

**UNIVERSIDADE FEDERAL DE SANTA CATARINA - UFSC  
CURSO SISTEMA DE INFORMAÇÃO**

**TRABALHO DE CONCLUSÃO DE CURSO**

**ARQUITETURA DE TOLERÂNCIA A FALHAS EM SISTEMAS WEB**

Rafael de Souza Andrade  
Professor Orientador: Lau Cheuk Lung  
Área de Concentração: Sistema de Informação

Florianópolis, SC  
2009

**RAFAEL DE SOUZA ANDRADE**

**ARQUITETURA DE TOLERÂNCIA A FALHAS EM SISTEMAS WEB**

Trabalho de Conclusão de Curso apresentado Universidade Federal de Santa Catarina (UFSC), Florianópolis/SC, como requisito para conclusão do Curso de Graduação em Sistema de Informação.

Orientador: Lau Cheuk Lung

Florianópolis, SC  
2009

## TOLERANCIA A FALHAS EM UM SISTEMA WEB 2.0

**RAFAEL DE SOUZA ANDRADE**

Este Trabalho de Conclusão de Curso (TCC) foi julgado adequado para obtenção da graduação em Sistema de Informação e aprovado em sua forma final junto a Universidade Federal de Santa Catarina - UFSC, Florianópolis/SC.

.....  
Maria Marta Leite  
Coordenadora de Cursos

Apresentada à Banca Examinadora, integrada pelos Professores:

.....  
Lau Cheuk Lung, Dr  
Orientador

.....  
Carlos Barros Montez  
Banca Examinadora

.....  
Frank Augusto Siqueira  
Banca Examinadora

“... A imaginação é mais importante que a ciência, porque a ciência é limitada, ao passo que a imaginação abrange o mundo inteiro...”

*“Albert Einstein”*

## AGRADECIMENTOS

## RESUMO

ANDRADE, Rafael de Souza. Tolerância a falhas em um Sistema Web 2.0 . 2009 f. Trabalho de Conclusão de Curso (Sistema de Informação) – Universidade Federal de Santa Catarina - UFSC, Florianópolis, 2009.

O presente trabalho tem como objetivo desenvolver uma arquitetura que garanta a tolerância a falhas em sistemas Web 2.0. Para o desenvolvimento da arquitetura proposta foram utilizados conceitos de clusterização de servidores de aplicações (Jboss) e também replicação de dados em Banco de dados Mysql, garantindo assim principalmente tolerância a falhas de hardware, para manter a alta disponibilidade de sistemas Web 2.0 mesmo com algum de seus servidores fora do ar. O resultado almejado foi alcançado ao se verificar que a arquitetura proposta garante que mesmo em caso de falha de um servidor de aplicação ou de algum servidor de banco de dados, o sistema continua com o comportamento padrão, sem que o usuário perceba o problema ocorrido.

Palavras-chave: Tolerância a falhas, Banco de dados Distribuídos, Clusterização de Servidores de Aplicações, Web 2.0.

## ABSTRACT

ANDRADE, Rafael de Souza. Tolerância a falhas em um Sistema Web 2.0 . 2009 f. Trabalho de Conclusão de Curso (Sistema de Informação) – Universidade Federal de Santa Catarina - UFSC, Florianópolis, 2009.

This present paper aims to develop an architecture that ensures fault tolerance in Web 2.0 systems. For the development of the proposed architecture were used concepts of clustering application servers (Jboss) and also data replication in MySql Database servers, that architecture ensures mainly hardware faults, to maintain high availability of Web 2.0 systems even with some of its servers down. The desired result was succeed from the fact that the proposed architecture ensures that even in case of failure of an application server or any database server, the system continues with the default behavior without the User notices the problem occurred.

Key words: Fault Tolerance, Distributed Databases, Application Server Clustering, Web 2.0

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>9</b>
1.1	CONTEXTUALIZAÇÃO DO TEMA E PROBLEMA	9
1.2	OBJETIVOS	9
1.2.1	Objetivo Geral	10
1.2.2	Objetivos Específicos	10
1.3	JUSTIFICATIVA	10
1.4	ESTRUTURA DO ESTUDO	11
<b>2</b>	<b>REVISÃO DA LITERATURA</b>	<b>12</b>
2.1	TOLERÂNCIA A FALHAS	12
2.1.1	Dependabilidade	13
2.1.2	Tolerância a falhas	19
2.1.3	Fases na Tolerância a Falhas	20
2.1.4	Mascaramento de Falhas	23
2.1.5	Redundância	23
2.2	BANCO DE DADOS	30
2.2.1	Banco de dados Relacional	31
2.3	BANDO DE DADOS DISTRIBUÍDOS	33
2.3.1	Porque usar Sistemas de Banco de Dados Distribuído?	34
2.3.2	Características de um sistema de Banco de dados Distribuído	34
2.3.3	Complicações de sistemas em banco de dados distribuídos	36
2.3.4	Arquitetura de SGBD Distribuídos	36
2.4	APLICAÇÕES WEB	37
2.4.1	Tolerância a falhas em aplicações web 2.0	38
2.4.2	Clusterização de Servidores de Aplicação(Jboss)	39
<b>3</b>	<b>DESENVOLVIMENTO DO SISTEMA</b>	<b>40</b>
3.1	REGRAS DE NEGÓCIO DO SISTEMA	40
3.2	BANCO DE DADOS MYSQL	42
3.3	CLUTERIZAÇÃO DE MYSQL	42
3.4	SERVIDOR DE APLICAÇÕES JBOSS	42
3.4.1	Clusterização com Jboss	43
3.4.2	Configurando o cluster e os nodes	43
3.4.3	Arquiteturas de serviço	47
3.4.4	Políticas de Balanceamento	49
3.4.5	Replicações de Sessões e Load Balance para aplicações Web	50
3.5	ARQUITETURA PROPOSTA	52
3.5.1	Camada Cliente	53
3.5.2	Camada de Aplicação	54
3.5.3	Camada de Banco de Dados	56
3.5.4	Configuração da arquitetura proposta	56



3.5.5 Avaliação de Desempenho .....	64
<b>4 TRABALHOS FUTUROS.....</b>	<b>67</b>
<b>5 CONSIDERAÇÕES FINAIS .....</b>	<b>68</b>
<b>6 REFERÊNCIAS.....</b>	<b>69</b>
<b>7 ANEXOS .....</b>	<b>73</b>

# 1 INTRODUÇÃO

Este trabalho visa desenvolver uma aplicação que utiliza banco de dados distribuídos, e também garante tolerância a falhas utilizando cluster em servidores de aplicações, que seria o sistema de *clustering* do servidor de aplicações *Jboss*.

## 1.1 CONTEXTUALIZAÇÃO DO TEMA E PROBLEMA

Estamos presenciando um crescimento cada vez maior da Internet, aonde praticamente tudo em nossas vidas pode ser feito pela mesma. E isso fez com que surgisse uma forte demanda de novas plataformas para desenvolvimento de aplicações Web, por exemplo, Internet banking, e-commerce, e-learning e etc. Todos estes são exemplos reais que aplicações web têm crescido muito e que continuarão assim no futuro.

Atualmente estamos vivendo a fase em que todos os sistemas, além de robustos e confiáveis, devem ser amigáveis para os usuários finais. A chamada Web 2.0, aonde a interação do usuário com seus sistemas se torna fácil, amigável e muito mais interessante, é um exemplo dessa nova tendência.

Porém, atualmente a maioria das aplicações desenvolvidas não oferece suporte com garantias de tolerância a falhas. Como por exemplo, se um de seus servidores desliga por alguma falha, o sistema deixa de funcionar. Desta forma, a comunidade científica da área tem proposto diversas soluções tecnológicas para desenvolvimento de aplicações tolerante a faltas para garantir que mesmo em caso de algumas falhas parciais os sistemas possam continuar operacionais.

Garantias de tolerância a faltas têm sido muito estudadas, mas somente para sistemas críticos, e hoje em dia tem aumentado a necessidade de se ter essas garantias em sistemas web, para o funcionamento correto dos mesmos.

Como podemos garantir que um sistema é tolerante a falhas?

## 1.2 OBJETIVOS

O objetivo é definido como alvo que se pretende atingir e também orienta a revisão da literatura e a metodologia do estudo.

A seguir apresentam-se os objetivos que este trabalho atingirá.

### 1.2.1 Objetivo Geral

Desenvolver uma aplicação web utilizando banco de dados distribuídos e tolerância a falhas através de clusterização de servidores de aplicação.

### 1.2.2 Objetivos Específicos

- a) Desenvolver a revisão de literatura;
- b) Realizar um *benchmark* de banco de dados distribuídos para escolher o que tiver melhor desempenho;
- c) Desenvolver um sistema tolerante a falhas aplicando técnicas de *clustering* de servidores de aplicações;
- d) Fazer análise completa de um sistema *web* utilizando as técnicas e diagramas UML;
- e) Desenvolver um sistema web completo para um usuário final.

## 1.3 JUSTIFICATIVA

Aplicações Web estão cada dia mais presentes em nossas vidas, porém, até hoje, não é possível garantir o seu correto funcionamento em cem por cento dos casos. Devido a esse fato, estão sendo sempre estudadas e aperfeiçoadas formas de garantir o melhor funcionamento de sistemas.

Até pouco tempo atrás, os únicos sistemas que usavam técnicas de tolerância a falhas eram sistemas críticos, em que sua execução não poderia ser parada nunca. Porém atualmente esta necessidade vem crescendo cada vez em todos sistemas conhecidos, desde sistemas de bancos a sistemas internos de empresas.

Nossa dependência nesses sistemas fez quem o os mesmo não pudessem parar de ser executados de forma alguma, e por isso estão sendo estudadas técnicas que garantam cem por cento do funcionamento de novos sistemas, principalmente sistemas Web.

Estão sendo lançadas novas idéias de garantir o funcionamento destes sistemas, assim como a arquitetura proposta neste trabalho, que visa garantir o funcionamento de sistemas web 2.0 a falhas, principalmente falhas de hardware.

## 1.4 ESTRUTURA DO ESTUDO

Neste presente trabalho de conclusão serão abordados os seguintes temas:

No capítulo I, Introdução.

No capítulo II, Revisão da Literatura – Tolerância a falhas, Banco de Dados, Banco de dados Distribuídos, Aplicações Web.

No capítulo III, Desenvolvimento do sistema.

No capítulo IV, Trabalhos Futuros.

No capítulo V, Considerações Finais.

No capítulo VI, Referências.

No capítulo VII, Anexos.

## 2 REVISÃO DA LITERATURA

Nesta etapa, apresentam-se os conceitos que dão o embasamento teórico necessário para o entendimento do tema desenvolvido.

### 2.1 TOLERÂNCIA A FALHAS

Hoje em dia, computadores e sistemas atuam cada vez mais na vida das pessoas, em quase todas as tarefas que são efetuadas pela sociedade há interação com algum sistema. Porém, garantir que o funcionamento de um sistema após sofrer falhas se torna cada vez mais essencial para todos esses sistemas.

Todos os sistemas estão sujeitos a falhas, sejam elas falhas de software ou falhas de hardware. Entretanto, o mais importante é a consequência dessas falhas, ou melhor, saber tratar estas falhas para não ter consequência no sistema em andamento.

Tolerância a falhas não era tratado por sistemas convencionais até pouco tempo, e sendo só utilizados por sistemas críticos, como por exemplo, aviões, sondas entre outros. Mas, devido à grande popularização dos sistemas e das redes de computadores, fez com que houvesse um aumento na dependência tecnológica em grande parte da população. Segundo WEBER (1999, p.12), “[...] Com a espantosa popularização de redes, fornecendo os mais variados serviços, aumentou a dependência tecnológica de uma grande parcela da população aos serviços oferecidos. Falhas nesses serviços podem ser catastróficas para a segurança da população ou para a imagem e reputação das empresas.”

