a melhor resiliência. Este feito decorre do fato do HRDB ser dependente de um controle centralizado, o coordenador, para realizar a ordenação das transações, de acordo com

algumas regras pré-estabelecidas, para que seja possível assegurar a corretude do protocolo e a consistência do banco de dados. Por esta razão, não é requerida a execução de um acordo bizantino para ordenar as transações para execução pelas réplicas, tal como ocorre nos demais protocolos, pois o coordenador é quem define a ordem das transações. No caso, apenas $2f + 1$ réplicas são necessárias para assegurar a validade dos resultados, por meio de um mecanismo de voto majoritário. Por outro lado, é importante verificar que uma falha do coordenador inviabiliza toda a execução do HRDB, de modo que o sistema permanecerá bloqueado até a recuperação do mesmo, situação que não ocorre nos demais protocolos. Note que, com exceção do HRDB, o protocolo proposto (PP) e os demais avaliados requerem um número de réplicas considerado como mínimo para o acordo bizantino (i.e., resiliência ótima), nos termos da literatura (PEASE; SHOSTAK; LAMPORT, 1980; LAMPORT; SHOSTAK; PEASE, 1982).

De outro modo, para analisar as métricas de desempenho usuais da algorítmica distribuída, tais como a latência (ou passos) de comunicação, número e complexidade de mensagens (LYNCH, 1996), para os protocolos referenciados na Tabela 6, alguns aspectos merecem atenção. Note que, embora similares, a latência representa o número de passos de comunicação requeridos durante o processamento da transação, enquanto a outra se refere ao número de mensagens enviadas no intervalo de execução da transação, pela passagem por todos os passos de comunicação. Em termos de latência, o protocolo proposto e o Byzantium apresentam os mesmos resultados, pois o padrão de comunicação empregado em ambos é bastante similar. Neste quesito, o protocolo que apresenta o melhor desempenho é o HRDB, pelo fato de não haver acordo entre as réplicas. No caso do BFT-DUR, o asterisco (*) foi incluído na expressão porque cabe uma observação, os dois passos de comunicação são requeridos apenas para as operações de leitura, pois as operações de escrita não são realizadas sobre as réplicas durante o processamento da transação. Por outro lado, a despeito das latências verificadas, se considerado o número de mensagens trocadas no decurso de uma transação, a eficiência dos protocolos varia em relação ao número de operações executadas pela transação, isto é, em determinados casos, um protocolo é melhor, e noutros casos é pior.

Como exemplo para tal afirmação, considere uma transação com 50 operações e uma configuração onde $f = 1$ e n igual ao número de réplicas daquele protocolo (p. ex.: $2f + 1$, $3f + 1$). Para o cômputo das mensagens, considere também o Paxos bizantino (ZIELINSKI, 2004) como protocolo subjacente de difusão com ordem total BFT (i.e., o

TOMcast da expressão). Neste caso, tem-se os seguintes resultados: HRDB = 405 mensagens, Byzantium = 314 mensagens, BFT-DUR = 132 mensagens e PP = 164 mensagens. Note que nesta situação, o protocolo proposto é menos eficiente apenas em relação ao BFT-DUR, o que se justifica pelo tipo de banco de dados usado. Por outro lado, para uma transação com apenas 5 operações, os resultados são os seguintes: HRDB = 45 mensagens, Byzantium = 89 mensagens, BFT-DUR = 42 mensagens e PP = 74 mensagens. Em tal situação, o protocolo proposto é o terceiro melhor em termos de eficiência, sendo o BFT-DUR o mais eficiente, seguido pelo HRDB.

Em se tratando da complexidade de mensagens, com exceção do HRDB, que apresenta uma complexidade linear, todos os demais apresentam complexidades quadráticas (i.e., n^2), o que decorre do uso dos protocolos de difusão com ordem total tolerantes a faltas bizantinas. Em termos de nível/critério de consistência assegurado, embora todos os protocolos provejam critérios considerados como fortes (*strong consistency*) (BERENSON et al., 1995), o ***snapshot isolation*** provido pelo Byzantium é um pouco mais fraco em relação ao **serializável**, e sob determinadas circunstâncias ele pode violar a consistência e a integridade do banco de dados – conforme explicado na Seção 2.1.4.1. Em relação ao controle, o HRDB é o único que depende de um elemento central para conduzir o protocolo, razão pela qual ele se apresenta como mais eficiente em determinados aspectos. Por fim, o BFT-DUR é o único a não operar sobre bancos de dados relacionais, e este é um dos motivos que o leva a apresentar a melhor latência para o processamento da transação.

Embora o protocolo proposto não seja o melhor em todos os aspectos verificados acerca da eficiência, ele se mostra bastante atrativo como solução para replicação de bancos de dados relacionais. Note que os únicos aspectos avaliados nos quais ele é inferior são justificados pelas escolhas dos modelos de sistemas daqueles protocolos que são avaliados como sendo mais eficientes – que são sensivelmente diferentes do modelo de sistema adotado para a proposição do protocolo desta tese.

6.1.2 Avaliação Experimental

Nesta seção se apresenta os resultados obtidos pela realização do protótipo do protocolo de replicação, através da implementação do *middleware* apresentado na Seção 4.8. Neste sentido, além dos resultados, são descritos os cenários em que foram realizados os experimentos,

os parâmetros e as configurações utilizadas em cada cenário. Maiores detalhes sobre a especificação dos elementos que compõem a arquitetura de *middleware* e sua implementação estão disponíveis na Seção 4.8.

6.1.2.1 Metodologia

No intuito de avaliar a implicação do mecanismos para tolerância a faltas bizantinas provido pelo protocolo proposto, sobre um ambiente real de banco de dados, a mensuração dos resultados foi realizada a partir da implementação do *benchmark* TPC-C (TPC, 2014). O TPC-C pode ser considerado um padrão no que diz respeito à avaliação de desempenho de sistemas baseados no processamento de transações, pois ele testa muitas das funcionalidades de um banco de dados (p. ex.: atualizações/escritas, leituras/consultas, etc.). Além disso, uma carga de trabalho real, como é o caso do TPC-C, é entendida na literatura como uma alternativa bastante plausível, no que concerne à avaliação do desempenho de um sistema (JAIN, 1991). Não obstante, o próprio consórcio responsável pela sua especificação (i.e., o TPC) o considera como um dos mais indicados e confiáveis para a mensuração do desempenho de transações em bancos de dados, dada sua semelhança ao ambiente de um sistema real de computação baseado no processamento de transações de negócio. A carga provida pelo TPC-C é composta por cinco transações de leitura e de escrita, as quais simulam as atividades típicas encontradas em um ambiente de negócio. A finalidade por trás do TPC-C, é fornecer dados relevantes que permitam uma comparação objetiva acerca do desempenho de um sistema computacional, quanto à sua capacidade em processar transações. Para tanto, a medição a ser aferida através do TPC-C, deve ser realizada em termos de transações por minuto (tpm).

O TPC-C define um *schema* para o banco de dados a ser usado nos experimentos, sendo este composto por nove tabelas, as quais são referenciadas/usadas pelas transações implementadas para o *benchmark*. A Figura 46 apresenta o digrama de Entidades e Relacionamentos (ER) que é implementado como *schema* para o TPC-C. Note que o banco de dados é composto por nove tabelas, onde cada uma delas mantém alguma restrição de integridade referencial com outra(s), o que pode ser observado pela cardinalidade de cada relacionamento (p. ex.: um-para-muitos, um-para-um, etc.).

Em se tratando das transações, elas são especificadas a partir de

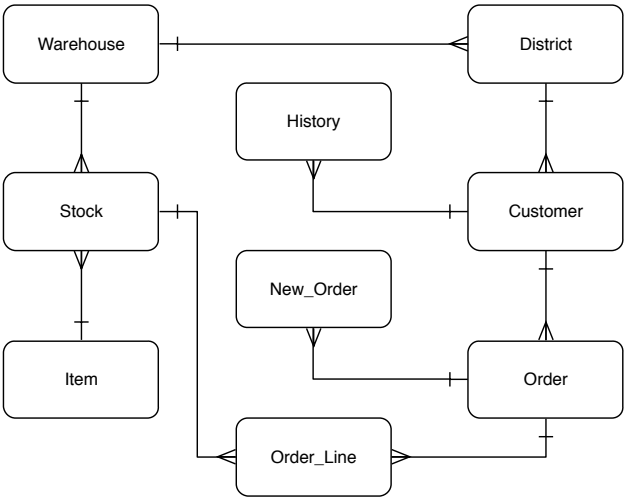


Figura 46 – Diagrama ER para o *schema* do TPC-C.

cinco tipos de operações, que são executadas segundo uma distribuição probabilística. As transações e suas respectivas probabilidades de execução são as seguintes:

- **New Order**: transação de leitura/escrita que acrescenta uma nova operação de compra de um cliente no sistema – (45%);
- **Payment**: transação de leitura/escrita que efetua o registro do pagamento de uma compra de um cliente – (43%);
- **Delivery**: transação de leitura/escrita que registra uma entrega de produtos, para um lote de x pedidos, sendo x proporcional ao número de distritos – (4%);
- **Order Status**: transação somente-leitura que é executada para verificar a situação da última compra efetuar por um cliente – (4%);
- **Stock Level**: transação somente-leitura que é executada para verificar se os níveis de estoque dos itens vendidos recentemente está abaixo do limite – (4%).

Para a realização dos experimentos, cada transação do *benchmark* foi implementada como um método de uma classe Java, nos

quais cada operação efetuada por aquela transação requer uma interação com o banco de dados, isto é, as operações não são enviadas em lotes ao banco de dados. Em se tratando dos experimentos, para se obter resultados fidedignos, cada experimento foi repetido 14 vezes, com vista para um grau de confiança de 90% (i.e., $z = 1,645$). O número de amostras para o grau de confiança desejado, foi estimado a partir de cálculos realizados pelas equações apresentadas abaixo, as quais foram obtidas em (RYAN, 2009). No entanto, para que fosse possível estimar o tamanho da amostra (n) para obter o grau de confiança desejado (z), alguns parâmetros tiveram que ser obtidos a partir da execução dos mesmos experimentos numa amostra piloto (i.e., com um tamanho amostral arbitrário) (RYAN, 2009).

$$s = \sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n} - \left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^2} \qquad E = \frac{s}{\sqrt{n}} \qquad n = \left(z\frac{s}{E}\right)^2 \quad (6.1)$$

Cabe salientar, que, embora cada cenário de experimento tenha sido repetidos 14 vezes, um cenário de experimento executa um número suficiente de transações – i.e., cada cliente submete 15.000 transações por experimento –, no intuito de prover confiabilidade e validade aos resultados. À vista disso, os resultados que serão apresentados na próxima seção representam a média obtida pelas 14 execuções de cada cenário de experimento.

6.1.2.2 Resultados Obtidos

Para coletar as amostras de dados decorrentes da avaliação experimental do protótipo, foi utilizado um ambiente computacional composto por 6 máquinas, das quais 4 foram utilizadas para executar o código das réplicas e 2 para executar o código dos clientes. Nesta avaliação, os testes foram realizados num cenário onde $f = 1$, para um total de $n = 3f + 1$ réplicas. A opção por realizar experimentos apenas com $f = 1$, decorreu do fato de que este é o cenário recorrente da literatura, e também por compreender que este é o suficiente para mascarar as faltas ocorridas num ambiente real, nos termos discutidos pela literatura (CASTRO; LISKOV, 1999; KOTLA et al., 2007; GASHI; POPOV; STRIGINI, 2007). As 4 (quatro) máquinas utilizadas como réplicas nos experimentos foram equipadas exatamente com os mesmos componentes de *hardware*, tendo a seguinte configuração: 1 (um) microproces-

sador Intel®Core™i7 3,50Ghz com quatro núcleos; 12GB de memória RAM; 1 (um) disco rígido de 750GB Seagate Barracuda ST3750525AS, 7200 RPM, 32MB de Cache, Interface SATA III 6.0Gb/s; Interface Ethernet Gigabit. As 2 (duas) máquinas utilizadas para executar os clientes tinham as seguintes configurações: 1 (um) microprocessador Intel®Core™2 Duo 3,0Ghz com dois núcleos; 4GB de memória RAM; Interface Ethernet Gigabit. Todas as máquinas envolvidas nos experimentos foram interligadas por um comutador (*switch*) D-Link DGS-3100-24P Ethernet 10/100/1000.

Em termos de *software*, nas 4 máquinas utilizadas como réplicas foi instalado o sistema operacional Debian 7.0 ("wheezy"), kernel 3.2.0-4-amd64 #1 SMP x86_64 GNU/Linux, tendo o Ext4 como sistema de arquivos; e nas 2 máquinas utilizadas para os clientes foi instalado o sistema operacional Ubuntu 12.04.4 LTS, kernel 3.11.0-15-generic #25 ~precise1-Ubuntu SMP i686 i386 GNU/Linux. Como o protótipo foi implementado na linguagem Java, nas 4 (quatro) máquinas utilizadas como réplicas, instalou-se o JDK Oracle 1.7.0_45 64-Bit, enquanto que nas máquinas utilizadas para os clientes foi instalada a mesma versão de JDK, porém, na compilação de 32-Bit. Outrossim, o SGBD utilizado pelas réplicas para o provimento dos bancos de dados foi o MySQL Community Server, sendo que em duas máquinas utilizou-se a versão 5.5.34 e noutras duas utilizou-se a versão 5.5.39. O *driver* JDBC nativo utilizado pelo *middleware* em nível de réplicas, foi o MySQL Connector/J versão 5.1.23. Como mecanismo de armazenamento, utilizou-se o InnoDB – um mecanismo de armazenamento transacional estável –, com ajustes em alguns de seus parâmetros para prover um melhor desempenho, são eles: innodb_file_per_table: ON, innodb_flush_log_at_trx_commit: 0, innodb_open_files: 1000, innodb_log_file_size: 100MB e innodb_buffer_pool_size: 2GB. O *schema* de banco de dados criado para o TPC-C, em cada réplica, foi populado com 5 *warehouses* e 10 distritos para cada *warehouse*, sendo que as demais tabelas foram populadas de acordo com os valores padrões (TPC, 2014). E para prover um melhor desempenho sobre a execução das operações no banco de dados, para todas as tabelas contidas no *schema* do TPC-C, foram coletadas as respectivas estatísticas antes da execução de cada experimento.

Note que os propósitos da realização destes experimentos estão centrados em dois aspectos: (*i*) na avaliação do custo relacionado à adição do suporte tolerante a faltas bizantinas em um ambiente replicado de banco de dados (*ii*) na avaliação da eficiência do protocolo proposto, em termos de sua implementação. Neste sentido, para permitir uma

comparação e mensurar o efeito causado pela replicação sobre o SGBD, o *benchmark* foi executado sobre: (*i*) um banco de dados MySQL sem replicação, usado como linha de base; (*ii*) sobre o protocolo proposto, sem o uso de replicação (i.e.: apenas *proxy*); e, (*iii*) sobre o protocolo proposto com replicação. Na execução direta sobre o MySQL, o cliente se conecta ao SGBD através do *driver* JDBC nativo. Note que uma comparação acerca dos resultados obtidos para o protocolo proposto, nos cenários sem o uso de replicação e com replicação é algo interessante, do ponto de vista de que o primeiro representa o melhor caso, e portanto, o melhor resultado que se pode esperar numa execução do protocolo. No caso, as execuções sobre o *proxy* do protocolo proposto não executam a fase de terminação, já que não há replicação naquele experimento. Os resultados obtidos estão divididos em dois cenários, no primeiro são apresentados os valores das amostras obtidas sem a ocorrência de faltas durante a execução dos experimentos, e o segundo apresenta os valores para as amostras obtidas e um cenário em que houve a ocorrência de faltas.

Experimentos sem Faltas

Os primeiros experimentos foram realizados sobre um cenário com a ausência de faltas, isto é, o caso mais favorável. Este experimento foi realizado com base na carga padrão do TPC-C, na qual há o predomínio da execução de transações de leitura/escrita, sendo 92% de transações leitura/escrita e 8% de transações somente/leitura. Para tanto, diversas execuções foram realizadas com variação no número de clientes, em múltiplos de 5, no intervalo de 1 a 60. Com este primeiro experimento, o interesse é avaliar o desempenho do protocolo proposto em meio à três situações: a capacidade de processamento em transações por minuto; o tempo médio de execução das transações; e a taxa média de anulações, pois, como se trata de um protocolo otimista, é natural que ocorram anulações não-espontâneas, por preempção, em decorrência de conflitos (KUNG; ROBINSON, 1981).

Neste sentido, com o propósito de mensurar o efeito que o uso do *middleware* implementado para o protocolo proposto causa no desempenho do sistema, foi realizada a comparação do desempenho da execução do TPC-C diretamente sobre o MySQL – a partir do *driver* JDBC nativo –, em relação ao protótipo de `middleware` implementado. À vista disso, as execuções para o TPC-C a partir do protótipo implementado foram realizadas de duas maneiras: pelo *Proxy*, que consiste na implementação da ponte JDBC para o ambiente de banco de dados

replicado, porém com conexão apenas sobre uma réplica, e portanto, sem o uso de replicação; e pelo *middleware* implementado para o protocolo proposto (PP). É digno de nota que todos os experimentos não consideram operações em lotes (cfm. Seção 6.1.2.1).