Otávio Alcântara (2008) entende que tolerância a falhas em sistemas de computação é:

“Alvo de estudo desde a década de 50. O primeiro pesquisador a apresentar trabalhos na área foi *Jonh Von Neuman* que desenvolveu trabalhos de como aumentar a confiabilidade aplicando redundância de circuitos. Outros pesquisadores também contribuíram bastante para a fundamentação da área, é válido citar os nomes de *Avizienis* que projetou o computador STAR (*Self-Testing And Repairing*) um dos primeiros projetos a ter como foco um longo período de missão sem intervenções humanas.”

Confiabilidade e disponibilidade em sistemas estão sendo cada vez mais desejados, o que criou uma nova necessidade para os desenvolvedores de sistemas.

O termo ainda utilizado para definir um sistema com essas características, porém está entrando em desuso, é Tolerância a Falhas, que foi utilizado pela primeira vez em 1967 por Avizienis, e a partir disto tem sido muito usado por toda comunidade. Atualmente está sendo cada vez mais utilizado um novo termo para definir a confiabilidade de disponibilidade em sistemas, este termo é Dependabilidade, que será melhor definido a diante neste trabalho.

### 2.1.1 Dependabilidade

Dependabilidade, apesar de ser um termo relativamente novo no mercado, é uma das principais características buscadas por sistemas hoje em dia, pois dependabilidade é o atributo de um sistema que diz o quanto um sistema é tolerante a falhas.

De acordo com Algirdas Avizienis, (200?) sistemas são formados por cinco atributos básicos: funcionalidade, usabilidade, performance (desempenho), custo, e dependabilidade.

Dependabilidade em sistemas computacionais é a habilidade que um sistema tem de prover um serviço de forma confiável e garantindo a integridade dos dados enviados pelo mesmo.

Este atributo de um sistema não pode ser medido numericamente, porém os atributos que a dependabilidade possui podem ser mensurados, e assim podemos calcular a confiança de um sistema.

De acordo com Pradhan, (1996), “os principais atributos da dependabilidade são: confiabilidade, disponibilidade, segurança de funcionamento (*safety*), segurança (*security*), manutenibilidade (*maintainability*), testabilidade e comprometimento do desempenho (*performability*).“

A figura abaixo mostra uma árvore com os atributos e significados da dependabilidade.

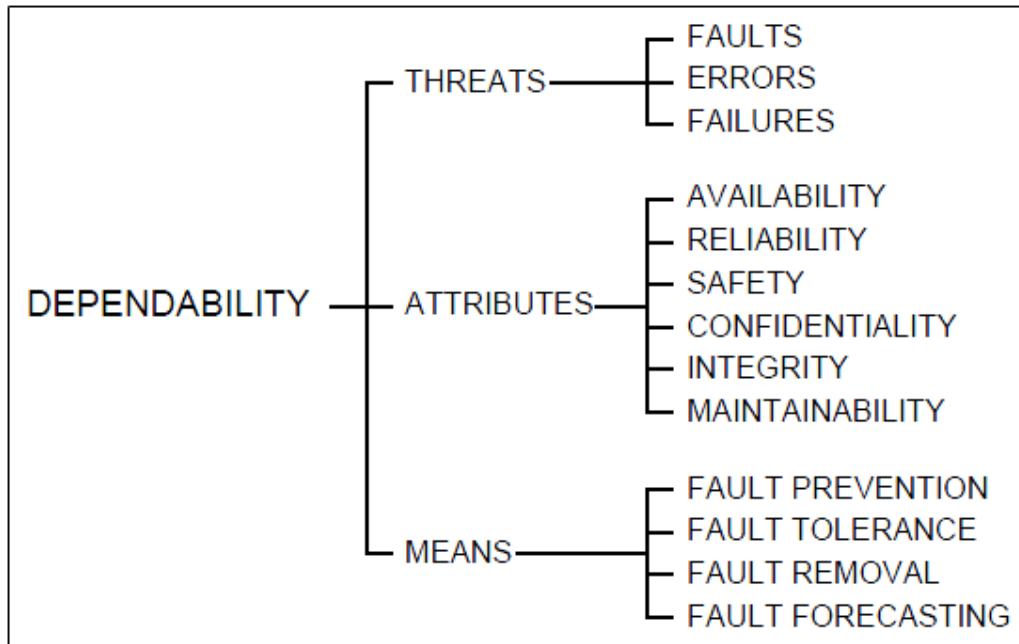


Figura 1: Atributos da Dependabilidade  
 Fonte: Algirdas Aviz̃ ienis

Estes atributos da dependabilidade serão melhor especificados abaixo:

a) Confiabilidade – este atributo se refere à característica de o sistema atender a especificação feita para o mesmo, durante um tempo previsto, essa característica fica melhor especificada por Weber (2002) aonde diz que “[...] é a capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período.”

Confiabilidade é a principal característica da dependabilidade, qualquer sistema que deseja ter alta dependabilidade deve ter alta confiabilidade.

Exemplos de sistemas que precisam desta característica são sistemas de aviação e exploração espacial, pois são sistemas em que não podem ter falhas nem no menor espaço de tempo, e também porque são sistemas que não é possível se arrumar em tempo real.

Como dito anteriormente, este atributo pode ser mensurado, e por isso se trata de um dado probabilístico, porém está característica não pode ser confundida com disponibilidade, pois como o exemplo de aviões, estes sistemas podem ter alta confiabilidade quando estão operando, mas precisam de correções e reparos a cada pouso, fazendo assim com que não tenha uma alta disponibilidade.

b) Disponibilidade – esta característica está mais visível em sistemas que precisa estar disponíveis sempre que for necessário, ou seja, não há um tempo fixo de funcionamento, devendo estar sempre disponível. Exemplos de sistemas que possuem alta disponibilidade são: servidores, banco de dados, entre outros.

De acordo com (Weber, 2002), “[...] Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, quando está sendo reparado, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço”.

Esta característica, assim como confiabilidade, é muito desejada por usuários atualmente, pois todos querem sistemas confiáveis e estejam sempre disponíveis para eles.

c) Segurança de funcionamento - esta característica consiste em garantir que o sistema não trabalhe de forma incorreta, ou seja, sempre garantirá que o sistema trabalhará de forma correta, e se o mesmo detectar falha no seu funcionamento, ele o levará para um estado aonde não causará danos e nem de repostas erradas.

Um exemplo de sistema com alta segurança de funcionamento são os trens, que quando o sistema identifica algum problema de funcionamento, o mesmo controla sua aceleração e para em lugar seguro.

Esta característica também pode ser medida quantitativamente, fazendo assim com que se possa calcular a segurança de funcionamento de um sistema.

d) Comprometimento de Desempenho - esta característica consiste em garantir que o sistema, mesmo após falhas, consiga garantir certo desempenho primeiramente acordado, ou seja, o desempenho do sistema poderá piorar após falhas, mas somente até um certo ponto, garantindo seu desempenho.

e) Manutenibilidade - esta característica significa a facilidade que se tem para dar manutenção ao sistema, ou seja, mensurar o tempo que leva para corrigir um defeito no sistema, lembrando que corrigir o problema consiste em localizar, recuperar e colocar em operação novamente.

f) Testabilidade - de acordo com (Weber, 2002), “testabilidade é a capacidade de testar certos atributos internos ao sistema ou facilidade de realizar certos testes”.



Como foi visto, dependabilidade é uma característica cada vez mais buscada em sistemas, que antes era somente vista em sistemas críticos, mas que agora está sendo buscada por empresas para garantir a qualidade de seus serviços, e a partir disso viu-se a necessidade de se trazer estes atributos para sistemas comerciais usado mundialmente.

### 2.1.1.1 Formas de se Alcançar Dependabilidade

Para garantir que um sistema tenha o atributo da dependabilidade, é necessário implementar métodos e técnicas, pois através destes podemos garantir que um sistema possua dependabilidade ou não. Tolerância a falhas, na verdade, é um atributo de Dependabilidade, porém como atualmente usuários não estão acostumados com o termo citado, o texto foi inicialmente chamado de tolerância a falhas. Além de tolerância a falhas, sistema que tem dependabilidade possui outras características, entre elas podemos listar: prevenção de falhas, remoção de falhas, previsão de falhas, e também, tolerância a falhas.

A tabela abaixo mostra a classificação de dos métodos e técnicas para alcançar a dependabilidade.

<b>Técnica</b>	<b>Função</b>
Prevenção de falhas	Impede a ocorrência ou introdução de falhas. Envolve a seleção de metodologias de projeto e de tecnologias adequadas para seus componentes.
Tolerância a falhas	Fornece o serviço esperado mesmo na presença de falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização, confinamento, recuperação, reconfiguração, tratamento.
Validação	Remoção de falhas, verificação da presença de falhas.
Previsão de falhas	Estimativas sobre presença de falhas e estimativas sobre conseqüências de falhas.

Tabela 1 – Técnicas para alcançar dependabilidade.

Fonte: Weber, T. Tolerância a falhas: conceitos e exemplos.

Antes de se estudar especificamente cada uma destas técnicas é necessário entender como falhas ocorrem em sistemas, existem três tipos básicos de problemas que são definidos de acordo com seus níveis em relação ao sistema. Esses tipos básicos são chamados de: falha, erro e defeito.

A parte de tolerância a falhas será especificada separadamente, pois é a técnica que deseja se aprofundar mais neste trabalho.

### 2.1.1.2 Falha, erro e defeito

De acordo com Otávio Alcântara (2008):

Falha é um defeito físico, imperfeição que atinge um componente de *hardware* ou *software*. Podemos tomar como exemplo uma solda fria que gera um mau contato entre dois componentes, ou um algoritmo que gera um laço infinito, ou seja, falhas estão associadas ao universo físico dos sistemas.

Uma falha pode gerar um erro no sistema, o erro é a representação da falha na parte de informação do sistema. Normalmente se tem um erro quando uma falha corrompe uma informação entre outros.

Segundo Wez (2005), “O defeito (*failure*) é um desvio na especificação, e ocorre em consequência de um erro. Nesta etapa, o sistema já está em estado errôneo, a informação já está corrompida e conseqüentemente irá gerar um defeito. O defeito é percebido pelo usuário, por isso os defeitos estão associados ao universo do usuário.”

Na figura abaixo tem um exemplo simplificado do funcionamento de falha, erro e defeito.

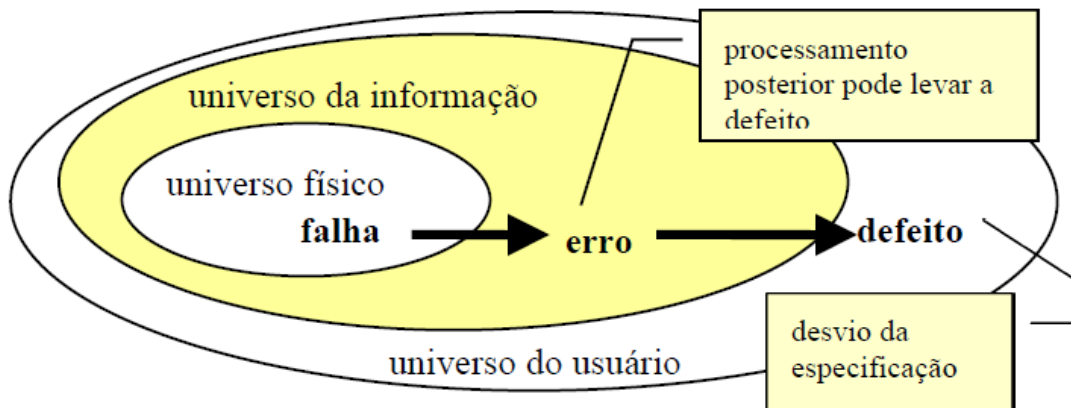


Figura 2: Modelo de três universos  
Fonte: Weber, T. Tolerância a falhas: conceitos e exemplos.

As falhas podem ser categorizadas em duas formas, as falhas físicas e as falhas humanas.

As falhas físicas são as falhas relacionadas ao hardware do sistema, relacionada aos componentes deste sistema, atualmente a maioria das falhas físicas já são tratadas pelos sistemas, ou seja, não é visível para o usuário.

O principal problema atualmente para os sistemas, principalmente os sistemas críticos são as falhas de software, que são falhas humanas, decorrentes de

especificações erradas, implementações erradas ou qualquer outro problema relacionado a uma falha humana.

A seguir mostra-se uma tabela comparativa com as estatísticas referentes às disponibilidades de acordo com o tipo do sistema.

<b>Sistemas tradicionais</b>		<b>Redes cliente-servidor (não tolerantes a falhas)</b>
<b>Não tolerante a falhas</b>	<b>Tolerante a falhas</b>	
Mean time to failure: 6 a 12 semanas Indisponibilidade após defeito: 1 a 4 h	Mean time to failure: 21 anos (Tandem)	Disponibilidade média: 98%
<b>Defeitos:</b>	<b>Defeitos:</b>	<b>Defeitos:</b>
Hardware 50%	Software 65%	Projeto 60%
Software 25%	Operações 10%	Operações 24%
Comunicações / ambiente 15%	Hardware 8%	Físicos 16%
Operações 10%	Ambiente 7%	

Tabela 2: Causas usuais de defeitos em sistemas de computação

Fonte: Weber, T. Tolerância a falhas: conceitos e exemplos.

### 2.1.1.3 Prevenção de Falhas

Prevenção de falhas é garantido por técnicas de controle de qualidade que são implementadas durante a fase de projeto e desenvolvimento do software ou hardware. Estas técnicas garantem que o desenvolvimento estará correto, e assim, fará com que não haja falhas, exemplos de técnicas para prevenção a falhas, é uma especificação correta de um sistema, de forma formal, ou regras rígidas para desenvolver um hardware.

### 2.1.1.4 Remoção de Falhas

Remoção de falhas é uma técnica usada tanto na parte de desenvolvimento do sistema como também durante seu ciclo de vida de operação.

Para utilizar esta técnica durante o desenvolvimento de um sistema, é necessário seguir três etapas, são elas: verificação, diagnóstico e correção.

Verificação é a fase aonde se checa se o sistema em desenvolvimento esta de acordo com o que foi especificado para o mesmo, e se for verificado que o sistema não está sendo desenvolvido de forma correta, segue-se para a fase de diagnóstico, que

consiste em se verificar o que não esta de acordo com o que deveria estar, e assim, corrigir os erros.

#### *2.1.1.5 Previsão de falhas*

Previsão de falhas consiste na técnica aonde se faz estatísticas qualitativas e quantitativas para se fazer testes nos sistemas, e se verificar o comportamento do mesmo.

A partir destes testes estatísticos, faz-se um estudo para verificar se o comportamento do sistema de alguma forma ficará de forma incorreta no futuro, e assim poder corrigi-los antes mesmo de o erro acontecer.

#### **2.1.2 Tolerância a falhas**

Como dito anteriormente, a fase de tolerância a falhas consiste em que uma mensagem enviada pelo sistema chegará de forma correta mesmo após sofrer falhas, normalmente esta técnica utiliza técnicas de detecção de erros e de técnicas de recuperação de sistema.

De acordo com Weber (2002), prevenção e remoção de falhas:

“[...] não são suficientes quando o sistema exige alta confiabilidade ou alta disponibilidade. Nesses casos o sistema deve ser construído usando técnicas de tolerância a falhas. Essas técnicas garantem funcionamento correto do sistema mesmo na ocorrência de falhas e são todas baseadas em redundância, exigindo componentes adicionais ou algoritmos especiais”.

A tolerância a falhas pode ser classificada de duas formas diferentes:

- Mascaramento;
- Detecção, localização e reconfiguração.

Mascaramento de falhas consiste em fazer com que o sistema, mesmo em estado errôneo, não mostra para o usuário, ou seja, mesmo com um erro, o funcionamento do sistema continua de forma normal. Esta técnica é mais utilizada em sistemas que não podem ficar em estado errôneo esperando para serem corrigidos.

Já detecção, localização e reconfiguração são quando um erro é encontrado, e automaticamente existem mecanismos que fazem a localização do erro a sua correção, como dito anteriormente, esta técnica é mais usada para sistemas que pode permanecer em estado errôneo por um período de tempo até ser corrigido.

A tolerância a falhas é separada em fases, essas fases serão melhores especificadas nos próximos tópicos.

### **2.1.3 Fases na Tolerância a Falhas**

Como já dito anteriormente, um sistema tolerante a falhas visa principalmente alcançar a dependabilidade, ou seja, busca garantir que um sistema funcione corretamente mesmo após que o sistema tenha alguma falha. Para garantir que um sistema tenha a dependabilidade, existem fases definidas entre a detecção de falhas e a correção das mesmas, segundo (ANDERSON, T.; LEE, 1981), estas fases são definidas como: Detecção de erros, confinamento e avaliação de erros, recuperação de erros e tratamento de falhas.

Estas fases serão melhores explicadas nos tópicos abaixo.

a) Detecção de erros - trata-se de umas das principais características a ser analisada na tolerância a falhas, pois com a detecção de erros é que se pode definir a confiabilidade de algum sistema.

As falhas em sistemas dificilmente são detectadas diretamente em sistemas, é necessário que aconteça erros no sistema para assim identificá-los. Para isso é necessário fazer muitos testes no sistema, e assim identificar o maior número possível de falhas no sistema, o ideal seria identificar todas as falhas, porém sabe-se que hoje em dia, devido à complexidade dos sistemas em desenvolvimento é quase impossível atingir tal medida.

Para garantir a eficiência desta fase, é necessário que os testes estejam de acordo com algumas propriedades, elas são:

- Um teste ideal deve se basear somente na especificação do sistema, ou seja, sem conhecimento de como foi desenvolvido, este tipo de teste é chamado de teste “caixa preta”;
- O teste deve ser completo e correto, ou seja, todos os erros possíveis devem ser detectados;
- Teste deve ser independente do sistema, pois eles também podem falhar, e isso não pode se refletir no sistema.

O teste do tipo caixa preta é o mais importante, pois desconsidera a implementação do sistema e garante somente que ele deve funcionar de acordo com o que foi especificado.

Existem avaliações de várias formas de se testar os sistemas, as principais formas de teste existentes são:

- Testes de Replicação: este tipo de teste é um dos mais comuns e um dos mais eficientes para garantir a rendabilidade de um sistema, este tipo de teste consiste em “copiar” um componente do sistema, e assim tendo os dois componentes fazer uma votação dos resultados dos mesmos, e a partir disso comparar os resultados obtidos a fim de detectar os erros.
- Testes de Timing: este teste é bastante usado para quando um componente ou um sistema tem restrições quanto ao tempo de resposta, ele simplesmente faz uma requisição ao sistema e cronometra o tempo de resposta, e se o tempo exceder o que foi decidido ele estará em estado de erro.
- Testes Estruturais e Codificação: de acordo com Nélio Pereira (2002), “[...] Em quaisquer dados, dois tipos gerais de testes são possíveis: testes de semântica e estruturais. Testes Semânticos tentam garantir se o valor é consistente com o resto do sistema. Testes Estruturais só consideram a informação e garantem que internamente a estrutura dos dados é como deveria ser. A forma mais comum de teste estrutural é a codificação, que é usada intensamente em hardware. Nela, bits extras são adicionados aos dados, de forma que é possível detectar se existe algum bit corrompido. Tal teste também pode ser usado em software, sendo aplicado às estruturas de dados.”
- Testes de Diagnósticos: são testes executados a partir de dados de entrada no sistema, e com o conhecimento prévio do resultado que o sistema deverá “dar” ao teste, e a partir disso se o resultado não se enquadra com o esperado, é verificado um erro no sistema.

b) Confinamento e avaliação de erros - sistemas não são constantemente monitorados, por isso, até se descobrir que houve uma falha no mesmo, ele pode já ter se propagado no sistema, através da comunicação entre os componentes do sistema. Portanto se faz necessário determinar exatamente o quanto esta falha afetou o sistema, e descobrir quais componentes do mesmo foram afetados por ela. Após descoberto onde ocorreu a

falha, é necessário saber o fluxo que a mesma percorreu dentro do sistema, para assim se determinar o que causou o problema.

c) Recuperação de erros - uma vez que o erro foi descoberto e sua extensão avaliada, é necessário retirar este erro do sistema, e para isso existem duas técnicas, que serão explicadas abaixo.

- Recuperação para Trás (*Backward Recovery*): Esta técnica é uma das mais utilizadas para a recuperação de erros, ela consiste em fazer o sistema voltar para um estado anterior ao qual ocorreu o erro, esperando que este estado esteja livre de erros, e para isso utiliza *checkpoints* periódicos para garantir estes pontos de retorno. Esta técnica se mostrou muito confiável, porém ela consome um certo processamento por parte do sistema, para sempre estar atualizando estes *checkpoints* e também para fazer o *rollback* do sistema exige bastante processamento.
- Recuperação para Frente (*Forward Recovery*): esta técnica consiste em conduzir o sistema para um estado ainda não ocorrido, para tentar assim tornar o estado livre de falhas, porém esta técnica é dificilmente implementada, pois necessita uma avaliação muito precisa dos erros do sistema, para assim saber para qual estado deve levar o sistema. Esta técnica não consegue ser genérica, pois depende muito de regras de negocio de cada aplicação, fazendo assim que dependendo da situação leve o sistema para estados diferentes diante de um mesmo erro.

A tabela abaixo descreve resumidamente o funcionamento destas técnicas:

<b>Técnica</b>	<b>Definição</b>	<b>Características</b>	<b>Implementação</b>
<i>forward error recovery</i>	condução a novo estado ainda não ocorrido desde a última manifestação de erro	eficiente, mas danos devem ser previstos acuradamente	específica a cada sistema
<i>backward error recovery</i>	condução a estado anterior	alto custo, mas de aplicação genérica	pontos de recuperação ( <i>checkpoints</i> ), pistas de auditoria, ...

Tabela 3 – Técnicas de Recuperação

Fonte: Weber, T. Tolerância a falhas: conceitos e exemplos.

d) Tratamento de falhas - mesmo após todos os tratamentos dos erros citados acima, o sistema ainda não está totalmente livre de falhas, pois estes erros podem ser alguma falha de um componente entre outros exemplos.

Portanto é necessário identificar o ponto de falha do sistema para assim poder realmente reparar a falha ocorrida para a mesma não venha a se repetir. Se for uma falha persistente do sistema, deve isolar esta falha para garantir que o sistema não entre mais neste estado faltoso.

#### 2.1.4 Mascaramento de Falhas

Mascaramento de Falhas é a garantia de que o sistema enviará uma resposta certa ao usuário mesmo sofrendo alguma falha no caminho desta repostas, ou seja, a falha causada no sistema não se transforma em um erro, e assim é visto como erro no sistema, fazendo assim com que não se necessite ser tratada esta falha, pois será invisível para o usuário.

Porém em caso de falhas permanentes nos sistemas, de hardware ou de software é necessário ainda assim corrigir este erro.

Segundo (Weber, Tais, 2006), existem alguns mecanismos para o mascaramento de falhas em sistemas, como mostrado na tabela abaixo.