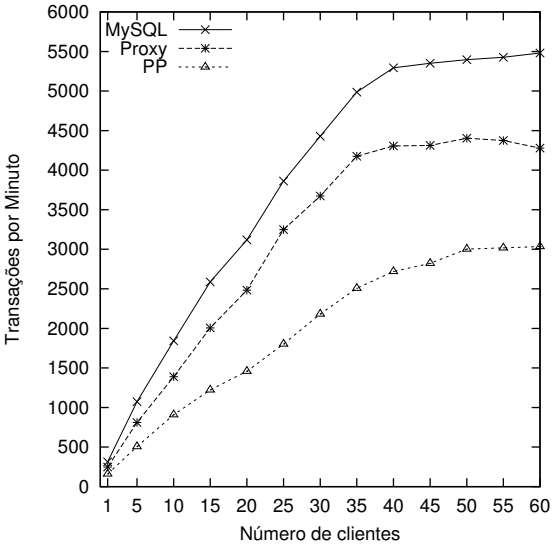


Figura 47 – Desempenho verificado para a carga padrão do TPC-C.

Os resultados obtidos em termos de vazão (i.e., *throughput*) para este primeiro experimento são reportados na Figura 47, onde, nos moldes da especificação do TPC-C (TPC, 2014), o desempenho é medido em transações por minuto (tpm). Os resultados da Figura 47 demonstram que o custo adicional imposto pela replicação do banco de dados, através do protocolo proposto, é em torno de 53%. Note que estes valores são obtidos quando comparados aos resultados da execução do TPC-C diretamente sobre o MySQL. Do mesmo modo, o custo verificado para o *Proxy* em relação ao JDBC nativo do MySQL é de aproximadamente 20%. Em se tratando dos resultados obtidos para o *Proxy*, eles são justificados pelo seguinte, a conexão efetuada pela interface JDBC provida pelo *Proxy* realiza tarefas que não ocorrem em uma conexão nativa JDBC, as quais são necessárias para prover o suporte à replicação do banco de dados, e portanto, para tolerar as faltas bizantinas. No caso da execução de operações da transação, que compreende aos valores re-

portados para o *Proxy* na Figura 47, o custo adicional decorre, principalmente, da manutenção do *write-set* da transação para cada operação de escrita e leitura/escrita executada no âmbito da transação. Uma tarefa que não é realizada por um *driver* JDBC nativo (i.e., sem suporte à replicação), em que a execução da operação de escrita ou leitura/escrita retorna apenas um booleano (i.e., pelo método *executeUpdate* da classe *Statement*). A interface JDBC implementada pelo *Proxy*, retorna, além do valor booleano, um *hash* mapeado sobre os itens de dados afetados pela operação execucada na réplica. Isso é necessário, de acordo com a especificação do protocolo, para permitir a comparação/validação dos resultados obtidos na execução da operação, com aqueles que serão obtidos para a validação da transação junto às demais réplicas, quando da execução do protocolo de terminação. Este *hash* é calculado a partir da interface *ResultSet* implementada pelo *Proxy*, a qual é sensivelmente diferente daquela tradicionalmente implementada pelos *drivers* JDBC nativos dos SGBDs. O *ResultSet* implementado pelo *middleware* mapeia todos os resultados de uma operação em uma cadeia de *bytes* de tamanho fixo, para prover uma representação uniforme dos tipos de dados em um ambiente heterogêneo.

Por outro lado, ao observar os resultados para o protocolo de replicação, a diferença de desempenho ocorre por dois fatores, o primeiro é devido aos custos impostos pelo *Proxy* já explicados anteriormente. O segundo e principal fator, decorre da execução do protocolo de terminação, o qual é parte da etapa que realiza a validação da transação, e portanto, é onde são realizadas as duas verificações requeridas para a terminação com êxito, da transação. A primeira verificação, que compreende ao teste de certificação, é onde ocorrem as verificações sobre os controles adicionais introduzido pelo protocolo de replicação, para assegurar a serialização e a consistência da transação, a despeito dos efeitos da concorrência e de faltas. Neste ponto, a transação é confrontada com os *write-set*s das transações que não precedem a transação em questão (cfm. Seção 4.5.3.3), para verificar se ela é uma transação serializável e pode ser validada e incluída no histórico de transações. A segunda verificação é necessária para assegurar a corretude/exatidão quanto à execução otimista daquela transação, em que todas as operações da transação são re-executadas, mas apenas nas réplicas não-líderes daquela transação. Em se tratando da sujeição à faltas bizantinas, este é um passo necessário para verificar se os resultados obtidos são corretos e não violam a consistência e a integridade do banco de dados. No entanto, se considerado estes aspectos em relação aos resultados obtidos, ambos sugerem que o custo de replicação verificado para o protocolo

se mantém constante, o que corrobora para que o grau de concorrência do protocolo proposto seja equivalente aquele obtido por um banco de dados não replicado, a despeito dos desempenhos verificados. Já em relação ao desempenho do *Proxy* quando comparado ao protocolo de replicação, o custo da replicação é superior em aproximadamente 20%.

À luz dos resultados obtidos em termos de vazão, um aspecto importante e que também requer avaliação é a latência verificada na execução das transações. A latência representa o tempo médio que é percebido pelo cliente, para o processamento de uma transação. A latência verificada para o TPC-C é determinada por meio de uma equação, na qual alguns valores obtidos para a vazão são empregados. Para um cenário de execução onde C é o número de clientes que obtém uma vazão T (em transações por minuto), a latência L (em milissegundos) é dada pelo inverso da média do *throughput* (TPC, 2014), a qual é expressa pela seguinte equação:

$$L = \left(\frac{60 * C}{T} \right) * 1000 \tag{6.2}$$

Os resultados verificados para a latência são reportados na Figura 48, cujos valores representam os tempos médios consumidos para o processamento das transações, e observado pelos clientes. Este valor compreende ao intervalo decorrido desde o início da transação (i.e., submissão da operação `BEGIN`), até a conclusão do protocolo de terminação e recepção da(s) notificação(ões) de conclusão pela validação ou anulação. Evidentemente que os valores crescem na medida em que se aumenta o número de clientes, e de maneira proporcional ao número de transações processadas por minuto. Note que o crescimento da latência não é decorrente do aumento da utilização dos recursos por parte das réplicas, já que o *throughput* se mantém em crescimento, conforme verificado na Figura 47. Se analisada a latência para cada protocolo, individualmente, o crescimento da latência em função do aumento da concorrência é suave. Este crescimento pode ser atribuído, principalmente, pelos tempos gastos por atividades requeridas para o gerenciamento das transações simultâneas, tais como trocas de contexto e gerenciamento de bloqueios para o controle de concorrência. O que também pode corroborar para estes resultados, é o fato de que os processos que executaram os clientes foram distribuídos em apenas duas máquinas distintas (i.e., por meio de *threads*), de modo que, em partes, se pode atribuir este crescimento também ao escalonamento das *threads* em cada uma das máquina. Ademais, em se tratando dos resultados verificados para o protocolo proposto, há um custo adicional sobre a la-

tência, o qual é introduzido pelo protocolo de terminação, cujas tarefas ocorrem de maneira distribuída, pelo uso de um protocolo subjacente de difusão com ordem total.

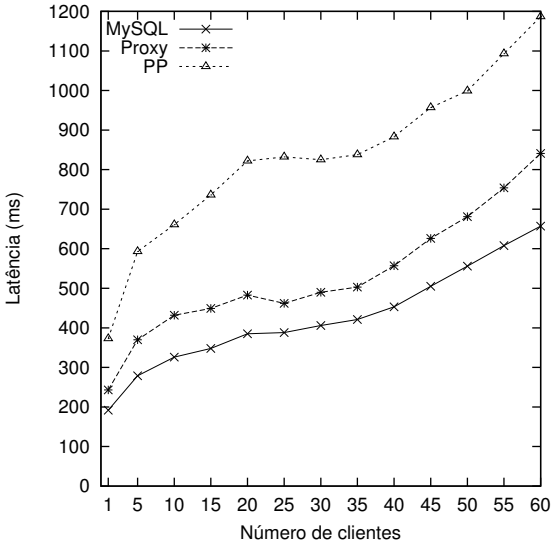


Figura 48 – Latência observada para a carga padrão do TPC-C.

Por fim, um aspecto que também dever ser analisado acerca do TPC-C e também de protocolos de replicação otimista – que é o caso do protocolo proposto –, é em relação às transações anuladas, já que parcela de transações executadas de maneira otimista tendem a sofrer anulação não-espontânea por preempção e/ou conflitos (KUNG; ROBINSON, 1981). Neste caso, deve-se considerar, além do *throughput* total do TPC-C (em tpm), o *throughput* verificado apenas para aquelas transações que foram concluídas pela validação. O cruzamento destes valores resulta na taxa de anulações obtida pela execução do TPC-C. A Figura 49 apresenta estes números, que são complementados com os dados apresentados na Tabela 7. A Tabela 7 enumera os resultados que representam a taxa média de anulações ocorridas durante a execução deste cenário de experimentos.

É importante ressaltar, que em qualquer execução do *benchmark* TPC-C, sempre é esperado que parcela de transações não seja bem sucedida (TPC, 2014), o que implica que um percentual de transações

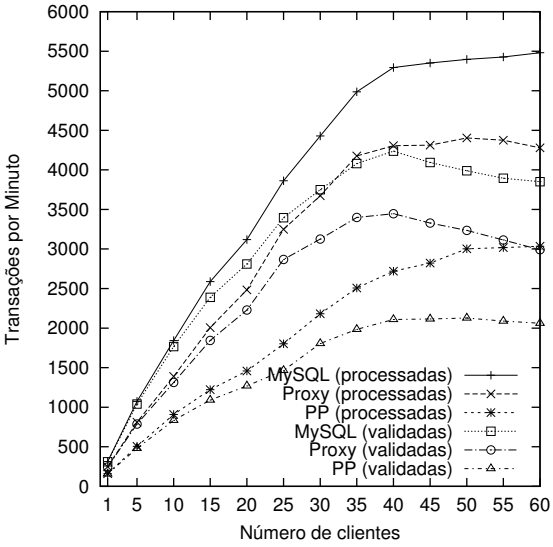


Figura 49 – Transações processadas vs. validadas no TPC-C.

sempre é anulada. Do mesmo modo, na medida em que ocorre o aumento no número de clientes com acesso simultâneo ao banco de dados, também é esperado um aumento no número de transações anuladas, e consequentemente, uma redução do número de transações bem sucedidas em suas execuções – o que também é um aspecto inerente ao *benchmark* TPC-C. Por outro lado, o que deve ser analisado acerca dos resultados são as diferenças entres os percentuais obtidos para cada execução realizada, isto é, para o MySQL, para o *Proxy* e para o ambiente replicado através do protocolo proposto. Neste caso, uma análise sobre os resultados demonstram que, para cada execução realizada, a diferença entre os valores das anulações resultantes das execuções do MySQL e do *Proxy* são muito próximas e inferiores a 1% (um por cento). Embora o desempenho do *Proxy* seja sensivelmente inferior ao da conexão nativa do MySQL, a taxa de anulações entre eles se mantém muito próximas. De outro modo, ao verificar os mesmos resultados para o protocolo proposto, nota-se um acréscimo entre 3% e 4% em relação aos resultados obtidos para o MySQL. E por conseguinte, se pode concluir que a despeito da complexidade introduzida pelo protocolo de replicação e de sua natureza otimista, estas não afetam em potencial o

processamento de transações, induzindo-as a excessivas anulações, pelo menos para a carga padrão do TPC-C.

Tabela 7 – Anulações ocorridas pela execução do TPC-C.

Transações executadas pelo *benchmark* TPC-C									
	# Processadas			# Validadas			% Anulação		
# clientes	MySQL	*Proxy*	PP	MySQL	*Proxy*	PP	MySQL	*Proxy*	PP
1	313	247	161	310	244	156	0,982%	1,035%	3,116%
5	1074	811	506	1041	788	480	3,041%	2,861%	5,083%
10	1842	1389	908	1766	1313	835	4,117%	5,464%	8,009%
15	2587	2006	1223	2389	1844	1090	7,665%	8,087%	10,913%
20	3119	2484	1459	2810	2229	1269	9,893%	10,276%	13,026%
25	3862	3249	1802	3396	2867	1551	12,072%	11,768%	13,919%
30	4429	3673	2181	3750	3126	1805	15,336%	14,893%	17,245%
35	4986	4176	2507	4079	3398	1985	18,194%	18,623%	20,833%
40	5293	4306	2719	4236	3446	2109	19,971%	19,974%	22,418%
45	5351	4312	2822	4093	3328	2115	23,518%	22,823%	25,061%
50	5397	4403	3002	3989	3236	2128	26,092%	26,505%	29,124%
55	5426	4375	3018	3895	3113	2089	28,218%	28,841%	30,784%
60	5481	4279	3034	3851	2991	2061	29,731%	30,096%	32,072%

Porém, é importante salientar que um cenário com muitos conflitos pode ser agravado com o crescimento do número de clientes em acesso simultâneo ao banco de dados. Uma discussão sobre os resultados demonstrados pelas Figuras 47, 48 e 49, demonstram que o protocolo proposto é sensível ao aumento do número de clientes, visto que na medida em que é acrescido o número de clientes e também das operações de leitura e de escrita em uma transação, a probabilidade de conflitos aumenta devido à contenção sobre os itens de dados do banco de dados, o que também pode resultar num acréscimo em termos de anulações na fase de terminação. No entanto, note que esta situação é mais agravada em se tratando de um ambiente que utiliza um banco de dados de pequena dimensão, de modo que ela tende a ser reduzida na medida em que se aumenta a dimensão de um banco de dados – nestes os conflitos são mais raros.

Por outro lado, a despeito dos bons resultados verificados na literatura para alguns protocolos de replicação de bancos de dado que toleram faltas por **parada**, principalmente no tocante a escalabilidade pelo acréscimo de réplicas, como é o caso da DBSM (PEDONE; GUERRAOUI; SCHIPER, 2003) e do C-JDBC (CECCHET; MARGUERITE; ZWAENEPOEL, 2004); infelizmente estes resultados não são reproduzidos e/ou verificados em protocolos de replicação de bancos de dados que toleram faltas bizantinas. Visto que protocolos de replicação que toleram apenas faltas por parada não preveem que réplicas falhem de maneira arbi-

trária, isso implica que o resultado obtido durante o processamento da transação é suficiente para assegurar a corretude/exatidão da mesma. Esta é a razão pela qual os protocolos de terminação de solução para replicação tolerantes a faltas por parada, apenas certificam a transação quanto aos possíveis conflitos, e não quanto à exatidão/validade dos resultados. No caso de faltas bizantinas, é impossível assegurar a corretude/exatidão e consistência de uma transação, sem que os resultados obtidos para esta sejam atestados/confirmados por, pelo menos $f + 1$ réplicas, já que a réplica que processou a transação de maneira otimista pode ter produzido resultados espúrios para as operações; o que evidentemente impõe um custo superior aos protocolos tolerantes a apenas faltas por parada.

Embora os resultados obtidos apontem para uma aparente ineficiência do protocolo proposto, em partes isso decorre do fato de ele foi implementado como uma solução baseada em *middleware*. É importante salientar que já é bem estabelecido na literatura que versa sobre replicação de bancos de dados (CECCHET; CANDEA; AILAMAKI, 2008; KEMME; PERIS; PATIÑO-MARTÍNEZ, 2010; KEMME; ALONSO, 2010), que as soluções baseadas em *middleware* – pela sua natureza de implementação –, apresentam um desempenho inferior em relação a soluções implementados em nível de núcleo do SGBD (i.e., *kernel-based*). Tal afirmação decorre do fato de que SGBDs são sistemas bastante complexos, em que a processamento de uma operação envolve alguns estágios de pré-processamento, no intuito e obter o melhor desempenho em sua execução (RAMAKRISHNAN; GEHRKE, 2003; HELLERSTEIN; STONEBRAKER; HAMILTON, 2007). Dentre estes estágios, estão a análise e a otimização, que visam construir um plano de execução adequado para a operação (p. ex.: acesso sequencial, acesso baseado em índice, etc.). E em se tratando de uma solução para replicação de banco de dados baseada em *middleware*, como ela é implementada como uma camada sobre o SGBD, e portanto, fora de sua área interna de abrangência, o protocolo não pode inferir com precisão no processo de otimização para a execução das operações sobre o banco de dados. No sentido de que os recursos disponíveis ao *middleware*, para a realização desta tarefa são bastante limitados. De outro modo, evidentemente que os custos inerentes à replicação poderiam ser minimizados a partir de uma implementação do protocolo proposto como uma solução *kernel-based*. Todavia, uma implementação *kernel-based* afetaria a possibilidade de uso do protocolo em um ambiente com diversidade, o que é algo desejável em termos de tolerância a faltas.

Experimentos com Faltas

Nesta parte da seção de experimentos, é apresentado o comportamento do protocolo proposto sob condições de falta. É importante salientar que, conforme a especificação do protocolo (vide Seção 4.5.3), as f réplicas faltosas se limitam a comprometer a execução daquelas transações, em que estas réplicas são atribuídas como réplicas líderes. Note que as faltas que ocorrem durante a fase de terminação podem até passar despercebidas pelo cliente, já que são necessárias no máximo $f+2$ notificações de finalização da transação, sendo que $n - f \geq f + 2$. Em se tratando dos experimentos com faltas, estes também foram realizados com base na carga padrão do TPC-C, nos mesmos moldes dos experimentos sem faltas. Como a especificação do protocolo (cmfe. Seção 4.2.2) admite que apenas as réplicas estão sujeitas a faltas bizantinas, onde os clientes falham somente por parada, os experimentos consideram apenas faltas nas réplicas. Como a injeção de faltas bizantinas é algo bastante complexo, dada a possibilidade de exibição do comportamento arbitrário por parte dos processos, para estes experimentos foi optado por manipular a execução das transações sobre a réplica faltosa de maneira aleatória ($f = 1$). A opção por afetar apenas e execução da transação e não sua terminação, se deu porque as faltas ocorridas durante a fase de terminação não produzem nenhum efeito sobre o cliente (i.e., elas são mascaradas) – que dá por finalizada a transação, ao receber as primeiras $f + 2$ notificações mutuamente consistentes das réplicas. Deste modo, o impacto mais negativo decorrente de uma falta se dá sobre a execução da transação.