Mecanismo	Aplicado a:
replicação de componentes	qualquer componente de hardware
ECC (código de correção de erros)	informação transmitida ou armazenada
diversidade, programação n-versões	especificação, projetos, programas
blocos de recuperação	software

Tabela 4 – Mecanismos para mascaramento de Falhas  
Fonte: Weber, T. Tolerância a falhas: conceitos e exemplos.

A redundância é a principal técnica e a mais utilizada para garantir que o sistema seja tolerante a falhas. Esta técnica será melhor descrita a seguir.

#### 2.1.5 Redundância

A forma mais utilizada e mais comum para garantir a tolerância a falhas em sistemas é a redundância, segundo (CREVELING, C.J., 1956), "... Redundância para aumento de confiabilidade é quase tão antiga como a história dos computadores.



Quase todas as técnicas utilizadas para garantir que um sistema seja tolerante a falhas usam alguma forma de redundância, tanto que atualmente no mercado, sistemas tolerantes a falhas são chamados de sistemas redundantes.

Existem várias formas de redundância em sistemas, entre elas podemos citar:

- Redundância de Hardware;
- Redundância de Software;
- Redundância Temporal;
- Redundância de informação.

Vale lembrar, que apesar de redundância ser uma das técnicas mais utilizadas para alcançar a dependabilidade, ela sempre tem um custo relacionado, ou seja, para fazer redundância de hardware, por exemplo, sempre se terá um custo elevado para se comprar dois componentes de cada que se deseja fazer a redundância, por isso é importante estudar bem a viabilidade de se utilizar esta técnica em projetos.

Todas estas formas são utilizadas, porém, a redundância de software e hardware são as mais usadas no mercado. Todas estas formas serão melhores especificadas abaixo.

#### *2.1.5.1 Redundância de Hardware*

Esta técnica de redundância é uma das mais utilizada para garantir um sistema tolerante a falhas, ela se baseia na idéia de se adicionar um componente a mais no sistema, que seja idêntico a um já existente, e a partir disto, toda requisição feita no sistema, será atendida pelos dois componentes, e assim, é feita uma votação para verificar a resposta correta.

Em algumas situações esses hardwares redundantes podem ser usados para substituir automaticamente um hardware que venha a ter problemas dentro do sistema.

Três formas de redundância de hardware são estudadas na literatura, elas são: redundância passiva ou estática, redundância ativa ou dinâmica, redundância híbrida.

Abaixo segue uma tabela comparativa entre os tipos de redundância de hardware citadas.

<b>Redundância de hardware</b>	<b>Técnicas</b>	<b>Vantagens</b>
redundância <b>passiva</b> ou <b>estática</b>	mascaramento de falhas	não requer ação do sistema, não indica falha
redundância <b>ativa</b> ou <b>dinâmica</b>	detecção, localização e recuperação	substituição de módulos, usada em aplicações de longa vida
redundância <b>híbrida</b>	combinação de ativa e passiva	usada para garantir mascaramento e longa vida; geralmente de alto custo

Tabela 5 – Redundância de Hardware  
 Fonte: Adaptado por Andrade, Rafael

Estas formas de redundância serão melhores especificadas abaixo:

a) Redundância de Hardware Passiva - esta técnica é usada principalmente para o mascaramento de falhas, ou seja, o sistema não saberá que ocorreu a falha, e dará uma resposta correta o usuário final.

Esta técnica consiste em que todos os elementos redundantes do sistema executem a mesma tarefa requisitada, e enviam suas respostas para um votador central, que verifica todas as respostas enviadas e escolhe a correta para “responder” ao sistema somente a correta. Existem dois exemplos de utilização desta técnica, uma delas é a *TMR(triple Modular Redundancy)* e a *NMR (N-Modular Redundancy)*.

De acordo com (Weber, Taís), “... TMR mascara falhas em um componente de hardware triplicando o componente e votando entre as saídas para determinação do resultado. A votação pode ser por maioria (2 em 1) ou votação por seleção do valor médio.” Apesar de ser uma técnica simples, como tem somente um votador no sistema, faz com que o mesmo se torne um ponto crítico de falha no sistema, pois é ele quem dá a resposta para o sistema, então se ele falhar o sistema apresentará um erro.

A figura a seguir ilustra o funcionamento da técnica TMR.

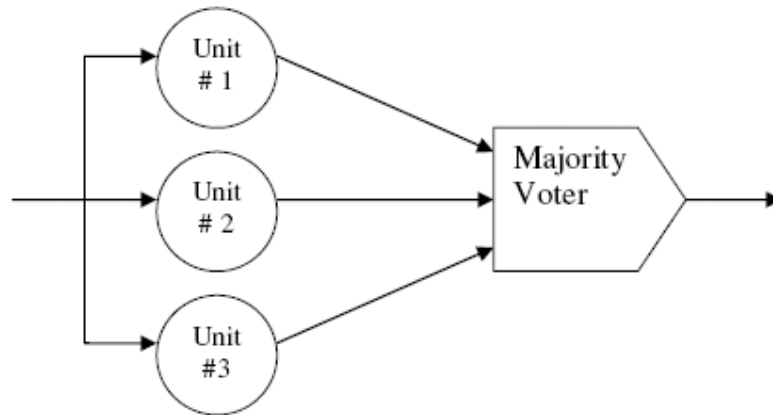


Figura 3 – Exemplo de TMR

Fonte: Pradhan, D. (Editor), Fault-Tolerant Computer System Design.

b) Redundância de Hardware Ativa ou Dinâmica - nesta técnica, a tolerância a falhas é aplicada através da detecção, localização e recuperação de falhas, porém esta técnica não mascara um erro, sendo assim, o caso ocorra uma falha, o sistema saberá e em tempo real a corrigirá.

Redundância dinâmica pode ser usada em sistemas que não são críticos, ou seja, que possam ficar em estado de erro por um certo período de tempo até a correção de falha. Normalmente esta técnica é mais empregada devido ao alto custo de se fazer replicação de todos os componentes que possam ser faltosos no sistema para garantir o mascaramento de falhas.

A figura a seguir mostra o fluxo desta técnica.

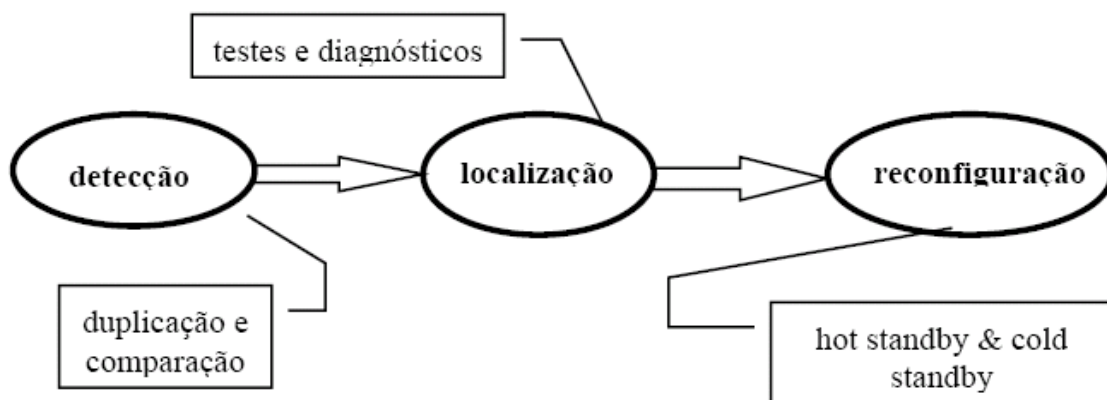


Figura 4 – Redundância Ativa

Fonte: Weber, T. Tolerância a falhas: conceitos e exemplos.

Um exemplo de uso desta técnica é o módulo estepe (*standby sparing*), que pode ser classificados de duas formas, o modo alimentado (*hot standby*), que consiste

em um módulo estepe sempre conectado ao sistema, e o modo não alimentado (*cold standby*), que consiste em o módulo só iniciar seu funcionamento quando conectado.

c) Redundância híbrida - esta técnica combina as vantagens de redundância passiva com a ativa, ou seja, quando o sistema tem uma falha, ela é detectada, localizada e corrigida e também garante com que a resposta do sistema seja correta, garantindo assim que a falha não seja visível.

A seguir a figura ilustra o funcionamento desta técnica.

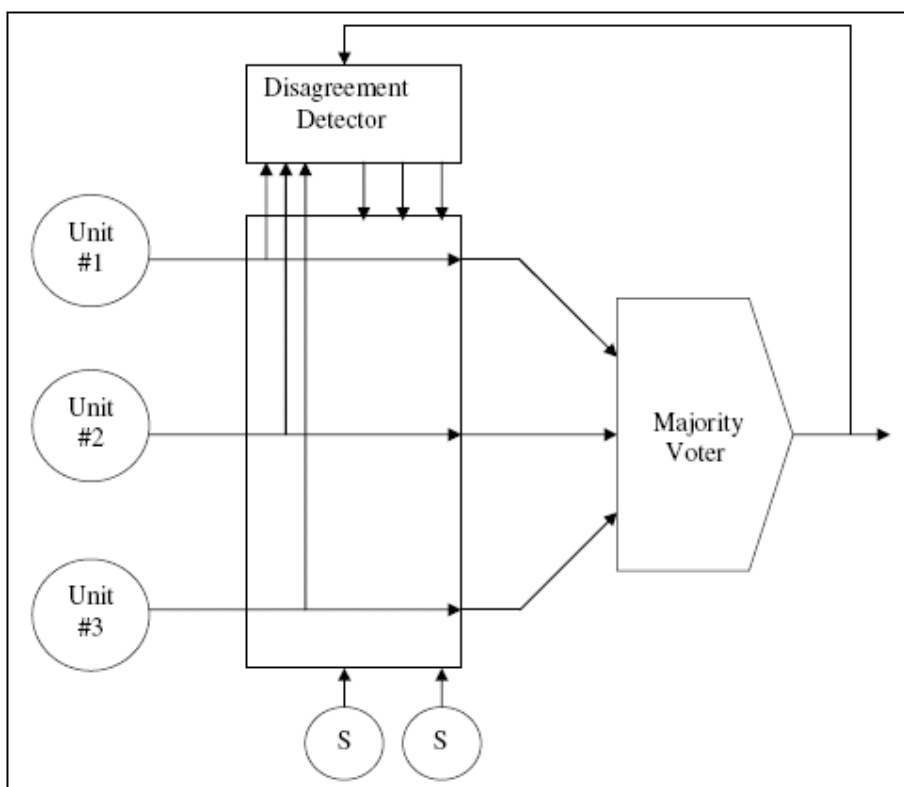


Figura 5: Exemplo de redundância híbrida

Fonte Pradhan, D. (Editor), Fault-Tolerant Computer System Design, Redundância de Software

Esta técnica se refere ao uso de códigos extras, ou até mesmo de sistemas completos, para garantir que a mensagem enviada pelo sistema esteja realmente correta.

Redundância de software está sendo muito utilizada, pois simplesmente replicar hardwares não vai garantir que um software irá funcionar, pois como software, na sua maioria, é independente do hardware, se o mesmo não funcionar em um dos componentes, a chance de não funcionar nos outros também será grande, causando assim um erro ao sistema.

Como disse (Weber, 2008), “... Componentes idênticos de software vão apresentar erros idênticos. Assim não basta copiar um programa e executá-lo em paralelo ou executar o mesmo programa duas vezes em tempos diferentes. Erros de programas idênticos vão se apresentar, com grande probabilidade, de forma idêntica...”.

Falhas em hardware, normalmente são aleatórias, e ocorrem devido a algum problema no componente ou no ambiente em que o mesmo se encontra, já falhas em software normalmente ocorrem devido a problemas de implementação, ou a problemas de especificação do mesmo.

Existem outras formas de garantir a tolerância a falhas em software, pois somente replicar o software pode se mostrar não confiável, entre elas podemos citar, diversidade (ou programação n-versões), blocos de recuperação e *self checking software*.

a) Diversidade - diversidade é uma técnica para alcançar a tolerância a falhas, que inicia com uma especificação de um problema que precisa ser solucionado, e a partir deste, são implementados várias soluções diferentes para o mesmo, ou seja, a partir de uma especificação são criadas várias especificações.

O uso desta técnica requer o uso de um votador também, para a partir das versões diferentes implementadas é feito uma votação dos resultados obtidos. O componente que faz essa votação precisa ser livre de falhas também, ou a confiabilidade obtida por esta técnica se prejudica.

Segundo (Weber, 2002), “... Diversidade pode ser utilizada em todas as fases do desenvolvimento de um programa, desde sua especificação até o teste, dependendo do tipo de erro que se deseja detectar (erro de especificação, de projeto, ou de implementação).” Este tipo de projeto citado é chamado de projeto diversitário, pois utiliza diferentes equipes que desenvolvem o mesmo sistema, porém utilizando metodologias diferentes.

O uso desta técnica não garante totalmente que um sistema estará livre de falhas, pois, de acordo com (Weber, 2002), “... Vários fatores influenciam a eficácia da programação diversitária: as equipes podem trocar algoritmos entre si, os membros das equipes podem, por formação, tender a adotar os mesmos métodos de desenvolvimento, ou as equipes podem buscar suporte nas mesmas fontes...”.

b) Blocos de Recuperação - bloco de recuperação é um método de redundância de software desenvolvida por Randell, esta técnica também usa um votador para decidir se uma resposta do sistema está correta ou não.

O sistema provê várias codificações do mesmo método ou dele inteiro mesmo, da mesma forma do que a técnica de diversidade.

O votador envia uma requisição para o sistema, e espera uma resposta, após isso ele verifica a resposta enviada pelo sistema, e se for decidido que a resposta está errada, ou seja, que o componente teve alguma falha na execução, ele faz um *rollback* do sistema para o estado anterior a execução do método, e a partir disso envia uma nova requisição para o segundo módulo do sistema até obter uma resposta que ele considere correta.

Portanto ao contrário da técnica de Diversidade, o votador não envia a requisição para todos os módulos replicados, e sim envia somente as requisições até conseguir uma mensagem considerada correta.

Este votador precisa ser implementado de forma simples e rápida para não afetar o desempenho do sistema no total.

Esta técnica também se mostra complicada, pois além de precisar fazer a verificação da resposta recebida pelos módulos, o sistema tem que fazer todo o tratamento de checkpoints para poder fazer o *rollback* do sistema em caso de falha.

c) *Self checking software* - este tipo de software é normalmente desenvolvido para sistemas que precisam ter uma disponibilidade extremamente alta, ou seja, sistemas extremamente críticos, que sua falha pode ter conseqüências enormes.

Estes tipos de softwares são sistemas que contém várias checagens extras, normalmente com muitos checkpoints e métodos de *rollback* implementados, e também tem outras execuções dentro deles, as quais ficam buscando erros e corrigindo o sistema em tempo real, para garantir sua integridade.

Apesar de ser um tipo de redundância de software não muito conhecida na literatura, ela se mostra muito eficiente no que se propõe a fazer.

Esta técnica tem sido usada em aplicações críticas, como sistemas para aviões e aparelhos de redes.

### *2.1.5.2 Redundância Temporal*

Redundância temporal consiste em usar um maior tempo para executar operações relacionadas à tolerância a falhas, ou seja, é usada para sistemas que podem ter um tempo de resposta maior. Esta técnica normalmente se baseia em repetir a execução de um método falho, e para isso, utiliza o mesmo hardware e software da execução anterior, ou seja, não é necessário nenhuma redundância de hardware nem de software, e sim somente reexecutar a função faltosa no sistema.

Falhas de tempo ocorrem normalmente devido as complexas interações que os sistemas executam, essa falha tem crescido cada vez mais, pois os sistemas tem se tornado cada vez mais complexos.

Esta técnica também é bastante usada para fazer rollbacks, normalmente técnicas de rollbacks e a união de redundância temporal junto com redundância de hardware ou software.

### *2.1.5.3 Redundância de Informação*

Esta forma de redundância se refere à adição de informação nos dados enviados para poder disponibilizar a detecção de falhas, o mascaramento de falhas ou até mesmo a tolerância a falhas. De acordo com (Keitel, 1999) "... Um exemplo deste tipo de redundância é incorporar códigos para detecção de erros e códigos para correção de erros que geralmente são usados em circuitos lógicos".

Outro exemplo clássico de redundância de informação são os códigos de paridade, que consiste em que para cada numero de bits  $n$ , será armazenado  $n+1$  bits. Este bit a mais apenas mostra se o número  $n$  de bits anteriores a ele é par, ou impar de acordo com o tipo de paridade usada. Esta técnica é usada para detecção de falhas simples, que ocorrem em somente um bit dos dados.

## 2.2 BANCO DE DADOS

A quantidade de informação necessária hoje em dia para qualquer empresa no mercado, fez com que as mesmas buscassem outras formas de armazenar a informação, e com isso, uma das principais idéias lançadas foram os bancos de dados, que são conjuntos de registros organizados, a fim de retirar informação dos mesmos.

Bancos de dados já estão incorporados no dia a dia de todas as pessoas, pois tudo que envolve algum sistema, tudo que precisa de algum componente computacional normalmente tem um banco de dados por trás do mesmo.

Banco de dados foi criado no ano de 1955, e a partir disso seu uso só foi aumentado, e como dito, quase tudo que fazemos hoje em dia envolve sistemas, e sistemas quase sempre tem banco de dados.

Para controlar e manipular os banco de dados existem sistemas próprios, os chamados Sistemas de Gerenciamento de Banco de Dados (SGBD), de acordo com (Korth & Silliberchatz), SGBDs são "... Módulo de programa que fornece a interface entre os dados de baixo nível armazenados num banco de dados e os aplicativos ou as solicitações submetidas aos sistemas.

SGBDs são sistemas para controle e gerenciamento de banco de dados, eles disponibilizam interface para incluir, alterar e consultar dados, com ele, o controle do banco de dados fica desacoplado do servidor, facilitando o uso do mesmo.

O tipo mais comum de banco de dados usado no mercado atualmente é o banco de dados relacional, que será melhor explicado a seguir.

### **2.2.1 Banco de dados Relacional**

Banco de dados relacional é uma forma de representar dados do mundo real em forma de tabelas, e através dos relacionamentos destas tabelas, mostrar os relacionamentos possíveis que as mesmas possuem.

Este tipo de banco de dados é baseado no Modelo relacional, e por isso seus modelos são representados de forma matemática e através de relações entre estes objetos.

As relações neste tipo de banco são definidas como "um modelo formado por relações (no sentido matemático) entre os domínios. Cada tupla é um elemento do conjunto relação".

O banco de dados relacional é baseado em alguns conceitos, como tabelas, registros, chaves, colunas entre outros. Abaixo serão explicados de forma melhor cada um destes.



### *2.2.1.1 Entidades e atributos*

Toda a Informação de um banco de dados relacional é armazenada em Tabelas, que na linguagem do modelo relaciona, também são chamadas de Entidades.

Uma tabela é uma simples estrutura de linhas e colunas. Em uma tabela, cada linha contém um mesmo conjunto de colunas. Em um banco de dados podem existir uma ou centenas de tabelas, sendo que o limite pode ser imposto tanto pela ferramenta de software utilizada, quanto pelos recursos de hardware disponíveis no equipamento.

As tabelas associam-se entre si através de regras de relacionamentos, estas regras consistem em associar um ou vários atributo de uma tabela com um ou vários atributos de outra tabela.

Estas tabelas possuem atributos (colunas) que são o que determina as características dos elementos que são desta tabela.

### *2.2.1.2 Chave Primária*

Chave primaria é a coluna de uma tabela, ou seja, um atributo desta tabela, que identifica as entidades desta tabela. Normalmente nesta coluna são usados números ou identificadores únicos possíveis, ou seja, identificadores que nunca se repetirão.

Quando é definida uma chave primaria para uma tabela, estamos dizendo ao banco de dados que nunca poderá haver dois objetos com o mesmo valor para este atributo.

Após ter definido um campo como sendo a Chave Primária da tabela, o próprio banco de dados, garante que não sejam inseridos dados duplicados no campo que é a chave primária. Por exemplo, se o usuário tentar cadastrar um pedido com o mesmo número de um pedido já existente, o registro não será cadastrado e uma mensagem de erro será emitida.

### *2.2.1.3 Relacionamentos*

Um relacionamento em banco de dados relacional é a associação entre entidades distintas, o relacionamento consiste em “a ligação de um campo de uma tabela X com um campo de uma tabela Y, por meio da inclusão do campo chave da tabela Y como um campo (conhecido como chave estrangeira) da tabela X.”

Conforme citado acima, chave estrangeira é a representação de uma coluna de uma tabela externa na tabela desejada, criando assim um relacionamento entre as duas tabelas.

### 2.3 BANDO DE DADOS DISTRIBUÍDOS

Defini-se de acordo com Bell e Grimson, 1992, “sistema de banco de dados distribuído (SBDD) como uma coleção de dados inter-relacionados que se encontram fisicamente distribuídos pelos nós de uma rede de computadores”.

Um sistema gerenciador de banco de dados distribuído (SGBDD) é um software que gerencia um banco de dados distribuído de tal modo que os aspectos da distribuição ficam transparentes para o usuário. Assim, o usuário tem a ilusão de integração lógica de dados, sem requerer integração física do banco de dados. O usuário de um sistema de banco de dados distribuído é incapaz de saber a origem das informações, tendo impressão de estar acessando um único banco de dados.

Enquanto um sistema gerenciador de banco de dados centralizado (SGDB) gerencia um único banco de dados, o SGBDD é responsável pelo gerenciamento global e local.

Cada nó de um banco de dados distribuído é capaz de processar transações locais, as quais acessam apenas dados daquele único nó, ou pode participar na execução de transações globais, que fazem acesso a dados em diversos nós. A execução de transações globais requer comunicações entre os nós, que podem ser feitos através de cabos, fibras óticas, linhas telefônicas, ligações de microondas, canais de satélite, etc.

Se o projeto de um sistema distribuído é executado top-down, isso é, sem um sistema já existente, é conveniente desenvolver um sistema homogêneo. Todavia, em alguns casos onde a criação de banco de dados distribuídos for feita pela integração de vários bancos de dados já existentes (chamados bottom-up), será necessário um SGBDD heterogêneo, capaz de fornecer interoperabilidade entre os bancos de dados locais.

### **2.3.1 Porque usar Sistemas de Banco de Dados Distribuído?**

Existem diversas razões para construir um banco de dados distribuído, como o compartilhamento de dados, confiabilidade, disponibilidade e aceleração de processamento de consultas. Entretanto, juntamente com essas vantagens há diversas desvantagens, como o custo de desenvolvimento do software, maior potencial para existência de erros e crescente sobrecarga de processamento.

A principal vantagem de sistemas de bancos de dados distribuídos é a capacidade de dividir e fazer acesso a dados de uma maneira confiável e eficiente. Pois, se uma série de nós diferentes estão conectados, então um usuário em um nó pode ser capaz de fazer acesso a dados disponíveis em um outro nó. Cada nó é capaz de reter um grau de controle sobre dados armazenados localmente.