Neste sentido, no intuito de simular a carga de faltas, uma das réplicas foi configurada como faltosa, de modo que o código desta sofreu algumas modificações para ela atuar de maneira faltosa. Estas modificações foram realizadas para simular cinco comportamentos de falta, os quais são:

1. simulação de falta por omissão, pelo não envio do resultado da operação ao cliente;

2. simulação de falta de valor, pelo envio de uma resposta contendo um valor incorreto para a operação do cliente;

3. simulação de falta de valor, pelo envio de um valor nulo para a operação do cliente;

4. simulação de falta de tempo, pela introdução de um atraso arbitrário antes do envio de uma resposta;

5. anulação unilateral da transação.

Embora o cenário construído resulte somente na ocorrência de faltas bizantinas benignas (i.e., não maliciosas), acredita-se que este representa um dos cenários mais recorrentes na prática (GASHI; POPOV; STRIGINI, 2004, 2007). Para gerar a carga de faltas sobre a réplica escolhida, previamente a execução do protocolo é definido um percentual de faltas a ser gerada sobre a amostra das transações executadas por aquela réplica. As transações nas quais serão injetados os comportamentos de falta, são escolhidas de maneira aleatória, e então marcadas como faltosas. Com isso, quando uma operação é recebida para uma transação marcada como faltosa, a réplica líder – que é faltosa – determina a ação de falha a ser realizada por meio do cômputo da equação *random*(*timestamp*-atual) **mod** 5. À vista disso, foi optado por simular uma carga com 10%, 30% e 60% de faltas sobre as transações a serem destinadas a executar de maneira otimista sobre a réplica faltosa, obtidas da amostra de cada experimento. E nos mesmos moldes dos experimentos sem faltas, foram realizadas 14 execuções para cada um dos experimentos, com as mesmas variações para o número de clientes. Os resultados em termos de vazão (tpm) e de latência são reportados nas Figuras 50 e 51, respectivamente.

Para prover uma comparabilidade acerca dos resultados, as Figuras 50 e 51 também demonstram a execução do TPC-C sem faltas, sendo esta apresentada nos gráficos como **PP (regular)**. Ao analisar os resultados obtidos em termos de *throughput*, se pode notar uma redução da capacidade de processamento do protocolo, o que já era esperado. No entanto, dentre os experimentos realizados, o melhor e pior caso em termos de desempenho incorreram na perda de 1,05% e 18,33% para o TPC-C, respectivamente. O melhor caso foi verificado sobre a carga com 10% de faltas, enquanto que o pior foi verificado para a carga com 60% de faltas. O caso médio verificado para cada carga de faltas, em valores discretos, foram de 3%, 7% e 14% para as cargas com 10%, 30% e 60% de faltas, respectivamente. Estes resultados se devem, principalmente, pelo fato do protocolo distribuir as transações para execução otimista, de maneira circular entre as réplicas, similar ao que ocorre no escalonamento *round-robin*. No caso dos experimentos, é esperado que até 25% do total de transações enviadas pelos clientes seja atribuída para execução pela réplica faltosa, já que $f = 1$ e $n = 3f + 1$.

Uma análise sobre os resultados em termos de faltas, em conformidade com a especificação do protocolo mostra que, como a(s) réplica(s) faltosa(s) pode(m) comprometer apenas a execução das transações processadas por ela, a perda de desempenho será sempre pro-

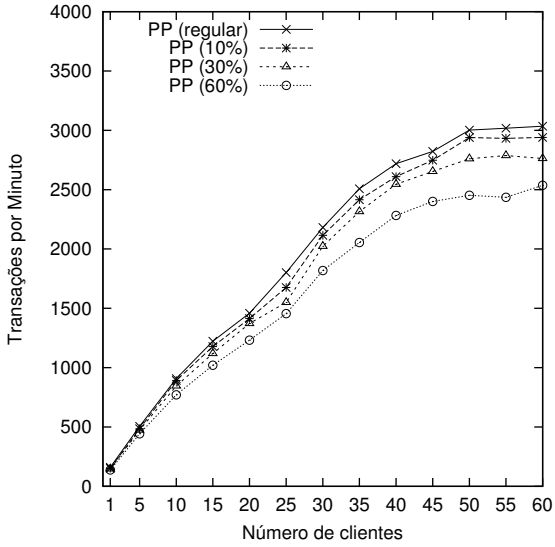


Figura 50 – Desempenho para a carga padrão do TPC-C com faltas.

porcional e limitada ao número de transações atribuídas a elas como líderes. Obviamente que num cenário completamente bizantino, uma réplica faltosa também poderia produzir perturbações sobre o protocolo subjacente de difusão com ordem total, as quais poderiam resultar em atrasos na entrega das mensagens enviada por aquele canal (i.e., onde é executado um consenso bizantino). Todavia, está fora do escopo deste experimento e também desta tese, mensurar o impacto que faltas ocorridas sobre o protocolo subjacente de difusão com ordem total impõem sobre o seu desempenho. A avaliação realizada nesta tese, se dá apenas no intuito de verificar o comportamento dos protocolos propostos em sua área de abrangência, nos termos da execução de transações sobre o ambiente replicado. Uma avaliação bastante ampla a respeito de protocolos de difusão com ordem total tolerantes a faltas bizantinas é relatada no trabalho de Singh *et al.* (SINGH et al., 2008).

Outrossim, tal como ocorre uma perda desempenho quanto ao processamento de transações, também há um incremento para a latência em cenários com faltas; justamente porque a latência é mensurada pelo inverso do *throughput* obtido para a execução do TPC-C. No caso, a latência é penalizada principalmente pelos atrasos introduzidos pela

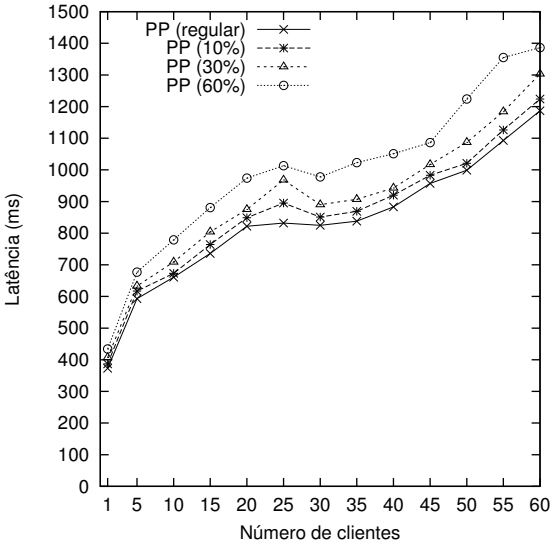


Figura 51 – Latência para a carga padrão do TPC-C com faltas.

réplica faltosa durante o processamento da transação, e também pelas anulações unilaterais, as quais impedem que as transações sejam concluídas. Esta última condição não é computada para efeito de desempenho no TPC-C, já que a transação não completa seu ciclo, e portanto, não pode ser considerada como processada pelo protocolo de replicação. De outro modo, um aspecto não demonstrado graficamente, mas que é digno de menção, é sobre o percentual de anulações verificados nos experimentos. Os experimentos demonstraram um crescimento entre 4% e 9% para os cenários com cargas de faltas. No entanto, também já era esperando um crescimento neste sentido, justamente porque as transações especificadas pelo *benchmark* são preparadas para anular transações, cujos resultados recebidos pelas operações não estão de acordo com o esperado. Na maioria dos casos, as transações utilizam os resultados de operações anteriores como argumentos para as operações posteriores. E no caso de um resultado em desacordo com o que é esperado, a operação imediatamente subsequente daquela transação não é executada, sendo a transação conduzida para anulação. É importante notar que este aspecto não pode ser encarado como generalizado, mas apenas inerente à especificação do *benchmark* TPC-C.

6.2 AVALIAÇÃO DO PROTOCOLO DE VALIDAÇÃO ATÔMICA FRACA NÃO-BLOQUEANTE

Nesta seção descreve-se alguns resultados obtidos a partir de avaliações realizadas sobre as contribuições apresentadas no Capítulo 5 desta tese. Para tanto, a Seção 6.2.1 descreve a primeira parte da avaliação, a qual é efetuada a partir de uma comparação analítica dos custos/eficiência teórica do protocolo HB-NBWAC em relação a alguns protocolos encontrados na literatura, que resolvem o mesmo problema com faltas por parada. A segunda parte da avaliação é reportada na Seção 6.2.2, a qual versa sobre os detalhes de implementação de um protótipo da arquitetura proposta, bem como do protocolo HB-NBWAC. No mesmo ensejo, a seção apresenta alguns resultados obtidos por meio de experimentos conduzidos sobre a implementação da arquitetura proposta, e da execução do protocolo HB-NBWAC sobre tal arquitetura.

6.2.1 Avaliação Analítica

Nesta seção é examinado o custo associado à terminação da transação, através de alguns protocolos de validação atômica encontrados na literatura. Neste sentido, a fim de verificar também a eficiência teórica acerca da solução proposta, esta avaliação é realizada pela comparação analítica dos resultados obtidos para o HB-NBWAC, em relação a outros protocolos de validação atômica propostos na literatura. É importante salientar que esta avaliação é realizada nos mesmos moldes dos trabalhos adotados como parte desta avaliação (GUERRAOUI; SCHIPER, 1995a; GUERRAOUI; LARREA; SCHIPER, 1996; ABDALLAH; PUCHERAL, 1999; GREVE; NARZUL, 2002). Para tanto, a Tabela 8 enumera alguns valores verificados através das medidas de desempenho mais recorrentes na literatura, nomeadamente a latência, a complexidade em termos de mensagens e a resiliência. Não obstante, algumas medidas adicionais e não usuais também são reportadas.

Para esta análise, da mesma maneira como é realizado pelos trabalhos comparados, os valores refletem a execução dos protocolos pelo cenário mais favorável, isto é, aquele no qual todos os participantes/processos votam `sim` e não há suspeitas de faltas durante a execução dos protocolos. No caso, a Tabela 8 descreve os resultados verificados para o HB-NBWAC, para os clássicos 2PC (*Two Phase Commit*) e 3PC (*Three Phase Commit*), e para os protocolos denominados por DNB-AC (*Decentralized Non-Blocking Atomic Commitment*) (GUERRAOUI; SCHI-

PER, 1995a), MD3PC (*Modular Decentralized 3PC*) (GUERRAOUI; LARREA; SCHIPER, 1996), NB-2PC (*Non-Blocking 2PC*) (GREVE; NARZUL, 2002) e ANB-CLL (*Assynchronous Non-Blocking Coordinator Logical Log*) (ABDALLAH; PUCHERAL, 1999). O termos n e f das expressões presentes na Tabela 8 denotam, respectivamente, o número de participantes e o limite de faltas tolerado. No caso específico do protocolo HB-NBWAC, as expressões apresentadas pela tabela são simétricas, isto é, considera-se um mesmo número de máquinas virtuais por sítio – note que a especificação do protocolo permite diferentes valores para f_v, no âmbito de cada sítio (vide Seção 5.6.1).

Tabela 8 – Custo verificado em protocolos de validação atômica.

Protocolo	Passos	Mensagens	Resiliência	Participantes	Processos	Faltas
2PC	3	$3n$	0	–	–	-
3PC	5	$5n$	–	–	–	parada
DNB-AC	3	$2n^2+n$	$f<n/2$	$2f+1$	$2f+1$	parada
MD3PC	3	$3nf+3n$	$f<n/2$	$2f+1$	$2f+1$	parada
NB-2PC	3	$2nf+3n$	$f<n$	$2f+1^*$	$2f+1$	parada
ANB-CLL	2	n^2+n	$f<n/2$	$2f+1$	$2f+1$	parada
HB-NBWAC	4	n^2+2n-3^*	$f<n/2$	$2f_s+1$	$2f_s2f_v+2f_s+2f_v+1+n$	híbrida/byz

Embora a literatura discorra que o 2PC (GRAY, 1978) é um dos protocolos mais utilizados na prática (LIU; AGRAWAL; ABBADI, 1998; SHAHZA et al., 2008) – o que se deve, principalmente, ao fato dele ser o mais eficiente dentre os comparados – ele não é resiliente. É importante verificar, que sua eficiência está pautada no requerimento de apenas três passos de comunicação e de $3n$ mensagens, para que uma decisão seja obtida por cada participante correto. A dinâmica do 2PC é a seguinte, o TM líder solicita os votos aos TMs de cada sítio participante, coleta seus votos, decide pelo resultado da transação e informa tal decisão aos demais TMs. Porém, se o TM líder falha durante o segundo passo, o protocolo é induzido a uma situação de blocagem até que ele seja recuperado. Isto posto, pode-se dizer que o 2PC não resolve o NBAC, nem o NBWAC. Para resolver tanto o NBAC com o NBWAC, os protocolos o fazem por meio de alguns passos de comunicação adicionais, o que naturalmente, requer a difusão de um número superior de mensagens.

Isto é o que ocorre com o protocolo 3PC (SKEEN, 1981), que consegue resolver o NBAC a um custo de cinco passos de comunicação e da difusão de $5n$ mensagens. O 3PC garante a ausência de blocagem (i.e., *liveness*) por meio da substituição do TM líder, se este vier a falhar. Todavia, esta substituição do TM líder pode induzir o protocolo a uma terminação não consistente nos participantes, se o TM líder

substituído não tiver, de fato, falhado – situação na qual eles podem decidir de maneiras diferentes (BERNSTEIN; HADZILACOS; GOODMAN, 1987). Da mesma maneira, a elegibilidade em torno da substituição do TM líder, requer que todos os TMs, independente de seu papel no protocolo, mantenham as informações necessárias para a terminação da(s) transação(ões), o que necessariamente incorre em um custo adicional de comunicação.

Em relação aos resultados verificados para os protocolos DNB-AC, MD3PC, NB-2PC e ANB-CLL, um aspecto que merece destaque é que todos eles são especificados a partir de um serviço de consenso subjacente, que é usado para a terminação em casos onde estes protocolos suspeitam da ocorrência de faltas. Os protocolos DNB-AC, MD3PC e NB-2PC têm a mesma estrutura básica, em que a terminação é realizada por meio de três passos de comunicação, sendo que a principal diferença entre eles está no número de mensagens requeridas para a realização do acordo por cada um deles. Todavia, uma decisão `commit` pelo DNB-AC requer uma estimativa de terminação favorável de todos os participantes da transação, enquanto que no MD3PC e no NB-2PC os participantes podem decidir ao receber os votos/estimativas de apenas uma maioria.

E este é caso em que o MD3PC e o NB-2PC se destacam de outros, a despeito da igualdade de resultados em termos de latência (e igualmente ao 2PC), pois ambos possibilitam melhoras sobre a resiliência do protocolo. Isto ocorre porque eles são especificados a partir de um princípio em que a terminação é delegada apenas a um subconjunto de processos dentre os participantes, este subconjunto tem cardinalidade inferior ou igual a n e é denominado por Set_{NB} no MD3PC, e S no NB-2PC. Note que, embora eles tenham denominações diferentes, seus papéis são idênticos, de modo que os protocolos se distinguem pela maneira na qual estes conjuntos são envolvidos na terminação. Em ambos os protocolos, o pedido dos votos é enviado apenas aos participantes pertencentes a estes conjuntos, de modo que apenas os votos destes participantes são computados para fins de decisão. De outro modo, se a decisão não puder ocorrer devido a alguma suspeita de falha, ambos iniciam um protocolo de consenso subjacente, onde o NB-2PC decide pelo consenso entre todos os participantes, enquanto que o MD3PC o faz apenas com os participantes do conjunto Set_{NB}. A despeito destas similaridades, há também uma distinção em relação às cardinalidades destes conjuntos, as quais são $|Set_{NB}| \geq 2f + 1$ e $|S| \geq f + 1$.

Dentre os protocolos comparados na Tabela 8, o ANB-CLL é o que apresenta a menor latência de comunicação, já que ele é baseado

no princípio do 1PC (*One Phase Commit*) (STONEBRAKER, 1979) e, portanto, elimina a fase de votação e requer apenas dois passos de comunicação para a terminação. Este feito decorre do fato de que o protocolo retira o direito de veto dos participantes quanto ao resultado da transação, o que quer dizer que todos os participantes decidem por imposição do líder. No entanto, tal abordagem requer alguns cuidados e incide em custos adicionais para o protocolo. Por exemplo, se um participante não puder de qualquer maneira validar sua parte na transação, ele decidirá pelo que foi imposto pelo líder, mas necessitará ser reparado em algum momento após a decisão. Para isso, o protocolo mantém no coordenador uma lista das operações que foram submetidas para processamento por cada participante, de modo a possibilitar o reenvio delas aos respectivos participantes, no caso de uma necessidade de reparação.

Notadamente, todos os protocolos avaliados apresentam maior eficiência em relação ao HB-NBWAC, justamente porque eles toleram apenas faltas por parada, enquanto que o HB-NBWAC se baseia em um modelo híbrido de falhas em que os participantes da terminação podem manifestar comportamentos arbitrários/bizantinos. Isso se deve, principalmente, pela abordagem adotada através da separação dos participantes do modelo original do NBAC em dois agentes distintos. Esta também é a razão pela qual o HB-NBWAC requer um número maior de processos do que os demais protocolos avaliados, já que a atuação do CTM (*Collaborative Transaction Manager*) requer, pelo menos, $2f_v + 1$ réplicas do agente TM por sítio – um requisito para tolerar/mascarar/confinar as faltas bizantinas na terminação da transação –, mais o processo que implementa o agente DM daquele sítio.