Caso ocorra uma falha em um dos nós do sistema distribuído, os nós remanescentes podem ser capazes de continuar operando, aumentando a confiabilidade e disponibilidade. Além disso, quando uma consulta envolve dados em diversos nós, é possível dividi-la em subconsultas que podem ser executadas em paralelo, acelerando seu processamento.

### **2.3.2 Características de um sistema de Banco de dados Distribuído**

A seguir são descritas algumas características do SBDD, que fazem com que estes sistemas proporcionem inúmeras vantagens a seus usuários.

#### *2.3.2.1 Gerenciamento transparente de dados distribuídos e replicados*

Transparência é a separação da semântica de alto nível de um sistema dos detalhes da implementação. Um sistema transparente esconde detalhes de implementação de usuários. Podemos citar vários tipos de transparência em banco de dados como, por exemplo: transparência de distribuição, de replicação e de fragmentação dos dados.

Em um SBDD, os programas de aplicação não sabem e não precisam saber onde os dados estão localizados. O SGBDD encontra os dados, onde quer que eles estejam, sem o envolvimento do programa de aplicação.

Transparência de replicação refere-se ao fato de que os programas de aplicações não sabem se os dados estão replicados ou não. Se estiverem, o SGBDD assegurará que todas as cópias sejam atualizadas consistentemente, sem o envolvimento do programa de aplicação.

#### *2.3.2.2 Independência de dados*

Independência dos dados, de acordo com Ozsu e Valdureiez, 1999, "... é uma forma de transparência fundamental que se procura em um SGBD." Refere-se à impossibilidade das aplicações dos usuários em alterar a definição e organização dos dados e vice-versa. Ou seja, os dados são armazenados de forma independente dos programas que o utilizam.

#### *2.3.2.3 Aumento do desempenho (performance)*

O desempenho aumenta em SGBDD devido principalmente a dois fatores. Primeiro, o SGBDD fragmenta o esquema conceitual de modo que os dados são armazenados próximos aos pontos nos quais são mais utilizados. Assim, cada local possui apenas uma parte do banco de dados, minimizando a sobrecarga de processamento. Um segundo fator é o paralelismo. Quando uma consulta envolve dados em diversos nós, é possível dividi-la em subconsultas que podem ser executadas em paralelo, acelerando seu processamento.

#### *2.3.2.4 Confiabilidade e disponibilidade através de transações distribuídas*

Uma característica importante de sistema de banco de dados distribuídos é a capacidade de dividir e prover acesso a dados de uma maneira confiável e eficiente.

Uma falha em um dos nós, ou em um link de comunicação que faça com que um ou mais sites fiquem inalcançáveis, não prejudicará o sistema como um todo. Os nós remanescentes podem ser capazes de continuar operando, aumentando a confiabilidade e disponibilidade.

### 2.3.3 Complicações de sistemas em banco de dados distribuídos

A complexidade em sistemas distribuídos aumenta a devido a vários fatores. Um deles refere-se à distribuição dos dados. Não é essencial que cada site da rede possua o banco de dados completo e sim que um banco de dados resida em mais de um site. Portanto, é necessário definir como será a distribuição e replicação (ou não) dos dados.

Outro fator refere-se a falhas (de hardware ou software) nos sites ou na rede de comunicação que possam vir a ocorrer na execução de uma atualização de dados. O sistema deve ter certeza de que os efeitos serão refletidos nos dados residentes nestes sites indisponíveis assim que o sistema retornar ao normal.

A maior desvantagem do sistema de banco de dados distribuído é a complexidade adicional requerida para assegurar a própria coordenação entre os nós. Devido a esta complexidade é requerido hardware e software adicionais, o que leva a um aumento de custos de desenvolvimento, potencialidade de defeitos e da sobrecarga de processamento. A tabela a seguir resume os principais complicadores em SGBD distribuídos.

Projeto	- como distribuir o banco de dados - distribuição dos dados replicados
Processamento de consultas	- conversão de transações de usuários em instruções de dados - problema de otimização
Controle de concorrência	- sincronização de acessos concorrentes - consistência e isolamento de efeitos de transações - gerenciamento de deadlocks
Confiabilidade	- como manter o sistema imune à falhas - atomicidade e durabilidade

Tabela 6: Fatores complicadores de SGBD distribuído.

Fonte: Adaptado por Andrade, Rafael

### 2.3.4 Arquitetura de SGBD Distribuídos

Uma arquitetura define a estrutura de um sistema: identificação, definição da função e inter-relacionamento e interações entre os componentes do sistema. Na arquitetura data-lógica é formada pelo esquema interno local de cada site, o esquema conceitual local de cada site, e o esquema conceitual global e esquema externos.

O esquema interno local (EIL) refere-se aos aspectos relacionados ao armazenamento físico dos dados do site. O esquema conceitual local (ECL) refere-se à

estrutura lógica (informações) do banco de dados. O esquema conceitual global (ECG) é formado pela união dos esquemas conceituais locais, permitindo uma visão global dos dados. Finalmente o nível mais externo, os esquemas externos (EE) são as visões definidas aos usuários globais.

Esta arquitetura é denominada ponto-a-ponto (peer-to-peer) devido ao fato de que cada site possui SGBD completo, diferente da arquitetura cliente servidor que concentra o gerenciamento dos dados em servidores, enquanto nos clientes ficam as interfaces e aplicações.

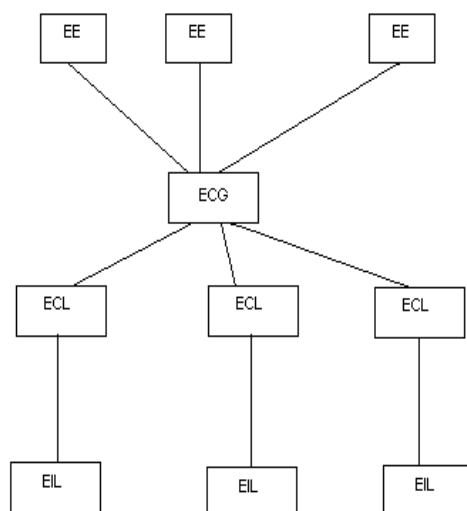


Figura 6: Arquitetura data-lógica de SGBD distribuído.  
Fonte: MANICA, Heloisa. Banco de dados distribuídos heterogeneos: Arquiteturas, tecnologias e tendências.

Quando um projeto de banco de dados distribuídos é realizado a partir de base de dados já existentes, é chamado de bottom-up. Deste modo, surge uma nova arquitetura data-lógica de multi-SGBD. A maior diferença entre esta arquitetura e a data-lógica é a forma de mapeamento entre esquemas.

## 2.4 APLICAÇÕES WEB

Aplicações web são as aplicações desenvolvidas para serem usadas nos browsers dos usuários, sem a necessidade de nenhum outro “programa” específico, somente um

acesso a internet, e devido a esse fato este tipo de sistema vem se popularizando cada vez mais.

Atualmente a maioria das soluções usadas pelas empresas tende a ser web, devido ao fato de serem aplicações centralizadas em um único servidor, e fazendo com que qualquer usuário tenha acesso a ele facilmente através de uma URI, que consiste no endereço onde fica localizada a aplicação na web.

Estas aplicações tem se popularizado mais na ultima década, devido à melhoria na infra-estrutura das redes e da web em si, permitindo altas velocidades de acesso para sistemas, fazendo assim com que se pudessem criar aplicações cada vez mais robustas e mais completas.

Porém as aplicações web estão atualmente sofrendo grandes alterações, pois além de completas e robustas, os usuários estão cobrando que as mesmas sejam cada vez mais intuitivas e fáceis de usar, e isso tem se tornado um ponto vital para aplicações web.

E nesta tendência estamos vivendo a web 2.0, que se baseia completamente na idéia de que os sistemas devem ter uma alta usabilidade, ou seja, devem ser fáceis de usar, e sempre intuitivos para os usuários, além de serem cada vez mais “bonitos”, aonde as novas tecnologias tem se mostrado muito eficiente.

Um das tecnologias atuais mais conhecidas para a web 2.0 é o Flex, uma linguagem de programação desenvolvida pela Adobe, que é baseada em Action Script para programar a parte de interface com o usuário. O Flex forma arquivos swf (flash) para gerar a interação com o usuário final, disponibilizando assim uma melhor interface para o usuário.

#### **2.4.1 Tolerância a falhas em aplicações web 2.0**

A maioria das aplicações web 2.0 não tem suporte a alguma técnica de tolerância a falhas, tornando assim seus serviços vulneráveis a qualquer falha, de hardware ou software.

O que torna suas aplicações não confiáveis, pois caso algum servidor falhe a aplicação irá “travar” para o usuário.

Este trabalho propõe uma arquitetura para aplicações web 2.0 que garantirá a tolerância a falhas de hardware, e também possibilitando tolerâncias a falhas de software, conforme será melhor especificado abaixo.

A arquitetura proposta se baseia no servidor de aplicação Jboss juntamente com Banco de dados Mysql. Ela se propõe a criar clusters de servidores de aplicação juntamente com clusters de banco de dados, garantindo assim que caso algum dos servidores caia a aplicação não “trave” para o usuário.

O servidor de aplicações Jboss se trata do servidor de aplicação mais usado para aplicações web Java no mercado, está no mercado há muitos anos e é muito estável. Ele oferece suporte para clusterização, cache entre outras funcionalidades que o garantem como um dos melhores servidores.

#### **2.4.2 Clusterização de Servidores de Aplicação(Jboss)**

Clusterização é a possibilidade de poder rodar aplicações em diversos servidores, com esta forma, uma aplicação poderá continuar rodando mesmo em caso de falha de algum dos servidores disponíveis dentro do cluster.

Para fazer clusterização com o Jboss, a forma mais simples é disponibilizar vários servidores dentro de uma mesma rede, e rodá-los com a configuração `-c all`. Assim todos iniciam dentro de um cluster desta rede.

Através da clusterização é possível garantir que aplicações tenham uma disponibilidade muito maior na rede, pois ela garante que mesmo com a falha de algum servidor, a aplicação continuará rodando transparentemente para o usuário final.

A clusterização do Jboss utiliza o framework Jgroups para fazer o controle do cluster, tendo sempre uma máquina considerada a “chefe” e as outras enviando requisições a ela.

Jgroups é um framework para comunicação multicast confiável, ou seja, é um framework que trabalha com grupos de máquinas trabalhando em conjunto e contém funções de criação, deleção de grupos, assim como entrar e sair dos mesmos, e também envia avisos a todos os componentes do grupo sobre aviso ou entrada de um novo componente neste grupo.

Este framework garante que as mensagens trocadas entre os servidores do cluster realmente sejam entregues entre eles, pois em caso de falha existem formas de garantir o envio, como reenvio ou então confirmação de recebimento do mesmo.

A arquitetura utilizada será melhor especificada no tópico de desenvolvimento do sistema.



### 3 DESENVOLVIMENTO DO SISTEMA

O sistema a ser desenvolvido será um sistema web desenvolvido para a empresa Direseg Seguros, que se trata de uma corretora de seguros. O sistema terá usuários finais, portanto se trata de uma aplicação real, e não somente aplicação de testes.

Ele será desenvolvido totalmente voltado a web 2.0, usando as ultimas tecnologia para a interface com seus usuário, como Flex, uma linguagem de programação desenvolvida pela Adobe, aonde é usado Action Script para gerar sistemas totalmente em flash.

O sistema também terá técnicas para tolerância a falhas, principalmente de *hardware*, as técnicas utilizadas para esta aplicação foram o clusterização de servidores de aplicação, usando servidores de aplicação Jboss 4.0.2, e também técnicas de replicação de banco de dados, usando banco de dados Mysql.

As técnicas utilizadas no desenvolvimento do sistema serão melhor explicadas nos tópicos abaixo.