De outro modo, embora seja requerido $2f_v+1$ réplicas de TM por sítio, estas atuam em conjunto, de tal maneira que apenas uma única mensagem assinada pelo CTM (i.e., por $f+1$ réplicas de TM) é suficiente para atestar a validade desta perante os demais sítios. Neste caso, uma nota a respeito da expressão referente as mensagens, é que esta denota o número requerido para o pleno funcionamento do protocolo. Todavia, devido à natureza replicada do CTM, em que todos os TMs corretos de um sítio enviam para todos os TMs dos demais sítios, o número total de mensagens trocadas pode chegar a $(n^2+2n-3)(2f+1)^2$. Em face aos aspectos avaliados e resultados apresentados, é importante salientar que o propósito do HB-NBWAC não é ser o protocolo mais eficiente, mas sim uma alternativa factível, que possibilite a terminação de transações distribuídas em ambientes passíveis de faltas bizantinas, através da hibridização do modelo de sistema.

6.2.2 Avaliação Experimental

E por fim, nesta seção são apresentados os detalhes acerca da implementação de um protótipo da arquitetura e protocolo propostos, bem como alguns resultados obtidos.

Aspectos de Implementação

Com o propósito de desenvolver uma prova de conceito acerca do funcionamento da arquitetura ilustrada pela Figura 43 e do protocolo propostos, foi realizada a implementação de um protótipo simplificado da arquitetura e do protocolo HB-NBWAC. Para tanto, o modelo adotado para a implementação da arquitetura com os componentes requeridos para seu funcionamento, se deu nos termos na Figura 52. O desenvolvimento do protótipo foi realizado inteiramente na plataforma Java, o que foi motivado pela profusão desta no âmbito de sistemas distribuídos, mas principalmente pelas vantagens e facilidades que ela oferece em relação às outras tecnologias/plataformas/ambientes de programação (p. ex.: portabilidade, segurança, etc.).

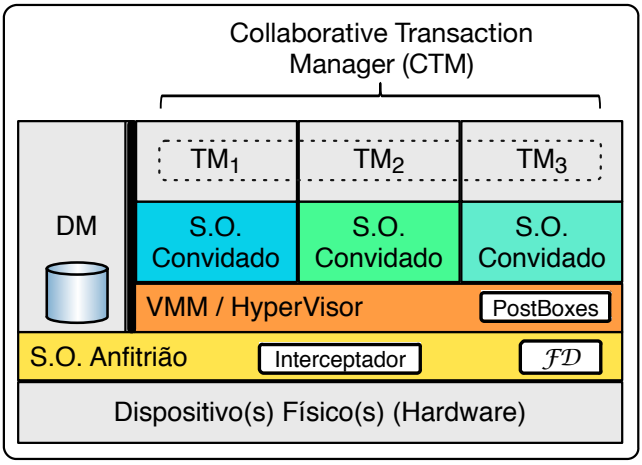


Figura 52 – Modelo adotado para a implementação da arquitetura.

Note que os módulos de interceptação e detecção de falhas foram implementados em nível de sítio. No caso das *PostBoxes*, elas também foram implementadas sobre o ambiente anfitrião, porém, o acesso a elas por partes das máquinas virtuais se dá através do VMM.

Algumas APIs e facilidades foram empregadas na construção do protótipo, dentre as quais se pode destacar: (*i*) a biblioteca *ThreshSig* (*http://threshsig.sourceforge.net*) como base para a implementação do esquema de assinaturas de limiar; (*ii*) a biblioteca *Chronicle* (*https://github.com/peter-lawrey/Java-Chronicle*) para implementar o ambiente de memória compartilhada para as *PostBoxes*. O módulo de interceptação de mensagens em nível de anfitrião foi implementado com o auxílio da biblioteca *Jpcap* (*https://github.com/jpcap*), como suporte para obter as mensagens que chegam através das interfaces virtuais de rede.

Porém, para prover a recepção confiável das mensagens interceptadas em nível de anfitrião, foi necessário o desenvolvimento de uma biblioteca de comunicação própria (i.e., primitivas *send* e *receive*), em que foram utilizados os *SocketChannels* disponíveis na API Java NIO, como abstração de comunicação, e as estruturas de dados do pacote java.util.concurrent (p. ex.: *Linked Blocking Queue*) para implementação dos *buffers* de recepção de mensagens. Para a concretização da comunicação por meio do módulo implementado, foi necessária a realização de alguns ajustes no ambiente. Por exemplo, em todos os ambientes anfitriões, cada máquina virtual foi equipada com duas interfaces de comunicação, uma em modo *bridge* para permitir a comunicação das máquinas virtuais com a rede de comunicação, e outra em modo *host-only* para o mecanismo de recepção confiável – esta última consiste em um canal de comunicação privado e síncrono entre o VMM e cada máquina virtual. Neste caso, quando a primitiva *send* é invocada pela máquina virtual, ela envia a mensagem pela interface de rede que opera em modo *bridge*, enquanto a recepção, que ocorre pela primitiva *receive*, é realizada pelo canal privado fornecido pela interface em modo *host-only*.

No que concerne à memória compartilhada provida pela *PostBox*, esta pode ser encarada como um dos principais componentes da arquitetura, cujo funcionamento é imperioso para corretude dos algoritmos especificados através dela, razão pela qual ela deve ser cuidadosamente implementada. A alternativa mais viável encontrada para sua implementação foi através de um recurso existente em VMMs do Tipo II (vide Seção 2.3) denominado por *Shared Folders*, o qual provê uma espécie de sistema de arquivos distribuído similar à um NFS (SUN, 1989), que pode ser utilizado apenas entre as máquinas virtuais e o ambiente anfitrião. Um aspecto interessante deste recurso, é que ele permite o compartilhamento de arquivos e diretórios entre as máquinas virtuais e sistema anfitrião, de maneira independente do(s) serviço(s) de rede, o que requer apenas a habilitação de um serviço específico no VMM. E

por assim dizer, este componente foi implementado por meio da biblioteca *Chronicle* com o uso de arquivos mapeados em memória (*memory mapped files*), a fim de prover um melhor desempenho sobre a execução das operações de E/S básicas nos arquivos.

No entanto, devido à restrições tecnológicas, o único VMM do Tipo II que provê o suporte para o uso de arquivos mapeados em memória sobre o sistema de arquivos distribuído é o *VMWare*, razão pela qual ele foi adotado para a implementação, e por conseguinte, para os experimentos. Todavia, o uso deste recurso também requer alguns cuidados no que diz respeito ao controle de acesso aos recursos compartilhados, principalmente no âmbito do funcionamento das *PostBoxes*, a fim de evitar acesso indevidos por parte de máquinas virtuais comprometidas. Este controle foi realizado da seguinte maneira, cada máquina virtual tem acesso de escrita no diretório que abriga os arquivos que são usados pelo *Chronicle* para implementar sua *PostBox*, enquanto que nos diretórios que abrigam os arquivos usados pelas *PostBoxes* das demais máquinas virtuais o acesso é de somente-leitura. Esse controle de acesso é gerenciado pelo próprio VMM. O acesso aos diretórios pelas máquinas virtuais é algo bastante simples, de modo que para sistemas operacionais convidados baseados em Unix este é feito pelo comando `mount -t vmhgfs .host:/diretório_compartilhado /mnt/ponto_de_montagem/`, e em sistemas operacionais baseado em Windows o respectivo comando é `net use <letra_unidade_montagem>: \\vmware-host\diretório-compartilhado`.

Resultados Obtidos

É importante salientar que o interesse por trás desta avaliação experimental é demonstrar apenas os custos associados à terminação da transação por meio do HB-NBWAC, de modo que a intenção não é avaliar o desempenho acerca do processamento de transações, mas tão somente o custo incidente da(s) rodada(s) (i.e., *rounds*) requeridas para obter uma decisão unânime dos participantes. Note que uma avaliação acerca do desempenho em termos do processamento de transações envolve diversos fatores e depende de uma série de características específicas das transações, tais como, tipos de operações realizadas, a quantidade de operações, a complexidade destas, o tempo de execução, etc. Por esta razão, a avaliação foi realizada somente por meio de alguns *microbenchmarks*, a fim de avaliar as partes consideradas como mais críticas da arquitetura, e que por conseguinte, tendem a causar o maior impacto sobre o desempenho dos algoritmos.

Os experimentos foram realizados com vista para simular um am-

biente de computação distribuída onde $n = 2f + 1$ e $f = 1$. Para tanto, estes ocorreram em um ambiente de rede local, a partir de três máquinas com a mesma composição de *hardware*, em que cada uma apresentava as seguintes configurações: 8GB de RAM; 1 (uma) Interface Ethernet Gigabit, 1 (um) microprocessador Intel® Core™i7 2,3GHz com 6MB cache e 4 núcleos. Estas máquinas foram interligadas através de suas interfaces Ethernet por um comutador (*switch*) D-Link DGS-3100-24P Ethernet 10/100/1000. Todas as máquinas foram instaladas com o mesmo sistema operacional para operar no nível de anfitrião, sendo este o Debian Linux 7.0 ("wheezy"), e sobre o qual foram instalados o *VMware Workstation* 10 como VMM e o MySQL 5.5.33 para prover o suporte para os bancos de dados locais dos sítios – embora não tenha sido usado nos experimentos. Em cada VMM foram criadas 3 instâncias de máquinas virtuais, para atender as especificações requeridas para o HD-NBWAC (i.e., $2f + 1$, onde $f = 1$). As instâncias de máquinas virtuais foram configuradas pela instalação do mesmo sistema operacional (Debian 7.0), sendo que para cada uma delas foi disponibilizado 1GB de memória RAM, 1 (um) núcleo do microprocessador e 10GB de armazenamento em disco.

Neste sentido, um primeiro *microbenchmark* consistiu na realização de uma série de operações de escrita e de leitura nas *PostBoxes*, no intuito de verificar, e sobretudo, de demonstrar o impacto que o mecanismo de comunicação *intra-sítio* incide sobre a execução e desempenho do protocolo. Este experimento foi realizado em uma única máquina física e três máquinas virtuais executando sobre esta, o que ocorreu da seguinte maneira. Uma única máquina virtual vm_1 realizou as operações de escrita sobre sua *PostBox*, enquanto que as demais máquinas virtuais (i.e., vm_2 e vm_3) efetuaram operações de leitura na mesma *PostBox* que a vm_1 efetuou as escritas. A opção por tal cenário de experimento se deu em razão de que o interesse era introduzir concorrência no acesso à memória compartilhada, justamente para analisar um cenário semelhante àquele que ocorre através da execução do protocolo sobre a arquitetura. Para tanto, os resultados verificados para a latência compreendem à execução de operações de escrita pela vm_1, com variação no tamanho das mensagens entre 1024 *bytes* e 65536 *bytes* (i.e., 1 Kb a 64 Kb), com alternância dos tamanhos por uma escala exponencial.

Os resultados para os experimentos deste primeiro *microbenchmark* são reportados na Figura 53. Os valores observados para a latência são reportados na ordem de *microssegundos* (i.e., μs), e compreendem aos tempos médios observados pela execução de 100.000 operações

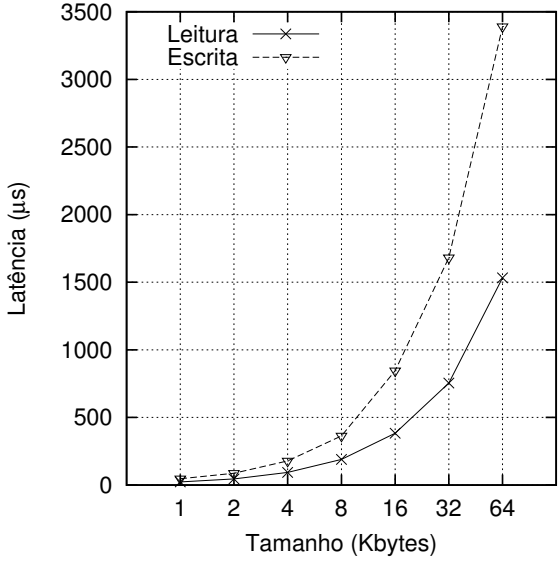


Figura 53 – Latência observada para operações na *PostBox*.

de leitura e de escrita, para cada tamanho distinto de mensagem. Note que os resultados são bastante satisfatórios, sendo que os extremos observados foram $47\mu s$ para escritas de 1024 *bytes* e $24\mu s$ para leituras de 1024 *bytes*; e de $3390\mu s$ para escritas de 65536 *bytes* e $1532\mu s$ para leituras de 65536 *bytes*. Estes valores corroboram para que o mecanismo de comunicação *intra-sítio* proposto seja eficiente e sua incidência seja ínfima sobre o desempenho do protocolo. Em partes, o custo de comunicação via *PostBox* incide em pouco sobre o desempenho do protocolo por duas razões, primeiro pelo fato da mesma ter sido implementado a partir de *memory mapped files*, e segundo, porque as mensagens utilizadas no protocolo contém poucas dezenas de *bytes*, e portanto, a latência é inferior à da menor latência observada pela execução deste *microbenchmark*.

Um aspecto recorrente na literatura discorre sobre o gargalo tipicamente observados em protocolos tolerantes a faltas bizantinas, em decorrência do custo pela adoção de mecanismos criptográficos (CASTRO; LISKOV, 1999, 2001; CACHIN; KURSAWE; SHOUP, 2005; CORREIA; NEVES; VERÍSSIMO, 2006). Embora haja uma constância em termos do crescimento do poder de processamento dos computadores modernos, o

tempo consumido para a realização de operações criptográficas em um protocolo é algo que não pode ser negligenciado. Neste sentido, também foi realizado um *microbenchmark* a fim de verificar o custo em termos práticos, decorrente das operações criptográficas utilizadas para prover as garantias do protocolo. Para isso, foram realizados alguns experimentos por meio da geração de assinaturas digitais em três cenários, em que as assinaturas foram produzidas através de um algoritmo criptográfico para gerar assinaturas de limiar compatíveis com o padrão RSA (SHOUP, 2000); e também através de um algoritmo criptográfico para gerar apenas assinaturas RSA (RIVEST; SHAMIR; ADLEMAN, 1978). Em cada cenário de experimento, variou-se o mecanismo criptográfico e o tamanho da chave usada naquele mecanismo. Os tempos foram observados através da geração de 100.000 assinaturas para uma mensagem de 256 *bytes* (p. ex.: um tamanho próximo ao das mensagens usadas pelo protocolo), sobre cada mecanismo e tamanho de chave. Os resultados são apresentados na Figura 54. Note que, para a Figura 54 os valores são observados na ordem de *milissegundos*, o que decorreu do fato de que os tempos observados para as assinaturas de limiar foram nesta escala.

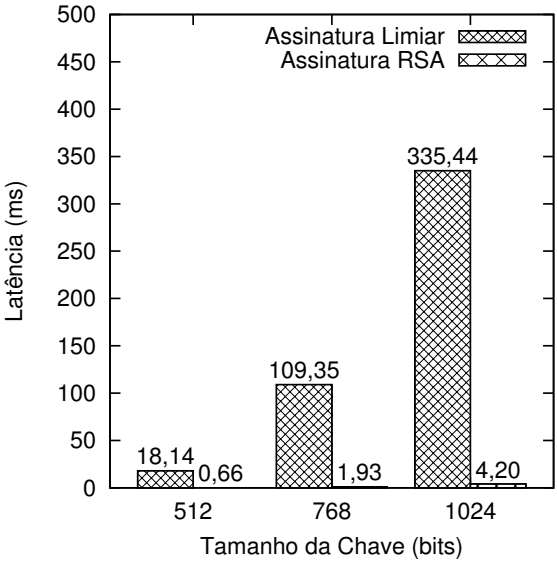


Figura 54 – Latência observada para operações de assinatura.

As assinaturas foram geradas pela atuação conjunta das máquinas virtuais, onde cada uma assina a mensagem com sua chave parcial, coloca na respectiva *PostBox* para uso pelas demais, busca nas *PostBoxes* das outras mais uma assinatura parcial para gerar a assinatura de limiar. É importante notar que os experimentos envolvem sempre três máquinas virtuais, e neste caso, duas assinaturas parciais são suficientes para produzir uma assinatura de limiar. A cada mensagem assinada por este esquema, as máquinas virtuais alternam os pares para a geração da assinatura (p. ex.: vm_1 e vm_2, vm_1 e vm_3, etc.). Para os experimentos a partir das assinaturas RSA, foi considerado o tempo médio consumido para concatenar duas assinaturas em uma mesma mensagem. Isto é, uma vm_i assina a mensagem e coloca-a em sua *PostBox*, em seguida, aguarda pela mesma mensagem com a assinatura da vm_j – o que indica que a maioria das máquinas virtuais daquele sítio está de acordo com o conteúdo da mensagem. Pelos resultados dos experimentos, se verifica que o tempo médio consumido para a geração de uma assinatura de limiar é muito superior ao tempo requerido para a geração de assinaturas RSA. Deste modo, note que o uso de $f + 1$ assinaturas RSA é uma alternativa que pode ser adotada em substituição ao esquema de assinaturas de limiar, quando apropriado.