#### 3.1 REGRAS DE NEGÓCIO DO SISTEMA

O sistema a ser desenvolvido será um sistema Web com garantias de Tolerância a falhas, principalmente, falhas de hardware.

O sistema será desenvolvido em Java, utilizando servidores de aplicação Jboss, versão 4.2.3, e utilizará banco de dados Mysql. As tecnologias usadas no desenvolvimento dos mesmos são as seguintes:

- Mentawai - Framework para controle de navegação e comunicação com a tecnologia Flex
- Spring - Framework consolidado no mercado, que contém todos controle de sessões, injeção de dependência, entre outros.
- Hibernate - Framework usado para comunicação com banco de dados, muito consolidado no mercado também.
- Flex - Tecnologia desenvolvida pela Adobe para aplicações RIA (Rich Internet Application), que são aplicação avançadas desenvolvidas para Web, voltada a web 2.0

A aplicação será desenvolvida completamente voltada à web 2.0, utilizando flex como camada de visão, que torna a interação com sistema mais fácil e mais bonita para o usuário final. Ela também terá um usuário final real, que necessita alta disponibilidade na aplicação.

O sistema será desenvolvido para uma corretora de seguros, e se trata de um sistema para administração dos segurados que a mesma possui, como também servirá para pedidos e cotações de novos seguros.

Ele será separado em três módulos inicialmente, o primeiro para os administradores da corretora, aonde poderão ver todos seus clientes, assim como os seguros de cada cliente, poderão ver também os pedidos de cotações novos, assim como relatórios completos de produção de seus produtores entre outros.

O segundo módulo será para revendedoras de automóveis, que, a cada venda de um novo veículo, enviará um pedido de cotação de seguros para o sistema, e assim os administradores da corretora poderão calcular e enviar a resposta em tempo real para a revendedora, facilitando assim o trabalho de ambos.

O terceiro módulo a ser desenvolvido será um módulo para administradoras de condomínio, aonde se formará uma parceria com as mesmas, e elas também farão pedidos de cotações de seguros e terão as respostas em tempo real. As regras de negócio do sistema estão melhor especificadas no **ANEXO A**.

O sistema deverá contar com garantias de tolerância a falhas, principalmente falhas de hardware, e para isso foram utilizadas técnicas de clusterização de servidores de aplicação e também técnicas de replicação de banco de dados. Todas estas técnicas serão melhor especificadas posteriormente.

O desenvolvimento do sistema foi feito usando técnicas de desenvolvimento rápido, chamado SCRUM, que se baseia na idéia de entregas constantes para o usuário final, para que o mesmo consiga visualizar o desenvolvimento do sistema. As entregas do sistema estão sendo feitas mensalmente, para garantir que o usuário final consiga ver a evolução do sistema a cada mês.

### 3.2 BANCO DE DADOS MYSQL

O banco de dados Mysql é atualmente o banco de dados open source mais usado no mercado, demonstrando grande maturidade e também um ótimo desempenho.

Foi escolhido utilizar este banco devido ao fato de ser o único banco de dados open source com características de replicação e clusterização de dados nativo, tendo em vista que os outros bancos estudados também têm essa possibilidade porém são soluções desenvolvidas por terceiros, sem ter a garantia de funcionamento das mesmas.

Para instalação do banco de dados foram usadas configurações padrões de instalação. A partir disso foram feitas as configurações do cluster de banco de dados conforme mostrado abaixo.

### 3.3 CLUSTERIZAÇÃO DE MYSQL

MySql Cluster é um banco de dados de alta disponibilidade construído utilizando uma arquitetura única e interface SQL padrão. O sistema consiste de um número de processos de comunicação, ou nós que podem ser distribuídas através de máquinas para garantir a disponibilidade contínua em caso de falha de servidor ou rede. Mysql Cluster usa um mecanismo de armazenagem, que consiste em um conjunto de dados nós para armazenamento de dados que pode ser acessado através do padrão SQL com o Mysql ou através do NDB API para acesso em tempo real.

A API NDB é uma linguagem de programação para aplicações orientada a objetos para o Cluster Mysql que implementa indexs, busca, transações e tratamentos de eventos.

O cluster de Mysql garante a tolerância a falhas em qualquer nó do cluster, para isso, ele se auto reconfigura para garantir seu funcionamento.

### 3.4 SERVIDOR DE APLICAÇÕES JBOSS

O servidor de aplicações Jboss é atualmente considerado o mais robusto e de melhor qualidade no mercado, e uma das suas maiores características é o fato de o mesmo ser open source, ou seja, ele é aberto à comunidade. O Jboss é um servidor

certificado pela SUN, garantindo assim sua compatibilidade com as aplicações Java mais utilizadas no mercado.

### **3.4.1 Clusterização com Jboss**

Clusterização é a possibilidade de poder rodar aplicações em diversos servidores, com esta forma, uma aplicação poderá continuar rodando mesmo em caso de falha de algum dos servidores disponíveis dentro do cluster.

Para fazer clusterização com o Jboss, a forma mais simples é disponibilizar vários servidores dentro de uma mesma rede, e rodá-los com a configuração `-c all`. Assim todos iniciam dentro de um cluster desta rede.

Através da clusterização é possível garantir que aplicações tenham uma disponibilidade muito maior na rede, pois ela garante que mesmo com a falha de algum servidor, a aplicação continuará rodando transparentemente para o usuário final.

### **3.4.2 Configurando o cluster e os nodes**

Cada instância de um servidor Jboss que está em uma máquina, será considerado como node, ou seja, uma parte do cluster. Este node deve especificar em qual cluster ele está. Para configurar isto é necessário configurar o arquivo `cluster-service.xml` do node. Para que os nodes pertençam ao mesmo cluster, é necessário que todos possuam a mesma configuração no `cluster-service.xml`. Abaixo está um exemplo de como ficaria a configuração padrão para configurar um cluster.

```

<mbean code="org.jboss.ha.framework.server.ClusterPartition"
  name="jboss:service=DefaultPartition">

  <! -- Name of the partition being built -->
  <attribute name="PartitionName">
    ${jboss.partition.name:DefaultPartition}
  </attribute>

  <! -- The address used to determine the node name -->
  <attribute name="NodeAddress">${jboss.bind.address}</attribute>

  <! -- Determine if deadlock detection is enabled -->
  <attribute name="DeadlockDetection">False</attribute>

  <! -- Max time (in ms) to wait for state transfer to complete.
    Increase for large states -->
  <attribute name="StateTransferTimeout">30000</attribute>

  <! -- The JGroups protocol configuration -->
  <attribute name="PartitionConfig">
    ... ..
  </attribute>
</mbean>

```

Figura 7: Exemplo de configuração de um node.  
 Fonte: Brian Stansberry, 200?

Para formar um cluster, é necessário que todos os nodes estejam com exatamente o mesmo `PartitionName` e `PartitionConfig`, pois eles configuram a localização e nome do cluster.

Para fazer a parte de comunicação entre os nodes, o Jboss usa um framework conhecido chamado de Jgroups, este framework será melhor explicado abaixo.

### 3.4.2.1 Jgroups

O Jboss usa o Jgroups para fazer a comunicação ponto-a-ponto entre os nodes do seu cluster, com ele cada node do cluster pode enviar os mensagens para os outros e descobrir se houve falha entre outras coisas.

Para configurar o Jgroups dentro do Jboss, normalmente é feito dentro do `PartitionConfig` dos nodes, e lá é configurável o tipo de transporte da comunicação, o tempo para timeout de uma requisição entre outras coisas. A figura abaixo ilustra um exemplo de um node com Jgroups configurado usando UDP multicast para fazer a comunicação entre ele e os outros nodes.

```

<mbean code="org.jboss.ha.framework.server.ClusterPartition"
  name="jboss:service=DefaultPartition">
  ... ..
  <attribute name="PartitionConfig">
    <Config>
      <UDP mcast_addr="228.1.2.3" mcast_port="45566"
        ip_ttl="8" ip_mcast="true"
        mcast_send_buf_size="800000" mcast_rcv_buf_size="150000"
        ucast_send_buf_size="800000" ucast_rcv_buf_size="150000"
        loopback="false"/>
      <PING timeout="2000" num_initial_members="3"
        up_thread="true" down_thread="true"/>
      <MERGE2 min_interval="10000" max_interval="20000"/>
      <FD shun="true" up_thread="true" down_thread="true"
        timeout="2500" max_tries="5"/>
      <VERIFY_SUSPECT timeout="3000" num_msgs="3"
        up_thread="true" down_thread="true"/>
      <pbcast.NAKACK gc_lag="50"
        retransmit_timeout="300,600,1200,2400,4800"
        max_xmit_size="8192"
        up_thread="true" down_thread="true"/>
      <UNICAST timeout="300,600,1200,2400,4800"
        window_size="100" min_threshold="10"
        down_thread="true"/>
      <pbcast.STABLE desired_avg_gossip="20000"
        up_thread="true" down_thread="true"/>
      <FRAG frag_size="8192"
        down_thread="true" up_thread="true"/>
      <pbcast.GMS join_timeout="5000" join_retry_timeout="2000"
        shun="true" print_local_addr="true"/>
      <pbcast.STATE_TRANSFER up_thread="true" down_thread="true"/>
    </Config>
  </attribute>
</mbean>

```

Figura 8: Configurando o JGroups  
 Fonte: Brian Stansberry, 200?

O Jgroups possui vários protocolos possíveis para fazer a comunicação com seu cluster, entre elas podemos citar UDP, TCP, e TUNNEL. É possível usar somente um deste protocolo em cada node do cluster.

Para poder fazer a comunicação entre os nodes, o cluster precisa sempre manter atualizado sua lista com todos os nodes que estão nele, para que o load balancer saiba qual node está com falha e qual está ativo. Para fazer essa comunicação se usa os protocolos de descobrimento. Esses protocolos são baseados no protocolo escolhido para fazer a comunicação.

Esses protocolos podem ser dos seguintes tipos:

PING - usado pelo transporte UDP, ele consiste em o node mestre enviar uma requisição para todos nodes do servidor, e os mesmo responde com um pacote UDP para ele.

TCPPING – usado pelo transporte TCP, ele consiste em uma lista de IP's conhecida, e assim o node envia uma requisição para todos os ips cadastrados.

MPING – usado pelo transporte TCP, funciona exatamente como o PING do UDP, ele envia uma mensagem para todos os nodes que estiverem na configuração.

A tabela baixo mostrará cada protocolo e que tipo de transporte utilizar para ele.

Protocolo de Descobrimto	Protocolo de Transporte
PING	UDP
TCPPING	TCP
MPING	TCP

Tabela 7: Protocolos de descobrimto x protocolos de Transporte  
Fonte: Adaptado por ANDRADE, Rafael

Quando um node sofre alguma falha e fica indisponível, é necessário informar o load balance para que o mesmo não tente se conectar com ele, e para isso, é utilizado os protocolos de detecção de falhas. Existem três tipos deste protocolo citado. A primeira dela é o FD, que consiste em um node do cluster enviar sempre uma mensagem pergunta se o vizinho do mesmo está “vivo”, ou seja, está disponível, se o node vizinho não responde, o node que enviou a mensagem envia uma mensagem de SUSPECT para o cluster. Quando o coordenador do grupo recebe esta mensagem, ele teste novamente se o node está indisponível e atualiza a visão do cluster, retirando o que está fora.

Outra forma de detectar falhas é o FD\_SOCKET, ele é usado para enviar sempre mensagens perguntando se o vizinho está vivo pode aumentar muito o trafego de informação dentro da rede, por exemplo, se o servidor está muito carregado e o timeout do FD é muito baixo, ele irá enviar falsas mensagens de SUSPECT, pois é somente uma demora na comunicação. O FD\_SOCKET faz somente uma conexão TCP no node ao lado, e somente se não for possível tão conexão é que o mesmo envia a mensagem SUSPECT pra o cluster. Este protocolo é mais usado para redes que tem muita comunicação aonde todos os nodes são acessados freqüentemente.