E finalmente, no intuito de avaliar a capacidade do protocolo em termos de vazão (i.e., *throughput*), também foi realizado um *microbenchmark* para validar este comportamento para o protocolo. Neste sentido, cada sítio foi carregado com um mesmo arquivo contendo 10.000 entradas identificadas, que representaram transações não concorrentes e sem nenhuma operação. Outrossim, conforme a justificativa mencionada ao início desta seção, o propósito deste *microbenchmark* é tão somente verificar a capacidade de processamento do protocolo de terminação baseado no HB-NBWAC, em termos de rodadas por segundo. A partir deste arquivo, as instâncias do protocolo de terminação são iniciadas nos sítios, pelos respectivos TMs, com um intervalo de 5ms entre a submissão de cada transação para terminação pelo protocolo. Os experimentos realizados neste *microbenchmark* ocorreram através da implementação original do HB-NBWAC (p. ex.: com assinaturas de limiar) e de outra implementação do protocolo onde o esquema de assinaturas de limiar foi substituído por assinaturas RSA, embora com os mesmos tamanhos de chaves. Os resultados para este experimento são apresentados na Figura 55.

O intuito da avaliação através deste *microbenchmark* foi justamente o de analisar o custo incidente da rodada (*round*) do acordo requerido para a terminação da transação. Pelos experimentos, foi

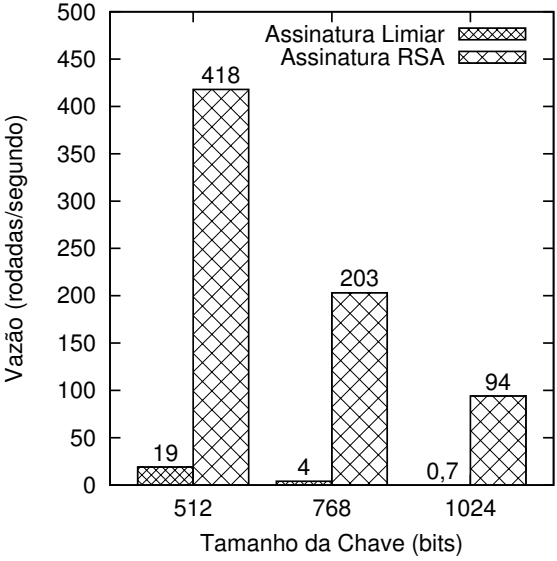


Figura 55 – Capacidade do protocolo em termos de vazão.

observado que o maior impacto no desempenho do HB-NBWAC foi causado pelo mecanismo de assinaturas utilizado, de modo que o uso de assinaturas de limiar incidiu em uma degradação demasiada de desempenho. Note que o tempo médio consumido por uma assinatura de limiar é drasticamente maior em ordens de magnitude do que o consumido pelas assinaturas RSA. Além disso, a estabilização em termos dos resultados também pode ser explicada pela escolha de não executar diversas instâncias do protocolo de acordo de maneira concorrente nos experimento, mas uma após a outra. É evidente que a execução de múltiplas instância do protocolo de terminação poderiam trazer ganhos de desempenho em termos de vazão. Por outro lado, isso não necessariamente não traria ganhos reais e, numa direção contrária poderia até mesmo introduzir problemas em um ambiente real de processamento de transações. O que decorre do fato de que as transações só podem ser validadas se elas não violam algum critério de consistência (vide Seção 2.1.2), usualmente aquele baseado em serialização.

À vista disso, um dos aspectos tomados como subsídio por um sítio para a sua declaração de voto, é a verificação de que a parte executada localmente por ele cumpre os critérios de consistência assumidos

para o sistema. Supondo que seja permitida a execução concorrente de instâncias do protocolo de terminação. Em um cenário onde há duas instâncias em execução i e $i+1$, um sítio s_1 pode ter votado `sim` para uma transação T_j na instância i porque ele verificou que a validação é possível no momento daquela execução, e também votou `sim` para uma transação T_k na instância $i+1$ pela mesma razão. Todavia, se houvesse uma relação de dependência entre as transações T_j e T_k, na qual a ordem de validação interferisse na consistência do banco de dados, a execução concorrente do protocolo de terminação permitiria que a instância $i+1$ fosse concluída antes da instância i. Neste caso, todos os sítios i[1] ficariam impedidos de validar a transação T_k até que a transação T_j fosse validada. Ou de outro modo, se neste cenário as transações fossem validadas logo após conclusão do protocolo de terminação, elas conduziriam o banco de dados a um estado inconsistente, e portanto, violariam as propriedades básicas ACID. O problema da execução concorrente de instância de protocolos de acordo já é conhecido na literatura, de modo que mais detalhes podem ser encontrados em (LAMPORT; MALKHI; ZHOU, 2010).

6.3 CONSIDERAÇÕES DO CAPÍTULO

Este capítulo apresentou as avaliações realizadas sobre os protocolos introduzidos no âmbito desta tese, para fins de validação das propostas desenvolvidas. Neste sentido, a validação realizada buscou demonstrar dois importantes aspectos em se tratando de algorítmica distribuída: a eficiência verificada para ambas as propostas – o que se deu através das avaliações analíticas realizadas; e a factibilidade em termos de implementação das soluções propostas.

Em relação aos resultados obtidos acerca da implementação do *middleware*, cabem algumas observações. Embora sua implementação/codificação tenha sido realizada com extrema cautela e cuidado, esta foi desenvolvida apenas com o propósito de servir como uma prova de conceito para o protocolo proposto. Uma razão pela qual, no estágio atual de desenvolvimento nenhuma tentativa foi realizada no intuito de torná-la eficiente a ponto de ser usada em ambientes reais. Não obstante, a despeito do desempenho verificado e aparente ineficiência decorrente dos resultados obtidos, acredita-se que os resultados mostram que o protocolo tem um desempenho competitivo quando comparado a um banco de dados não-replicado, ao passo de que ele oferece robustez

[1] Se a decisão foi pela validação é porque todos votaram `sim`

e garantias mais fortes para as aplicações. Do mesmo modo, a metodologia empregada para a avaliação de desempenho do protocolo e os resultados também mostram que existem vários *tradeoffs* que conduzem a melhores ou piores resultados em termos de algumas das métricas consideradas, por isso, é importante observar que cabe ao projetista do sistema a decisão sobre aquilo que é mais adequado para uma aplicação específica.

No que concerne aos resultados obtidos para a arquitetura de sistema distribuído, que visa prover o suporte à execução do protocolo HD-NBWAC; os *microbenchmarks* realizados sobre o protótipo demonstraram, de um modo geral, que os resultados foram aceitáveis/competitivos; uma vez que a arquitetura proposta provê condições para resolver um problema que não admite solução. O que, portanto, pode ser considerado como um *tradeoff* justificado. Evidentemente que os resultados também demonstram que as implementações carecem de otimizações, principalmente no que tange à vazão (i.e., *throughput*) do protocolo HD-NBWAC. No entanto, um dos aspectos que corrobora para aqueles resultados é o fato da implementação não ter considerado a execução de instâncias paralelas/concorrentes do acordo – o que fora devidamente justificado na conclusão do Capítulo 5. Todavia, este é um ponto cujos esforços para verificação acerca de sua viabilidade deverá ser avaliada em implementações e trabalhos futuros.

7 CONCLUSÃO

Oriunda da área de bancos de dados (LAMPSON, 1981), a noção de transação pode ser vista como um bloco básico de construção (i.e., *building block*), pelo qual, sistemas de computação podem ser especificados e implementados com garantias de confiabilidade, no que concerne à manipulação de dados. A despeito de efeitos oriundos de concorrência e de falhas no decurso da execução de um conjunto de operações de manipulação de dados, as transações provêem suas garantias através do satisfazimento às propriedades básicas ACID. Todavia, a literatura nos mostra que se considerada a ocorrência de falhas mais severas como as bizantinas (LAMPORT; SHOSTAK; PEASE, 1982) tais garantias não podem ser plenamente atendidas tampouco no processamento (GARCIA-MOLINA; PITTELLI; DAVIDSON, 1984, 1986; GASHI; POPOV; STRIGINI, 2004, 2007), como na terminação de transações (CHARRON-BOST; SCHIPER, 2004).

Em face destes fatos, esta tese considerou estas questões através da discussão dos detalhes envolvidos no projeto de protocolos tolerantes a faltas bizantinas para o processamento de transações em bancos de dados relacionais, e também no projeto de protocolos de terminação de transações em um sistema de computação distribuída através da validação atômica não-bloqueante. Com vista para a importância das transações no contexto de sistemas computacionais, foram propostas algumas soluções para lidar com a sujeição a faltas bizantinas no processamento de transações em bancos de dados relacionais, e também na terminação de transações através de protocolos que satisfazem as propriedades requeridas para a ausência de blocagem quanto à validação atômica destas.

As proposições realizadas no âmbito desta tese visaram, sobretudo, apresentar soluções capazes de lidar com problemas encontrados na especificação de protocolos tolerantes a faltas bizantinas envolvendo transações, já que resultados demonstrados por trabalhos predecessores aos desenvolvidos nesta tese, apontaram para alguns *trade-offs* relacionados ao desempenho, a consistência e a eficácia em tolerar faltas bizantinas em transações. Em partes, isto coloca em discussão quanto à viabilidade de adoção destas soluções para determinados ambientes de computação, já que em alguns casos não é aceitável sacrificar algum requisito em detrimento de outro(s).

Não obstante, as duas grandes contribuições desta tese foram demonstrar que tolerar faltas bizantinas em bancos de dados relacionais é

algo factível, ao passo que é possível assegurar a execução concorrente de transações e proporcionar uma consistência forte; e demonstrar que a partir de especificação de um novo paradigma baseado numa abordagem realista, é possível tolerar o comportamento bizantino no âmbito de um protocolo de validação atômica não-bloqueante.

No que segue, será feito um retrospecto geral dos trabalhos desenvolvidos no âmbito desta tese. Neste sentido serão revisados os objetivos derivados da motivação para a realização deste trabalho, e na sequência, serão elencados os trabalhos realizados no período e sugestões de trabalhos futuros.

7.1 REVISÃO DOS OBJETIVOS E REALIZAÇÕES

Um primeiro principal objetivo deste trabalho consistiu em prover uma solução alternativa às soluções até então existentes no âmbito de bancos de dados relacionais tolerantes a faltas bizantinas, no sentido do proporcionar maior flexibilidade pela admissão de hipóteses mais realistas para o ambiente do sistema computacional. O primeiro aspecto estava relacionado a superar algumas limitações encontradas nos trabalhos correlacionados, a fim de construir uma solução mais adequada para cenários reais. No caso, as abordagens empregadas nos trabalhos concernentes à replicação de bancos de dados: (*i*) não permitiam a execução concorrente de transações (GASHI; POPOV; STRIGINI, 2007); (*ii*) dependiam de uma entidade centralizada confiável em meio à um ambiente suscetível à faltas bizantinas; (*iii*) visavam critérios de consistência mais fracos que a serialização.

Neste sentido, para resolver este problema foi desenvolvido um protocolo para replicação tolerante a faltas bizantinas de bancos de dados relacionais baseado numa abordagem não-intrusiva, isto é, que é realizado em nível de *middleware* e independente do SGBD – razão pela qual ele não impõe modificações sobre a implementação do SGBD e tampouco das estruturas de dados que compõem o(s) banco(s) de dados. A especificação desta solução levou em conta a capacidade de prover um nível aceitável de concorrência, a despeito do critério de consistência visado que consiste naquele mais forte baseado em serialização. Portanto, os objetivos específicos apresentados na Seção 1.2 e enumerados por 1 e 2, foram alcançados. Não obstante, a implementação do protocolo de replicação culminou no desenvolvimento de um *middleware*, cuja arquitetura e visão geral de funcionamento foram discutidas. Tal arquitetura de *middleware* resultou da integração de uma

séria de componentes que tiveram de ser especificados para atender aos requisitos estabelecidos para o protocolo de replicação de bancos de dados. Os componentes especificados para o *middleware* foram detalhados, com ênfase para o papel desempenhado por cada um deles no processamento da transação.

Acredita-se que a arquitetura de *middleware* desenvolvida a partir do protocolo de replicação proposto pode ser útil para mascarar falhas em sistemas de bancos de dados reais, já que ela visa dar suporte à heterogeneidade na composição do ambiente replicado de banco de dados, um requisito fundamental para assegurar a independência de falhas. Em se tratando dos resultados experimentais obtidos pela implementação do protocolo com a realização do *middleware*, embora uma interpretação estrita destes à primeira vista possa soar como não muito satisfatórios, de outro espectro pode-se dizer que o protocolo e *middleware* apresentam um desempenho competitivo quando comparados a um banco de dados não-replicado. Em partes, os resultados obtidos são justificados pela adoção da abordagem baseada em *middleware* (CECCHET; CANDEA; AILAMAKI, 2008) para a implementação do protocolo, pois protocolos realizados sobre tal abordagem não são capazes de inferir diretamente sobre o processo de otimização envolvido na execução das operações da transação (KEMME; PERIS; PATIÑO-MARTÍNEZ, 2010). Por outro lado, se não fosse optado pela replicação baseada em *middleware*, não seria factível prover o suporte à heterogeneidade no âmbito do ambiente replicado.

Não obstante, o que deve ser evidenciado sobre os resultados é que todo o custo incidente da replicação pode ser justificado pelas garantias mais fortes que o protocolo e *middleware* oferecem para as aplicações. Por esta razão se acredita que os resultados são promissores e os custos proporcionam uma compensação adequada às vantagens oferecidas, pelo menos para algumas aplicações. Outrossim, a literatura mostra que alguns aspectos práticos relacionados ao desenvolvimento de protocolos de replicação de bancos de dados – tais como, métodos de propagação, semânticas de consistência, etc. – , bem como as metodologias empregadas para a avaliação de desempenho destes e seus respectivos resultados, tendem a apontar para vários *trade-offs* que podem induzir a melhores ou piores resultados em termos de algumas das métricas consideradas (KEMME; ALONSO, 2010). Neste sentido, o que resta é delegar ao projetista de uma aplicação, a decisão sobre aquilo que é mais adequado para um determinado sistema e/ou ambiente de computação.

Também é digno de nota, que um importante e surpreendente

resultado inédito trazido por esta tese foi aquele que demonstrou a susceptibilidade presente em um ambiente replicado de banco de dados que tolera faltas bizantinas, de incorrer num fenômeno/anomalia que pode afetar completamente a progressão do sistema. Este fenômeno/anomalia se deve a uma fragilidade que pode ser explorada sobre o controle de concorrência local das réplicas, quando é presumido que os clientes podem falhar de maneira bizantina, em que um conluio de clientes que exibem comportamento bizantino malicioso pode deturpar a execução do sistema.

Como segundo principal objetivo desta tese se buscou investigar um modelo de computação que permitisse resolver um problema real existente no âmbito de transações distribuídas, nomeadamente a validação atômica não-bloqueante (BABAOGLU; TOUEG, 1993). Neste sentido, foram estabelecidos os objetivos específicos enumerados por 2 e 3 (vide Seção 1.2), em que esta questão foi considerada através da discussão de aspectos envolvidos no projeto de protocolos de validação atômica, no sentido de conciliar tecnologias modernas de sistemas de computação com perspectivas práticas presentes na concepção de sistemas distribuídos tolerantes a faltas.

Não obstante, se buscou desenvolver uma solução inovadora que não apenas atendesse às especificações definidas pelo problema da validação atômica não-bloqueante (i.e., NBAC), mas que fosse principalmente factível de utilização em ambientes reais de sistemas de computação distribuída – já que a literatura mostra que as especificações originais do NBAC não são passíveis de atendimento pela maioria dos ambientes reais em sistemas de computação (GUERRAOUI; SCHIPER, 1995b; GUERRAOUI, 1995). Para tanto, esta tese introduziu um novo paradigma para a concepção de protocolos tolerantes a faltas bizantinas em sistemas de computação distribuída, que é concretizado através de um espaço de projeto inédito em termos da especificação e implementação do ambiente computacional, denominado por máquinas virtuais gêmeas (DETTONI et al., 2013a). Tal abordagem se mostra bastante plausível para o contexto de sistemas de computação distribuída, pois ela é baseada em uma tecnologia moderna, isto é, a virtualização, que na atualidade vem sendo reconhecida como um meio factível para a implementação de sistemas computacionais.

Em face da impossibilidade de resolução do problema da validação atômica fraca não-bloqueante com faltas bizantinas, a abordagem aprovisionada pela arquitetura de sistema especificada constituiu a base para a especificação de um protocolo de validação atômica fraca não-bloqueante tolerante a faltas bizantinas proposto por esta tese. Este

protocolo pode ser visto como o resultado de uma série de contribuições que resultaram neste novo paradigma envolvendo o ambiente de computação distribuída proposto para o contexto desta pesquisa. A arquitetura de sistema proposta permitiu resolver o problema da validação atômica fraca não-bloqueante através da separação das entidades presentes na especificação padrão do NBAC, não apenas em níveis distintos, mas também em implementações distintas e com diferentes suposições acerca do(s) modelo(s) de falhas tolerado(s).

Pela separação proposta nesta tese para a concretização de uma solução para o NBWAC, se tornou factível a especificação e implementação de uma abordagem pró-ativa para o agente que atua diretamente no protocolo NBWAC, cuja pró-atividade em questão consiste em uma técnica para detectar tentativas de faltas bizantinas no âmbito do sistema distribuído. A concretização desta abordagem pró-ativa se deu pela introdução de um novo agente denominado CTM (i.e., *Collaborative Transaction Manager*), cuja realização decorre da atuação conjunta de réplicas do agente TM (i.e., *Transaction Manager*) previsto na especificação do NBWAC. A atuação deste agente prevê a detecção antecipada do comportamento bizantino que pode ser exibido por algum TM do sítio que ele representa, ao passo que também elimina a possibilidade de um único ponto de falha no TM daquele sítio para evitar que agentes TMs faltosos venham a deturpar a execução do protocolo. Ao conhecimento dos proponentes, tal abordagem é inovadora, considerando que a literatura não dispõe de trabalhos no âmbito de algoritmos distribuídos tolerantes a faltas que usaram uma abordagem similar à proposta por esta tese.

Quanto aos últimos objetivos específicos traçados para esta tese, isto é, aqueles enumerados por 5 e 6, o quinto objetivo foi atendido e plenamente realizado através dos projetos e implementações dos protótipos para as soluções desenvolvidas no âmbito desta tese (vide Capítulo 6). O sexto e último objetivo será detalhado na próxima seção.

Por fim, a despeito dos resultados e contribuições realizadas, é lícito salientar que, infelizmente, não foi possível realizar tudo o que havia sido inicialmente previsto para esta tese. Em partes, isso deve ao fato de que as propostas iniciais culminaram por se expandir, de modo que os trabalhos tomaram dimensões amplas e algumas vezes discrepantes daquelas inicialmente previstas. Todavia, tem-se a convicção de que as contribuições trazidas por esta tese podem melhorar não apenas o processo de desenvolvimento de aplicações baseadas em bancos de dado relacionais, mas também o projeto de sistemas distribuídos tolerantes a faltas.

7.2 RESULTADOS E PUBLICAÇÕES

Dentre os trabalhos desenvolvidos no período de doutoramento e no âmbito desta tese, os principais resultados obtidos foram aqueles decorrentes do desenvolvimento (*i*) do protocolo de replicação tolerante a faltas bizantinas para bancos de dados relacionais; (*ii*) da arquitetura de *middleware* tolerante a faltas bizantinas para a replicação de bancos de dados relacionais; (*iii*) da arquitetura de sistema distribuído baseada na abordagem de máquinas virtuais gêmeas e; (*iv*) do protocolo de validação atômica fraca não-bloqueante tolerante a faltas bizantinas. Outrossim, tanto o projeto associado ao desenvolvimento das propostas como a realização dos protótipos podem ser elencados como contribuições desta tese. Não obstante, as principais contribuições, bem como alguns resultados intermediários e/ou correlatos obtidos através dos trabalhos realizados no período de doutoramento, foram objetos de publicação em periódicos, conferências, simpósios e workshops da área concernente às propostas. A relação destas publicações é disposta a seguir:

1. Byzantine Fault-Tolerant Transaction Processing for Replicated Databases. **Proceedings of the 11st IEEE International Symposium on Network Computing and Applications**, 2011 – autoria conjunta de Lau Cheuk Lung e Miguel Correia;

2. Protocolo Tolerante a Faltas Bizantinas para Bases de Dados Transacionais. **Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2011 – autoria conjunta de Lau Cheuk Lung e Miguel Correia;

3. RIST: Um Middleware para Replicação e Interoperabilidade de Sistemas Transacionais. **Anais do XXXVIII Seminário Integrado de Software e Hardware**, 2011 – autoria conjunta de Lau Cheuk Lung e Miguel Correia;

4. Byzantine Fault-Tolerant State Machine Replication with Twin Virtual Machines. **Proceedings of the 18th IEEE Symposium on Computers and Communications**, 2013 – autoria conjunta de Fernando Dettoni, Lau Cheuk Lung e Miguel Correia;

5. Using Virtualization Technology for Fault-Tolerant Replication in LAN. **Proceedings of the 8th International Conference on Dependability and Complex Systems**, 2013 – autoria conjunta de Fernando Dettoni e Lau Cheuk Lung;

6. Processamento Justo de Transações em Bancos de Dados Tolerantes a Faltas Bizantinas. **Anais do XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2013 – autoria conjunta de Lau Cheuk Lung e Miguel Correia;

7. Replicação de Máquina de Estados Tolerante a Faltas Bizantinas usando Máquinas Virtuais Gêmeas. **Anais do XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2013 – autoria conjunta de Fernando Dettoni, Lau Cheuk Lung e Miguel Correia;

8. Tolerância a Faltas Bizantinas usando Registradores Compartilhados e Distribuídos. **Anais do XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2013 – autoria conjunta de Marcelo Xavier Silva, Lau Cheuk Lung e Leandro Magnabosco;

9. DifATo - Difusão Atômica Tolerante a Faltas Bizantinas Baseada em Tecnologia de Virtualização. **Anais do XIV Workshop de Testes e Tolerância a Falhas**, 2013 – autoria conjunta de Marcelo Xavier Silva, Lau Cheuk Lung e Leandro Magnabosco;

10. Validação Atômica Não-Bloqueante com Faltas Bizantinas. **Anais do XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2014 – autoria conjunta de Lau Cheuk Lung, Miguel Correia e Valdir Stumm Júnior;

11. MITRA: Byzantine Fault-Tolerant Middleware for Transaction Processing on Replicated Databases. **ACM SIGMOD Record**, v. 43, n. 1, 2014 – autoria conjunta de Lau Cheuk Lung e Miguel Correia.

Embora as publicações abaixo elencadas não estejam diretamente relacionadas aos trabalhos desta tese, elas são fruto de outras pesquisas desenvolvidas/aperfeiçoadas durante o período de doutoramento:

1. Avaliando o Uso de Espaço de Tuplas como Alternativa para Implementação de Serviços Tolerantes a Faltas Bizantinas. **Anais do XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, 2010 – autoria conjunta de Lau Cheuk Lung e Alysson Neves Bessani;

2. On the Practicality to Implement Byzantine Fault Tolerant Services Based on Tuple Space. **Proceedings of the 28th IEEE In-**

ternational Conference on Advanced Information Networking and Applications, 2014 – autoria conjunta de Lau Cheuk Lung e Luciana de Oliveira Rech.

7.3 PERSPECTIVAS DE TRABALHOS FUTUROS

É evidente que os estudos realizados por esta tese certamente não dão por concluída a discussão acerca da tolerância a faltas bizantinas no âmbito de transações em bancos de dados. Embora as primeiras discussões acerca do tema em questão tenham ocorrido há bastante tempo, é notável que um maior interesse em investigação nesta área é algo relativamente recente. Algumas das limitações observadas durante o desenvolvimento das contribuições apresentadas nesta tese, abrem precedentes para novos trabalhos no mesmo campos de investigação. Uma perspectiva imediata consiste num refinamento do protótipo e aperfeiçoamento da implementação do *middleware*, para que possa ocorrer a portabilidade do mesmo para outros SGBDs que suportem o isolamento baseado em serialização, além do MySQL. Esta realização não apenas viabilizará, mas consolidará o suporte a heterogeneidade na composição do ambiente replicado, que compreende um aspecto bastante atraente no âmbito de um sistema que lida com faltas bizantinas.

Do mesmo modo, percebe-se a necessidade quanto ao aprimoramento das contribuições introduzidas nesta tese, no intuito de buscar soluções para outros problemas abertos no âmbito de transações e faltas bizantinas. Por exemplo, o teste de certificação e validação empregado no protocolo para assegurar a serialização das transações que são validadas; por vezes ele pode anular não espontaneamente uma transação em fase de terminação. Se não tratada adequadamente, tal situação pode incorrer num fenômeno conhecido por inanição (i.e., *starvation* – vide Seção 2.1.4.3), o que pode ser ainda mais agravado num ambiente de banco de dados sujeito a faltas bizantinas. Neste caso, investigar por uma solução capaz de lidar com o problema da inanição num ambiente de banco de dados tolerante a faltas bizantinas, é algo que também pode ser considerado como uma perspectiva de trabalho futuro desta tese.

Outrossim, no intuito de melhor explorar a arquitetura de sistema distribuído proposta, uma trabalho interessante consiste na investigação quanto à realização de uma generalização e expansão de tal arquitetura, para uma abordagem modular. Note que uma expansão/generalização da arquitetura para um caráter modular, permitiria

proporcionar um suporte para a especificação e implementação de uma série de protocolos fundamentais para o desenvolvimento de aplicações tolerantes a faltas, no contexto de sistemas de computação distribuída.

Um exemplo concreto, decorre da verificação acerca da possibilidade de especificação e implementação de um protocolo de pertinência de grupo sobre a arquitetura proposta (i.e., *group membership* (CRISTIAN, 1991)). Note que a pertinência de grupo é um problema bastante difícil de ser resolvido num ambiente de computação distribuída, tal como ocorre com a validação atômica não-bloqueante; uma vez que sua complexidade de solução não é nada trivial. De maneira análoga ao NBAC, a grande dificuldade por trás da solução da pertinência de grupo, decorre da imprecisão verificada quanto à detecção de falhas num ambiente de computação distribuída; o que pode acarretar no fracionamento de um grupo de processos. Tal aspecto pode ser encarado como algo não admissível, dependendo da natureza da aplicação, pois pode resultar em computações distribuídas não coerentes com a composição do ambiente.

Adicionalmente, a implementação de outros *benchmarks* padronizados pelo consórcio TPC (*http://www.tpc.org/information/benchmarks.asp*) também consiste numa possibilidade de realização futura. Tal avaliação poderia não apenas validar, mas também demonstrar a factibilidade do protocolo proposto, em termos de seu comportamento, quanto a cenários com condições de carga diversificada. Não obstante, um trabalho que soa como algo particularmente interessante, consiste na avaliação do comportamento e desempenho de protocolos baseados na arquitetura de sistema distribuído proposta, a partir de um ambiente de computação em nuvem (i.e., *cloud computing*) (NIST, 2011). É importante salientar, que o paradigma introduzido pelo conceito de computação em nuvem tem estimulado o surgimento de uma série de facilidades para, por exemplo, armazenar dados de maneira distribuída. Neste sentido, a manutenção de atributos, tais como confiabilidade, disponibilidade e tolerância a faltas; é requerida também, por parte de aplicações que lidam com transações sobre dados armazenados em provedores de nuvens. O que, portanto, caracteriza uma oportunidade para trabalhos futuros.

REFERÊNCIAS

ABD-EL-MALEK, M. et al. Fault-scalable Byzantine fault-tolerant services. In: **Proceedings of the 20th ACM Symposium on Operating Systems Principles**. [S.l.: s.n.], 2005. p. 59–74.

ABDALLAH, M.; PUCHERAL, P. A Low-Cost Non-Blocking Atomic Commitment Protocol for Asynchronous Systems. In: **Proceedings of the 11th International Conference on Parallel and Distributed Computing and Systems**. [S.l.: s.n.], 1999. p. 911–918.

ADYA, A. **Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions**. Tese (Ph.D.) — MIT, Cambridge, MA, USA, mar 1999. Technical Report MIT/LCS/TR-786.

ANSI. **American National Standard for Information Systems Stardard X3.135-1992: Database Language SQL**. Nov 1992.

ATTAR, R.; BERNSTEIN, P. A.; GOODMAN, N. Site initialization, recovery, and backup in a distributed database system. **IEEE Transactions on Software Engineering**, SE-10, n. 6, p. 645–650, 1984.

ATTIYA, H.; WELCH, J. **Distributed Computing: Fundamentals, Simulations and Advanced Topics**. 2. ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2004. ISBN 0-471-45324-2.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. **IEEE Transactions on Dependable and Secure Computing**, v. 1, n. 1, p. 11–33, 2004.

BABAOGLU, O.; TOUEG, S. Non-Blocking Atomic Commitment. In: MULLENDER, S. (Ed.). **Distributed systems**. 2. ed. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1993. p. 147–168. ISBN 0-201-62427-3.

BALDONI, R.; MARCHETTI, C.; TERMINI, A. Active software replication through a three-tier approach. In: **Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems**. [S.l.: s.n.], 2002. p. 109–118.

BARBORAK, M.; DAHBURA, A.; MALEK, M. The consensus problem in fault-tolerant computing. **ACM Computing Surveys**, v. 25, n. 2, p. 171–220, 1993.

BASU, A.; CHARRON-BOST, B.; TOUEG, S. Simulating reliable links with unreliable links in the presence of process crashes. In: **Proceedings of the 10th International Workshop on Distributed Algorithms**. [S.l.: s.n.], 1996. p. 105–122.

BELLARE, M.; CANETTI, R.; KRAWCZYK, H. Keying hash functions for message authentication. In: **Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology**. [S.l.: s.n.], 1996. p. 1–15.

BEN-OR, M. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: **Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing**. [S.l.: s.n.], 1983. p. 27–30.

BERENSON, H. et al. A Critique of ANSI SQL Isolation Levels. **SIGMOD Record**, v. 24, n. 2, p. 1–10, 1995.

BERNSTEIN, P.; NEWCOMER, E. **Principles of Transaction Processing: for the systems professional**. 2. ed. Burlington, MA, USA: Morgan Kaufmann Publishers Inc., 2009. ISBN 978-1-55860-623-4.

BERNSTEIN, P. A.; GOODMAN, N. Concurrency control in distributed database systems. **ACM Computing Surveys**, v. 13, p. 185–221, 1981.

BERNSTEIN, P. A.; GOODMAN, N. Multiversion concurrency control - theory and algorithms. **ACM Transactions on Database Systems**, v. 8, n. 4, p. 465–483, 1983.

BERNSTEIN, P. A.; GOODMAN, N. Serializability theory for replicated databases. **Journal of Computer and System Sciences**, v. 31, n. 3, p. 355–374, 1985.

BERNSTEIN, P. A.; HADZILACOS, V.; GOODMAN, N. **Concurrency Control and Recovery in Database Systems**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1987. ISBN 0-201-10715-5.

BESSANI, A.; SOUSA, J.; ALCHIERI, E. State Machine Replication for the Masses with BFT-SMaRt. In: **Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2014. p. 355–362.

BESSANI, A. N.; ALCHIERI, E. A guided tour on the theory and practice of state machine replication. In: **Minicursos do XXXII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, Brasil: Sociedade Brasileira de Computação, 2014. p. 146–193. ISSN 2177-4978.

BIRMAN, K. P. The process group approach to reliable distributed computing. **Communications of the ACM**, v. 36, n. 12, p. 37–53, 1993.

BIRMAN, K. P.; JOSEPH, T. A. Reliable communication in the presence of failures. **ACM Transactions on Computer Systems**, v. 5, n. 1, p. 47–76, 1987.

BRACHA, G.; TOUEG, S. Asynchronous consensus and broadcast protocols. **Journal of the ACM**, v. 32, n. 4, p. 824–840, 1985.

BRASILEIRO, F. et al. Consensus in one communication step. In: **Proceedings of the 6th International Conference on Parallel Computing Technologies**. [S.l.: s.n.], 2001. p. 42–50.

BRAYNER, A. **Transaction Management in Multidatabase Systems**. Aachen, DE: Shaker-Verlag, 1999. (Berichte aus der Informatik). ISBN 3-8265-6142-2.

BRESSOUD, T. C.; SCHNEIDER, F. B. Hypervisor-based fault tolerance. In: **Proceedings of the 15th ACM Symposium on Operating Systems Principles**. [S.l.: s.n.], 1995. p. 1–11.

BREWER, E. CAP twelve years later: How the "rules"have changed. **Computer**, v. 45, n. 2, p. 23–29, 2012.

CACHIN, C. State machine replication with byzantine faults. In: CHARRON-BOST, B.; PEDONE, F.; SCHIPER, A. (Ed.). **Replication: Theory and Practice**. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 5959). p. 169–184. ISBN 978-3-642-11293-5.

CACHIN, C.; GUERRAOUI, R.; RODRIGUES, L. **Introduction to Reliable and Secure Distributed Programming**. 2. ed.

Berlin, Germany: Springer-Verlag Berlin Heidelberg, 2011. ISBN 978-3-642-15259-7.

CACHIN, C.; KURSAWE, K.; SHOUP, V. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. **Journal of Cryptology**, v. 18, n. 3, p. 219–246, 2005.

CAHILL, M.; RÖHM, U.; FEKETE, A. Serializable isolation for snapshot databases. In: **Proceedings of the ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 2008. p. 729–738.

CASTRO, M.; LISKOV, B. Practical Byzantine fault tolerance. In: **Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation**. [S.l.: s.n.], 1999. p. 173–186.

CASTRO, M.; LISKOV, B. Byzantine fault tolerance can be fast. In: **Proceedings of the 31th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2001. p. 513–518.

CECCHET, E.; CANDEA, G.; AILAMAKI, A. Middleware-based database replication: The gaps between theory and practice. In: **Proceedings of the ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 2008. p. 739–752.

CECCHET, E.; MARGUERITE, J.; ZWAENEPOEL, W. C-JDBC: Flexible database clustering middleware. In: **Proceedings of the USENIX Annual Technical Conference**. [S.l.: s.n.], 2004. p. 9–18.

CHANDRA, T. D.; TOUEG, S. Unreliable failure detectors for reliable distributed systems. **Journal of the ACM**, v. 43, n. 2, p. 225–267, 1996.

CHANDY, K. M.; LAMPORT, L. Distributed snapshots: determining global states of distributed systems. **ACM Transactions on Computer Systems**, v. 3, n. 1, p. 63–75, 1985.

CHARRON-BOST, B. Agreement problems in fault-tolerant distributed systems. In: **Proceedings of the 28th Conference on Current Trends in Theory and Practice of Informatics Piestany**. [S.l.: s.n.], 2001. p. 10–32.

CHARRON-BOST, B.; DÉFAGO, X.; SCHIPER, A. Broadcasting messages in fault-tolerant distributed systems: the benefit of handling input-triggered and output-triggered suspicions differently. In: **Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems**. [S.l.: s.n.], 2002. p. 244–249.

CHARRON-BOST, B.; SCHIPER, A. Uniform consensus is harder than consensus. **Journal of Algorithms**, v. 51, n. 1, p. 15–37, 2004.

CHOCKLER, G.; MALKHI, D.; REITER, M. K. Backoff protocols for distributed mutual exclusion and ordering. In: **Proceedings of the 21st IEEE International Conference on Distributed Computing Systems**. [S.l.: s.n.], 2001. p. 11–20.

CHUN, B.-G.; MANIATIS, P.; SHENKER, S. Diverse replication for single-machine byzantine-fault tolerance. In: **Proceedings of the USENIX Annual Technical Conference**. [S.l.: s.n.], 2008. p. 287–292.

COAN, B. A.; WELCH, J. L. Transaction commit in a realistic timing model. **Distributed Computing**, v. 4, n. 2, p. 87–103, 1990.

CODD, E. F. A Relational Model of Data for Large Shared Data Banks. **Communications of the ACM**, v. 13, n. 6, p. 377–387, 1970.

CONNOLLY, T. M.; BEGG, C. E. **Database Systems: A Practical Approach to Design, Implementation, and Management**. 4. ed. Upper Saddle River, NJ, USA: Pearson Education Inc., 2005. ISBN 0-32121-025-5.

CORREIA JR., A. et al. Group-based replication of on-line transaction processing servers. In: **Proceedings of the 2nd Latin-American Symposium on Dependable Computing**. [S.l.: s.n.], 2005. p. 245–260.

CORREIA, M. et al. Efficient byzantine-resilient reliable multicast on a hybrid failure model. In: **Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems**. [S.l.: s.n.], 2002. p. 2–11.

CORREIA, M.; NEVES, N. F.; VERISSIMO, P. How to tolerate half less one byzantine nodes in practical distributed systems. In: **Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems**. [S.l.: s.n.], 2004. p. 174–183.

CORREIA, M.; NEVES, N. F.; VERÍSSIMO, P. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. **The Computer Journal**, v. 49, n. 1, p. 82–96, 2006.

CORREIA, M.; VERÍSSIMO, P.; NEVES, N. F. The design of a cots real-time distributed security kernel. In: **Proceedings of the 4th European Dependable Computing Conference**. [S.l.: s.n.], 2002. p. 234–252.

COULOURIS, G. et al. **Distributed Systems: Concepts and Design**. 5. ed. Upper Saddle River, NJ, USA: Pearson Education Inc., 2012. ISBN 0-13-214301-1.

COWLING, J. et al. HQ-Replication: A hybrid quorum protocol for byzantine fault tolerance. In: **Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementations**. [S.l.: s.n.], 2006. p. 177–190.

CRISTIAN, F. Reaching agreement on processor-group membrship in synchronous distributed systems. **Distributed Computing**, v. 4, n. 4, p. 175–187, 1991.

DATE, C. J. **An Introduction to Database Systems**. 8. ed. Upper Saddle River, NJ, USA: Pearson Education Inc., 2004. ISBN 0-321-18956-6.

DESMEDT, Y. Society and group oriented cryptography: A new concept. In: **Proceedings of the Conference on Theory and Application of Cryptographic Techniques: Advances in Cryptology**. [S.l.: s.n.], 1987. p. 120–127.

DETTONI, F. et al. Byzantine fault-tolerant state machine replication with twin virtual machines. In: **Proceedings of the 8th IEEE Symposium on Computers and Communications**. [S.l.: s.n.], 2013. p. 398–403.

DETTONI, F. et al. Replicação de máquina de estados tolerante a faltas bizantinas usando máquinas virtuais gêmeas. In: **Anais do XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.: s.n.], 2013. p. 965–978.

DIERKS, T. **The Transport Layer Security (TLS) Protocol Version 1.2 (RFC 5246)**. [S.l.], Aug 2008.

DOLEV, D.; STRONG, H. R. Authenticated algorithms for byzantine agreement. **SIAM Journal on Computing**, v. 12, n. 4, p. 656–666, 1983.

DOUDOU, A.; GARBINATO, B.; GUERRAOUI, R. Encapsulating failure detection: From crash-stop to Byzantine failures. In: **Proceedings of the 7th Ada-Europe International Conference on Reliable Software Technologies**. [S.l.: s.n.], 2002. p. 24–50.

DRISCOLL, K. et al. Byzantine fault tolerance, from theory to reality. In: **Proceedings of the 22nd International Conference on Computer Safety, Reliability and Security**. [S.l.: s.n.], 2003. p. 235–248.

DUBOURDIEUX, D. Implementation of distributed transactions. In: **Proceedings of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks**. [S.l.: s.n.], 1982. p. 81–94.

DWORK, C.; LYNCH, N. A.; STOCKMEYER, L. Consensus in the presence of partial synchrony. **Journal of ACM**, v. 35, n. 2, p. 288–322, 1988.

DÉFAGO, X.; SCHIPER, A.; URBÁN, P. Total order broadcast and multicast algorithms: Taxonomy and survey. **ACM Computing Surveys**, v. 36, n. 4, p. 372–421, 2004.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 6. ed. Boston, MA, USA: Addison-Wesley Publishing Company, 2010. ISBN 978-0-136-08620-8.

ESWARAN, K. P. et al. The notions of consistency and predicate locks in a database system. **Communications of the ACM**, v. 19, n. 11, p. 624–633, 1976.

FEKETE, A. Allocating isolation levels to transactions. In: **Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems**. [S.l.: s.n.], 2005. p. 206–215.

FEKETE, A. et al. Making snapshot isolation serializable. **ACM Transactions on Database Systems**, v. 30, n. 2, p. 492–528, 2005.

FEKETE, A.; O'NEIL, E.; O'NEIL, P. A Read-only Transaction Anomaly Under Snapshot Isolation. **SIGMOD Record**, v. 33, n. 3, p. 12–14, 2004.

FELDMAN, P. A practical scheme for non-interactive verifiable secret sharing. In: **Proceedings of the 28th IEEE Annual Symposium on Foundations of Computer Science**. [S.l.: s.n.], 1987. p. 427–438.

FIREBIRD, F. I. **Firebird Database Server**. 2014. Disponível em `http://www.firebirdsql.org/`. Acessado em 02 de abril de 2014.

FISCHER, M. J.; LYNCH, N. A.; PATERSON, M. S. Impossibility of distributed consensus with one faulty process. **Journal of the ACM**, v. 32, n. 2, p. 374–382, 1985.

FISHER, M.; ELLIS, J.; BRUCE, J. **JDBC API Tutorial and Reference**. 3. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0-321-17384-8.

FRITZKE JR., U. et al. Consensus-based fault-tolerant total order multicast. **IEEE Transactions on Parallel and Distributed Systems**, v. 12, n. 2, p. 147–156, 2001.

GARCIA-MOLINA, H.; PITTELLI, F.; DAVIDSON, S. Is byzantine agreement useful in a distributed database? In: **Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems**. [S.l.: s.n.], 1984. p. 61–69.

GARCIA-MOLINA, H.; PITTELLI, F.; DAVIDSON, S. Applications of byzantine agreement in database systems. **ACM Transactions on Database Systems**, v. 11, n. 1, p. 27–47, 1986.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Database Systems - The Complete Book**. 2. ed. Upper Saddle River, NJ, USA: Pearson Education Inc., 2009. ISBN 978-0-13-187325-4.

GARCIA, R.; RODRIGUES, R.; PREGUIÇA, N. Efficient middleware for Byzantine fault-tolerant database replication. In: **Proceedings of the 6th European Conference on Computer Systems**. [S.l.: s.n.], 2011. p. 107–122.

GARFINKEL, T.; ROSENBLUM, M. A virtual machine introspection based architecture for intrusion detection. In: **Proceedings of the 10th Annual Symposium on Network and Distributed System Security**. [S.l.: s.n.], 2003. p. 253–285.

GÄRTNER, F. C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. **ACM Computing Surveys**, v. 31, n. 1, p. 1–26, 1999.

GASHI, I.; POPOV, P.; STRIGINI, L. Fault diversity among off-the-shelf SQL database servers. In: **Proceedings of the 34th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2004. p. 389–398.

GASHI, I.; POPOV, P. T.; STRIGINI, L. Fault tolerance via diversity for off-the-shelf products: A study with SQL database servers. **IEEE Transactions on Dependable and Secure Computing**, v. 4, n. 4, p. 280–294, 2007.

GRAY, J. Notes on data base operating systems. In: BAYER, R.; GRAHAM, R. M.; SEEGMüLLER, G. (Ed.). **Operating Systems, An Advanced Course**. [S.l.]: Springer Berlin Heidelberg, 1978, (Lecture Notes in Computer Science, v. 60). p. 393–481. ISBN 978-3-540-08755-7.

GRAY, J. The transaction concept: virtues and limitations (invited paper). In: **Proceedings of the 7th International Conference on Very Large Data Bases**. [S.l.: s.n.], 1981. p. 144–154.

GRAY, J. et al. The dangers of replication and a solution. In: **Proceedings of the ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 1996. p. 173–182.

GRAY, J.; LORIE, R. A.; PUTZOLU, G. R. Granularity of locks in a shared data base. In: **Proceedings of the 1st International Conference on Very Large Data Bases**. [S.l.: s.n.], 1975. p. 428–451.

GRAY, J. et al. Granularity of locks and degrees of consistency in a shared data base. In: **IFIP Working Conference on Modelling in Data Base Management Systems**. [S.l.: s.n.], 1976. p. 365–394.

GRAY, J.; REUTER, A. **Transaction Processing: Concepts and Techniques**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992. ISBN 1-55860-190-2.

GREVE, F.; NARZUL, J.-P. L. Um protocolo de validação atômica não-bloqueante eficiente. In: **Anais do XX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.: s.n.], 2002. p. 309–323.

GRIGORIK, I. **High-Performance Browser Networking**. Sebastopol, CA, USA: O'Reilly Media, Inc., 2013. ISBN 978-1-449-34476-4.

GUERRAOUI, R. Revisiting the relationship between non-blocking atomic commitment and consensus. In: **Proceedings of the 9th International Workshop on Distributed Algorithms**. [S.l.: s.n.], 1995. p. 87–100.

GUERRAOUI, R. et al. The next 700 BFT protocols. In: **Proceedings of the 5th European Conference on Computer Systems**. [S.l.: s.n.], 2010. p. 363–376.

GUERRAOUI, R.; LARREA, M.; SCHIPER, A. Non blocking atomic commitment with an unreliable failure detector. In: **Proceedings of the 14th IEEE Symposium on Reliable Distributed Systems**. [S.l.: s.n.], 1995. p. 41–50.

GUERRAOUI, R.; LARREA, M.; SCHIPER, A. Reducing the cost for non-blocking in atomic commitment. In: **Proceedings of the 16th International Conference on Distributed Computing Systems**. [S.l.: s.n.], 1996. p. 692–697.

GUERRAOUI, R.; RAYNAL, M. A leader election protocol for eventually synchronous shared memory systems. In: **Proceedings of the 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems**. [S.l.: s.n.], 2006. p. 1–6.

GUERRAOUI, R.; SCHIPER, A. The decentralized non-blocking atomic commitment protocol. In: **Proceedings of the 7th Symposium on Parallel and Distributed Processing**. [S.l.: s.n.], 1995. p. 2–9.

GUERRAOUI, R.; SCHIPER, A. A generic multicast primitive to support transactions on replicated objects in distributed systems. In: **Proceedings of the 5th IEEE Workshop on Future Trends of Distributed Computing Systems**. [S.l.: s.n.], 1995. p. 334–342.

GUERRAOUI, R.; SCHIPER, A. Software-based replication for fault tolerance. **Computer**, v. 30, n. 4, p. 68–74, 1997.

HADZILACOS, V. On the relationship between the atomic commitment and consensus problems. In: SIMONS, B.; SPECTOR, A. (Ed.). **Fault-Tolerant Distributed Computing**. [S.l.: s.n.], 1990, (Lecture Notes in Computer Science, v. 448). p. 201–208.

HADZILACOS, V.; TOUEG, S. Distributed systems. In: MULLENDER, S. (Ed.). 2. ed. New York, NY, USA: ACM

Press/Addison-Wesley Publishing Co., 1993. cap. Fault-tolerant Broadcasts and Related Problems, p. 97–145. ISBN 0-201-62427-3.

HADZILACOS, V.; TOUEG, S. **A Modular Approach to the Specification and Implementation of Fault-Tolerant Broadcasts and Related Problems**. [S.l.], may 1994.

HAERDER, T.; REUTER, A. Principles of transaction-oriented database recovery. **ACM Computing Surveys**, v. 15, n. 4, p. 287–317, 1983.

HARRINGTON, J. L. **Object-oriented Database Design Clearly Explained**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. ISBN 0-12-326428-6.

HARRIS, J. **Green Computing and Green IT Best Practices on Regulations and Industry Initiatives, Virtualization, Power Management, Materials Recycling and Telecommuting**. London, UK: Emereo Pty Ltd, 2008. ISBN 978-1-921-52344-1.

HELLERSTEIN, J. M.; STONEBRAKER, M.; HAMILTON, J. Architecture of a Database System. **Foundations and Trends in Databases**, v. 1, n. 2, p. 141–259, 2007.

HERLIHY, M. P.; WING, J. M. Linearizability: A correctness condition for concurrent objects. **ACM Transactions on Programming Languages and Systems**, v. 12, n. 3, p. 463–492, 1990.

HITCHENS, R. **Java NIO**. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002. ISBN 0-596-00288-2.

HORSTMANN, C.; CORNELL, G. **Core Java 2 - Volume 1: Fundamentals**. 7. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN 01-314-8202-5.

HURFIN, M.; TRONEL, F. A solution to atomic commitment based on an extended consensus protocol. In: **Proceedings of the 6th IEEE Workshop on Future Trends of Distributed Computing Systems**. [S.l.: s.n.], 1997. p. 98–103.

INAYAT, Q. ul A.; EZHILCHELVAN, P. A performance study on the signal-on-fail approach to imposing total order in the streets of byzantium. In: **Proceedings of the 36th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2006. p. 578–590.

INTEL, C. **Virtualization and Cloud Computing**. 2013. Disponível em `http://www.intel.com/content/dam/www/public-/us/en/documents/guides/cloud-computing-virtualization-building-private-iaas-guide.pdf`.

JAIN, R. K. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. New York, NY, USA: John Wiley & Sons, Inc., 1991. ISBN 978-0-47-150336-1.

JALOTE, P. **Fault Tolerance in Distributed Systems**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994. ISBN 0-13-301367-7.

JOHNSON, R.; PANDIS, I.; AILAMAKI, A. Improving OLTP scalability using speculative lock inheritance. **Proceedings of the VLDB Endowment**, v. 2, n. 1, p. 479–489, 2009.

KEMME, B.; ALONSO, G. Database replication: A tale of research across communities. **The Proceedings of the VLDB Endowment**, v. 3, n. 1, p. 5–12, 2010.

KEMME, B.; PERIS, R. J.; PATIÑO-MARTÍNEZ, M. **Database Replication**. San Francisco, CA, USA: Morgan & Claypool Publishers LLC, 2010. (Synthesis Lectures on Data Management). ISBN 978-16-08453-81-8.

KING, S. T.; DUNLAP, G. W.; CHEN, P. M. Operating system support for virtual machines. In: **Proceedings of the 2003 USENIX Annual Technical Conference**. [S.l.: s.n.], 2003. p. 6–19.

KOTLA, R. et al. Zyzzyva: speculative byzantine fault tolerance. In: **Proceedings of 21st ACM Symposium on Operating Systems Principles**. [S.l.: s.n.], 2007. p. 45–58.

KSHEMKALYANI, A. D.; SINGHAL, M. **Distributed Computing: Principles, Algorithms, and Systems**. New York, NY, USA: Cambridge University Press, 2008. ISBN 978-0-521-87634-6.

KUNG, H. T.; ROBINSON, J. T. On optimistic methods for concurrency control. **ACM Transactions on Database Systems**, v. 6, p. 213–226, 1981.

KVM, P. **Kernel Based Virtual Machine**. 2014. Disponível em `http://www.linux-kvm.org/page/Main_Page`. Acessado em 15 de maio de 2014.

LAMPORT, L. Time, clocks, and the ordering of events in a distributed system. **Communications of the ACM**, v. 21, n. 7, p. 558–565, 1978.

LAMPORT, L. On interprocess communication. **Distributed Computing**, v. 1, n. 2, p. 77–101, 1986.

LAMPORT, L. Paxos made simple. **ACM SIGACT News**, v. 32, n. 4, p. 18–25, 2001.

LAMPORT, L.; MALKHI, D.; ZHOU, L. Reconfiguring a state machine. **SIGACT News**, v. 41, n. 1, p. 63–73, 2010.

LAMPORT, L.; SHOSTAK, R.; PEASE, M. The Byzantine generals problem. **ACM Transactions on Programing Languages and Systems**, v. 4, n. 3, p. 382–401, 1982.

LAMPSON, B. W. Atomic transactions. In: **Distributed Systems - Architecture and Implementation, An Advanced Course**. London, UK: Springer-Verlag, 1981, (Lecture Notes in Computer Science, v. 105). p. 246–265. ISBN 3-540-10571-9.

LAMPSON, B. W. Lazy and speculative execution in computer systems. In: SHVARTSMAN, A. A. (Ed.). **Proceedings of the 10th International Conference on Principles of Distributed Systems**. [S.l.]: Springer-Verlag, 2006. (Lecture Notes in Computer Science, v. 4305), p. 1–2. ISBN 3-540-49990-3.

LEVASSEUR, J. et al. Unmodified device driver reuse and improved system dependability via virtual machines. In: **Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation**. [S.l.: s.n.], 2004. p. 17–30.

LIMA, E.; REIS, E. **C# e .NET - Guia do Desenvolvedor**. Rio de Janeiro, RJ, Brasil: Editora Campus Ltda, 2002. ISBN 85-352-0954-9.

LIU, M. L.; AGRAWAL, D.; ABBADI, A. E. The performance of two phase commit protocols in the presence of site failures. **Distributed and Parallel Databases**, v. 6, n. 2, p. 157–182, 1998.

LUIZ, A. F.; LUNG, L. C.; CORREIA, M. Mitra: Byzantine fault-tolerant middleware for transaction processing on replicated databases. **SIGMOD Record**, v. 43, n. 1, p. 32–38, 2014.

LYNCH, N. A. **Distributed Algorithms**. San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc., 1996. ISBN 1-55860-348-4.

MAGYARI, A.; GENGE, B.; HALLER, P. Certificate-based single sign-on mechanism for multi-platform distributed systems. **Acta Universitatis Sapientiae - Electrical and Mechanical Engineering**, v. 1, p. 113–123, 2009.

MAHAJAN, S.; SINGHAL, R. Azvasa:- byzantine fault tolerant distributed commit with proactive recovery. In: **Proceedings of the 2nd International Conference on Emerging Trends in Engineering and Technology**. [S.l.: s.n.], 2009. p. 659–663.

MALKHI, D.; REITER, M. Byzantine quorum systems. In: **Proceedings of the 29th Annual ACM Symposium on Theory of Computing**. [S.l.: s.n.], 1997. p. 569–578.

MEYER, F. J.; PRADHAN, D. K. Consensus with dual failure modes. **IEEE Transactions on Parallel and Distributed Systems**, v. 2, n. 2, p. 214–222, 1991.

MICHIE, D. Memo functions and machine learning. **Nature**, v. 218, n. 5136, p. 19–22, 1968.

MICROSOFT. **Microsoft SQL Server**. 2014. Disponível em `http://www.microsoft.com/en-us/server-cloud/products/sql-server`. Acessado em 02 de abril de 2014.

MOHAN, C.; LINDSAY, B. Efficient commit protocols for the tree of processes model of distributed transactions. In: **Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing**. [S.l.: s.n.], 1983. p. 76–88.

MOHAN, C.; LINDSAY, B.; OBERMARCK, R. Transaction Management in the R* Distributed Database Management System. **ACM Transactions on Database Systems**, v. 11, n. 4, p. 378–396, 1986.

MOHAN, C.; STRONG, R.; FINKELSTEIN, S. Method for distributed transaction commit and recovery using byzantine agreement within clusters of processors. In: **Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing**. [S.l.: s.n.], 1983. p. 89–103.

MOSTÉFAOUI, A.; RAYNAL, M. Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In: **Proceedings of the 13th International Symposium on Distributed Computing**. [S.l.: s.n.], 1999. p. 49–63.

MPOELENG, D.; EZHILCHELVAN, P.; SPEIRS, N. From crash tolerance to authenticated Byzantine tolerance: A structured approach, the cost and benefits. In: **Proceedings of the 33rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. [S.l.: s.n.], 2003. p. 227–236.

MURO, S.; KAMEDA, T.; MINOURA, T. Multi-version concurrency control scheme for a database system. **Journal of Computer and System Sciences**, v. 29, n. 2, p. 207–224, 1984.

NARASIMHAN, P.; MOSER, L. E.; MELLIAR-SMITH, P. M. Exploiting the internet inter-orb protocol interface to provide corba with fault tolerance. In: **Proceedings of the 3rd USENIX Conference on Object-Oriented Technologies and Systems**. [S.l.: s.n.], 1997. p. 6–15.

NIGHTINGALE, E. B.; DOUCEUR, J. R.; ORGOVAN, V. Cycles, cells and platters: An empirical analysisof hardware failures on a million consumer PCs. In: **Proceedings of the 6th European Conference on Computer Systems**. [S.l.: s.n.], 2011. p. 343–356.

NIST. **National Institute of Standards and Technology Definition of Cloud Computing**. 2011. Disponível em `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`. Acessado em 20 de julho de 2014.

NORMANN, R.; OSTBY, L. T. A theoretical study of 'snapshot isolation'. In: **Proceedings of the 13th International Conference on Database Theory**. [S.l.: s.n.], 2010. p. 44–49.

OBELHEIRO, R. R.; BESSANI, A. N.; LUNG, L. C. Analisando a viabilidade da implementação prática de sistemas tolerantes a intrusões. In: **Anais do V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. [S.l.: s.n.], 2005. p. 99–112.

OBERMARCK, R. Distributed Deadlock Detection Algorithm. **ACM Transactions on Database Systems**, v. 7, n. 2, p. 187–208, 1982.

ORACLE, C. **MySQL Database**. 2014. Disponível em `http://www.mysql.com/`. Acessado em 02 de abril de 2014.

ORACLE, C. **Oracle Database Software**. 2014. Disponível em `http://www.oracle.com/us/products/database`. Acessado em 02 de abril de 2014.

PAPADIMITRIOU, C. H. The Serializability of Concurrent Database Updates. **Journal of the ACM**, v. 26, n. 4, p. 631–653, 1979.

PAPADIMITRIOU, C. H. **The Theory of Database Concurrency Control**. New York, NY, USA: Computer Science Press, Inc., 1986. ISBN 0-88175-027-1.

PARK, S.-H.; LEE, J.-Y.; YU, S.-C. Non-blocking atomic commitment algorithm in asynchronous distributed systems with unreliable failure detectors. In: **Proceedings of the 10th International Conference on Information Technology**. [S.l.: s.n.], 2013. p. 33–38.

PARMELEE, R. P. et al. Virtual storage and virtual machine concepts. **IBM Systems Journal**, v. 11, n. 2, p. 99–130, 1972.

PEASE, M.; SHOSTAK, R.; LAMPORT, L. Reaching agreement in the presence of faults. **Journal of ACM**, v. 27, n. 2, p. 228–234, 1980.

PEDONE, F.; GUERRAOUI, R. On transaction liveness in replicated databases. In: **Proceedings of the Pacific Rim International Symposium on Fault-Tolerant Systems**. [S.l.: s.n.], 1997. p. 104–109.

PEDONE, F.; GUERRAOUI, R.; SCHIPER, A. The database state machine approach. **Distributed and Parallel Databases**, v. 14, n. 1, p. 71–98, 2003.

PEDONE, F.; SCHIPER, N. Byzantine fault-tolerant deferred update replication. **Journal of the Brazilian Computer Society**, v. 18, n. 1, p. 3–18, 2012.

PEDONE, F.; SCHIPER, N.; ARMENDÁRIZ-IÑIGO, J. Byzantine fault-tolerant deferred update replication. In: **Proceedings of the 5th Latin-American Symposium on Dependable Computing**. [S.l.: s.n.], 2011. p. 7–16.

POKORNY, J. Nosql databases: A step to database scalability in web environment. In: **Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services**. [S.l.: s.n.], 2011. p. 278–283.

POPEK, G. J.; GOLDBERG, R. P. Formal requirements for virtualizable third generation architectures. **Communications of the ACM**, v. 17, n. 7, p. 412–421, 1974.

POSTGRESQL, G. D. G. **PostgreSQL**. 2014. Disponível em `http://www.postgresql.org`. Acessado em 02 de abril de 2014.

PREGUIÇA, N. et al. Byzantium: Byzantine-fault-tolerant database replication providing snapshot isolation. In: **Proceedings of the 4th USENIX Workshop on Hot Topics in System Dependability**. [S.l.: s.n.], 2008. p. 9–14.

RABUSKE, M. A. **Introdução à Teoria dos Grafos**. Florianópolis, SC, Brasil: Editora da UFSC, 1992.

RAHIMI, S. K.; HAUG, F. S. **Distributed Database Management Systems: A Practical Approach**. New Jersey, NJ, USA: Wiley-IEEE Computer Society Press, 2010. ISBN 978-0-470-40745-5.

RAMAKRISHNAN, R.; GEHRKE, J. **Database Management Systems**. 3. ed. New York, NY, USA: McGraw-Hill, Inc., 2003. ISBN 978-0-07246-563-1.

RANDELL, B. System structure for software fault tolerance. **IEEE Transactions on Software Engineering**, v. 1, n. 2, p. 221–232, 1975.

RAYNAL, M.; SINGHAL, M. Mastering agreement problems in distributed systems. **IEEE Software**, v. 18, p. 40–47, 2001.

REED, D. P. **Naming and Synchronization in a Descentralized Computer System**. Tese (PhD Thesis) — Massachusetts Institute of Technology, Boston, MA, US, 1978.

REISER, H. P.; KAPITZA, R. Hypervisor-based efficient proactive recovery. In: **Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems**. [S.l.: s.n.], 2007. p. 83–92.

RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. M. A method for obtaining digital signatures and public-key cryptosystems. **Communications of the ACM**, v. 21, n. 2, p. 120–126, 1978.

ROSENBLUM, M. The reincarnation of virtual machines. **ACM Queue**, v. 2, n. 5, p. 34–40, 2004.

ROTHERMEL, K.; PAPPE, S. Open commit protocols tolerating commission failures. **ACM Transactions on Database Systems**, v. 18, n. 2, p. 289–332, 1993.

RYAN, T. P. **Estatística Moderna para Engenharia**. Rio de Janeiro, RJ, Brasil: Elsevier Editora Ltda., 2009. ISBN 978-0-470-08187-7.

SCHEINERMAN, E. R. **Matemática Discreta - Uma Introdução**. São Paulo, SP, Brasil: Thomson Learning Edições Ltda., 2006. ISBN 978-8-52210-291-4.

SCHIPER, A.; RAYNAL, M. From group communication to transactions in distributed systems. **Communications of the ACM**, v. 39, p. 84–87, 1996.

SCHLICHTING, R. D.; SCHNEIDER, F. B. Fail-stop processors: An approach to designing fault-tolerant computing systems. **ACM Trans. Comput. Syst.**, v. 1, n. 3, p. 222–238, 1983.

SCHNEIDER, F. B. Implementing fault-tolerant service using the state machine approach: A tutorial. **ACM Computing Surveys**, v. 22, n. 4, p. 299–319, 1990.

SCHNEIDER, F. B. Distributed systems. In: MULLENDER, S. (Ed.). 2. ed. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1993. cap. What Good Are Models and What Models Are Good?, p. 17–26. ISBN 0-201-62427-3.

SCHNEIDER, F. B.; ZHOU, L. Implementing trustworthy services using replicated state machines. **IEEE Security & Privacy**, v. 3, n. 5, p. 34–43, 2005.

SCHROEDER, B.; GIBSON, G. A. Understanding failures in petascale computers. **Journal of Physics: Conference Series**, v. 78, n. 1, p. 12–22, 2007.

SCHROEDER, B.; PINHEIRO, E.; WEBER, W.-D. Dram errors in the wild: A large-scale field study. In: **Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems**. [S.l.: s.n.], 2009. p. 193–204.

SHAHZA, M. et al. In search of efficient non blocking atomic commit protocol. In: **Proceedings of the 9th Annual PostGraduate Symposium on The Convergence of Telecommunications, Networking and Broadcasting**. [S.l.: s.n.], 2008. p. 1–6.

SHOUP, V. Practical threshold signatures. In: **Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques**. [S.l.: s.n.], 2000. p. 207–220.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database System Concepts**. 6. ed. New York, NY, USA: McGraw-Hill, Inc., 2011. ISBN 978-0-07-352332-3.

SILVA, E. L. da; MENEZES, E. M. **Metodologia da Pesquisa e Elaboração de Dissertação**. 4. ed. Florianópolis: Universidade Federal de Santa Catarina, 2005.

SINGH, A. et al. BFT Protocols Under Fire. In: **Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation**. [S.l.: s.n.], 2008. p. 189–204.

SIPPU, S.; SOISALON-SOININEN, E. **Transaction Processing: Management of the Logical Database and its Underlying Physical Structure**. Switzerland: Springer International Publishing, 2014. (Data-Centric Systems and Applications). ISBN 978-3-319-12292-2.

SKEEN, D. Nonblocking commit protocols. In: **Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 1981. p. 133–142.

SMITH, J. E.; NAIR, R. The architecture of virtual machines. **Computer**, v. 38, n. 5, p. 32–38, 2005.

STEARNS, R. E.; LEWIS, P. M.; ROSENKRANTZ, D. J. Concurrency Control for Database Systems. In: **Proceedings of the 17th Annual Symposium on Foundations of Computer Science**. [S.l.: s.n.], 1976. p. 19–32.

STEARNS, R. E.; ROSENKRANTZ, D. J. Distributed database concurrency controls using before-values. In: **Proceedings of the ACM SIGMOD International Conference on Management of Data**. [S.l.: s.n.], 1981. p. 74–83.

STONEBRAKER, M. Concurrency control and consistency of multiple copies of data in distributed ingres. **IEEE Transactions on Software Engineering**, v. 5, n. 3, p. 188–194, 1979.

STONEBRAKER, M. et al. The end of an architectural era: (it's time for a complete rewrite). In: **Proceedings of the 33rd International Conference on Very Large Data Bases**. [S.l.: s.n.], 2007. p. 1150–1160.

STUMM JR., V. et al. Intrusion tolerant services through virtualization: A shared memory approach. In: **Proceedings of the 24th International Conference on Advanced Information Networking and Applications**. [S.l.: s.n.], 2010. p. 768–774.

SUN, M. I. **NFS: Network File System Protocol Specification (RFC 1094)**. Mar 1989. IETF Request For Comments.

SWANSON, A. K. **Development and Management of a Computer-centered Data Base**. [S.l.], nov 1963.

TANENBAUM, A. S. **Modern Operating Systems**. 3. ed. Upper Saddle River, NJ, USA: Prentice Hall, Inc., 2007. ISBN 978-0-136-00663-3.

TANENBAUM, A. S.; STEEN, M. van. **Distributed Systems: Principles and Paradigms**. 2. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0-132-39227-5.

THOMAS, R. H. A majority consensus approach to concurrency control for multiple copy databases. **ACM Transactions on Database Systems**, v. 4, n. 2, p. 180–209, 1979.

THOMASIAN, A. Concurrency control: Methods, performance, and analysis. **ACM Computing Surveys**, v. 30, n. 1, p. 70–119, 1998.

TIWARI, S. **Professional NoSQL**. Indianapolis, IN, USA: John Wiley & Sons, Inc., 2011. ISBN 978-0-470-94224-6.

TOUEG, S. Randomized Byzantine agreements. In: **Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing**. [S.l.: s.n.], 1984. p. 163–178.

TPC. **Transaction Processing Performance Council: TPC-C OLTP Benchmark**. 2014. Disponível em `http://www.tpc.org/tpcc/spec/tpcc_current.pdf`. Acessado em 27 de agosto de 2014.

VANDIVER, B. et al. Tolerating Byzantine faults in transaction processing systems using commit barrier scheduling. In: **Proceedings of the 21st ACM Symposium on Operating Systems Principles**. [S.l.: s.n.], 2007. p. 59–72.

VERONESE, G. S. et al. Spin one's wheels? byzantine fault tolerance with a spinning primary. In: **Proceedings of the 28th IEEE**

International Symposium on Reliable Distributed Systems. [S.l.: s.n.], 2009. p. 135–144.

VERONESE, G. S. et al. Ebawa: Efficient byzantine agreement for wide-area networks. In: **Proceedings of the 12th IEEE International Symposium on High-Assurance Systems Engineering**. [S.l.: s.n.], 2010. p. 10–19.

VERÍSSIMO, P. Travelling through wormholes: A new look at distributed systems models. **SIGACT News**, v. 37, n. 1, p. 66–81, 2006.

VMWARE, I. **Timekeeping in VMware Virtual Machines**. 2011. Disponível em `http://www.vmware.com/files/pdf/Timekeeping-In-VirtualMachines.pdf`. Acessado em 15 de maio de 2014.

VMWARE, I. **VMware Virtualization**. 2014. Disponível em `http://www.vmware.com/`. Acessado em 15 de maio de 2014.

VOGELS, W. Eventually consistent. **Communications of the ACM**, v. 52, n. 1, p. 40–44, 2009.

WEIKUM, G.; VOSSEN, G. **Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery**. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002. ISBN 978-1-55860-508-4.

WENSLEY, J. H. et al. Sift: Design and analysis of a fault-tolerant computer for aircraft control. **Proceedings of the IEEE**, v. 66, n. 10, p. 1240–1255, 1978.

WIESMANN, M. et al. Understanding replication in databases and distributed systems. In: **Proceedings of the 20th International Conference on Distributed Computing Systems**. [S.l.: s.n.], 2000. p. 464–474.

WILLIAMS, K. et al. **Professional XML Databases**. Birmingham, UK: Wrox Press Ltd., 2000. ISBN 1-86-100358-7.

WOOD, T. et al. ZZ and the art of practical BFT execution. In: **Proceedings of the 6th European Conference on Computer Systems**. [S.l.: s.n.], 2011. p. 123–138.

XEN, P. **Xen Project**. 2014. Disponível em `http://www.xenproject.org/`. Acessado em 15 de maio de 2014.

YIN, J. et al. Separating agreement from execution for Byzantine fault tolerant services. In: **Proceedings of the 19th ACM Symposium on Operating Systems Principles**. [S.l.: s.n.], 2003. p. 253–267.

ZHAO, W. A byzantine fault tolerant distributed commit protocol. In: **Proceedings of the 3rd IEEE International Symposium on Dependable, Autonomic and Secure Computing**. [S.l.: s.n.], 2007. p. 37–46.

ZIELINSKI, P. **Paxos at War**. [S.l.], jun 2004.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. 3. ed. New York, NY, USA: Springer New York, 2011. ISBN 978-1-4419-8833-1.