Existe também o FD\_SIMPLE, que é um protocolo que também envia mensagens de “vc-está-vivo” para um node qualquer do cluster, se o node não responder a mensagem enviada, um contador com o numero de falhas de cada node é incrementado, quando o contador chega ao limite, é enviado uma mensagem de SUSPECT para o cluster.

### **3.4.3 Arquiteturas de serviço**

A arquitetura do serviço de cluster é muito importante para o administrador do sistema. Mas os desenvolvedores estão mais preocupados com a arquitetura para o cliente. O Jboss suporta dois tipos de arquitetura para clusters, o client-side-interceptos (também chamados de proxies ou stubs), e também os load balancers.

#### *3.4.3.1 Client-Side Interceptors*

A maioria dos serviços do Jboss é feito através do cliente que obtém um stub (ou Proxy) que é implementado pelo servidor e enviado para o cliente. Assim o cliente envia as requisições para esse stub gerado e assim esse stub redireciona para o servidor de acordo com as necessidades.

Este objeto tem o conhecimento sobre o cluster, como o IP de todos nodes do cluster, o algoritmo para fazer o balanceamento entre os nodes, e também como fazer em caso de falha em uma requisição. A imagem abaixo mostra como fica a arquitetura utilizando client-side interceptors.



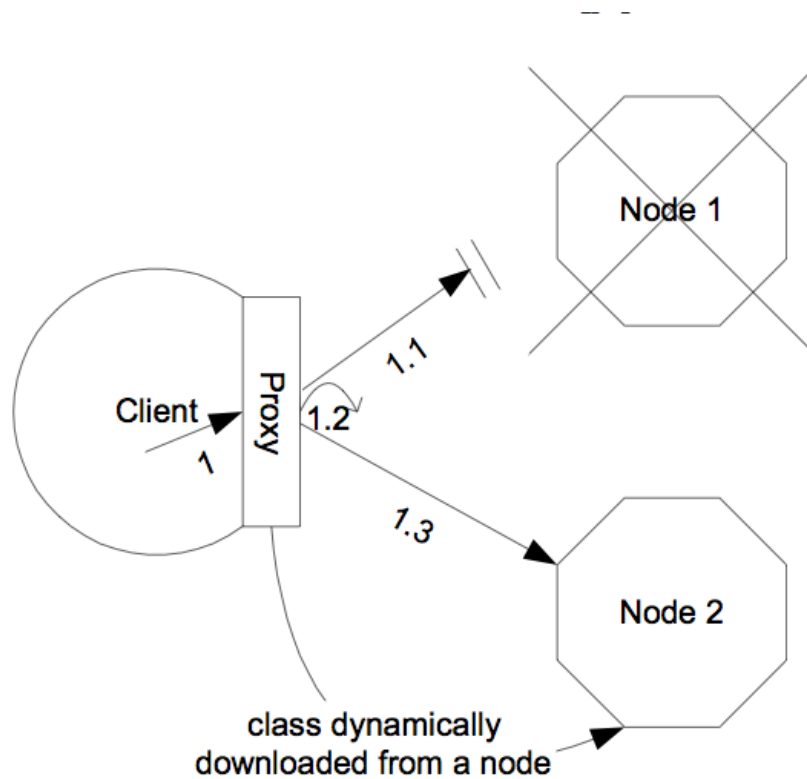


Figura 9: arquitetura de client-side Interceptor.  
 Fonte: Adaptado por ANDRADE, Rafael

### 3.4.3.2 Load Balancer

Alguns serviços do Jboss, como serviços HTTP, não requerem um cliente que faça o download de nada. Por exemplo, o cliente de uma aplicação web seria um Web browser, que envia uma requisição e a recebe diretamente do servidor. Para estes casos é usado o load balancer, ele processa todas as requisições enviadas e as encaminha para o node do cluster de acordo. Normalmente o load balancer é parte do cluster e o cliente precisa conhecer somente o load balancer. A figura abaixo ilustra esta arquitetura.

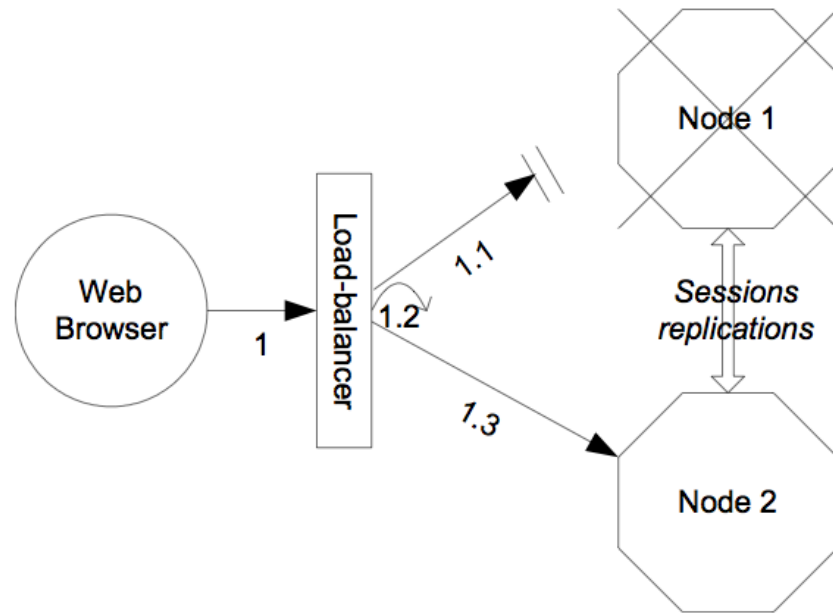


Figura 10: Arquitetura do load-balancer  
 Fonte: Adaptado por ANDRADE, Rafael

### 3.4.4 Políticas de Balanceamento

Em ambas as arquiteturas citadas, é usado algum tipo de política de balanceamento, que é feita para determinar para qual node do cluster envia uma requisição quando houver falha.

Atualmente existem três formas para o balanceamento de requisições, elas são o Round - Robin, o First Available e o First AvailableIdenticalAllProxies, eles serão melhor explicados abaixo.

Round - Robin – cada chamada para o servidor é enviada para um node novo. O node é selecionado aleatoriamente da lista.

First Available – um dos nodes disponíveis é selecionado como o principal e usado para receber todas as requisições enviadas. Quando houver alguma mudança na lista de nodes disponíveis, um novo node principal será eleito.

First AvailableIdenticalAllProxies – é semelhante ao First Available, porém para selecionar um novo principal, este é o mesmo para todos clientes da mesma “família”.

Estas políticas de balanceamentos são muito importantes para decidir qual a necessidade do sistema sendo desenvolvido. Cada uma delas tem sua forma

especifica de funcionamento e cada uma se adequa melhor de acordo com a necessidade.

### 3.4.5 Replicações de Sessões e Load Balance para aplicações Web

Replicação de sessões é usado para replicar, ou seja, copiar todos os dados de uma sessão de um node do Jboss para os outros. Isso é importante, pois se um dos nodes tem algum problema, e fica indisponível, outro node vão ser capazes de recuperar esta sessão. Para isso é necessário ter duas funções no servidor, o session state replication e Load-Balance das requisições que vão chegar.

Session state replication é feito diretamente pelo Jboss, ele é executado de forma default quando rodado na configuração all. Mas para rodá-lo, é necessário configurar no arquivo web.xml da sua aplicação e colocar a tag <distributed />.

Já para o load-balance é necessário usar outros programas para fazerem, o jeito mais comum de fazer isso é utilizando o servidor Apache junto com mod\_jk, um módulo para o mesmo.

Mod\_jk foi desenvolvido especialmente para que o servidor Apache consiga encaminhar requisições para um servlet, como o Jboss, ele também consegue fazer o load-balance das requisições HTTP que chegam ao servidor.

#### 3.4.5.1 Configurando Apache o Mod\_jk

Para aplicar load-balance no servidor, é necessário instalar o servidor web Apache, e também o seu módulo mod\_jk.

Para configurar o Apache para usar o mod\_jk é necessário alterar o arquivo httpd.conf e incluir os seguintes comandos.

```
# Include mod_jk's specific configuration file
Include conf/mod-jk.conf
```

Depois, é criado um arquivo chamado mod-jk.conf, que é colocado na pasta Apache\_home/conf. O arquivo ficará como mostrado na imagem abaixo.

```

# Load mod_jk module
# Specify the filename of the mod_jk lib
LoadModule jk_module modules/mod_jk.so

# Where to find workers.properties
JkWorkersFile conf/workers.properties

# Where to put jk logs
JkLogFile logs/mod_jk.log

# Set the jk log level [debug/error/info]
JkLogLevel info

# Select the log format
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"

# JkOptions indicates to send SSK KEY SIZE
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# JkRequestLogFormat
JkRequestLogFormat "%w %V %T"

# Mount your applications
JkMount /application/* loadbalancer

# You can use external file for mount points.
# It will be checked for updates each 60 seconds.
# The format of the file is: /url=worker
# /examples/*=loadbalancer
JkMountFile conf/uriworkermap.properties

# Add shared memory.
# This directive is present with 1.2.10 and
# later versions of mod_jk, and is needed for
# for load balancing to work properly
JkShmFile logs/jk.shm

# Add jkstatus for managing runtime data
<Location /jkstatus/>
  JkMount status
  Order deny,allow
  Deny from all
  Allow from 127.0.0.1
</Location>

```

Figura 11: Configurar Apache para usar mod\_jk  
 Fonte: Adaptado por ANDRADE, Rafael

A configuração mais importante da imagem acima é a Jkmount, que diz para o servidor Apache quais requisições devem ser tratadas pelo mod-jk, no exemplo acima, todas as requisições HTTP com application serão redirecionadas para o mod\_jk.

Após a configuração do mod\_jk é necessário configurar os work nodes, que fica no arquivo conf/worker.properties. Este arquivo serve para configurar os diferentes servlets, ou aplicações que estão nos servidores e também configura como funcionará o load-balance. O arquivo abaixo exemplifica as configurações dos workers.

```
# Define list of workers that will be used
# for mapping requests
worker.list=loadbalancer,status

# Define Node1
# modify the host as your host IP or DNS name.
worker.node1.port=8009
worker.node1.host=node1.mydomain.com
worker.node1.type=ajp13
worker.node1.lbfactor=1
worker.node1.cachesize=10

# Define Node2
# modify the host as your host IP or DNS name.
worker.node2.port=8009
worker.node2.host= node2.mydomain.com
worker.node2.type=ajp13
worker.node2.lbfactor=1
worker.node2.cachesize=10

# Load-balancing behaviour
worker.loadbalancer.type=lb
worker.loadbalancer.balance_workers=node1,node2
worker.loadbalancer.sticky_session=1
#worker.list=loadbalancer

# Status worker for managing load balancer
worker.status.type=status
```

Figura 12: Configuração dos workers nodes  
Fonte: Adaptado por ANDRADE, Rafael

A imagem acima mostra a configuração que faz com que o mod\_jk use o load-balance do tipo round Robin com sticky sessions em dois servidores Jboss tomcat.

### 3.5 ARQUITETURA PROPOSTA

Para a garantia a tolerância a falhas de hardware, a solução mais utilizada atualmente é a clusterização ou replicação de hardware, que apesar de ser custosa, pois necessidade redundância de hardwares se mostra eficiente neste ponto.

A arquitetura proposta neste artigo se baseia em clusterização de servidores de aplicação Jboss juntamente com replicação de dados Mysql, conforme mostrado figura abaixo.

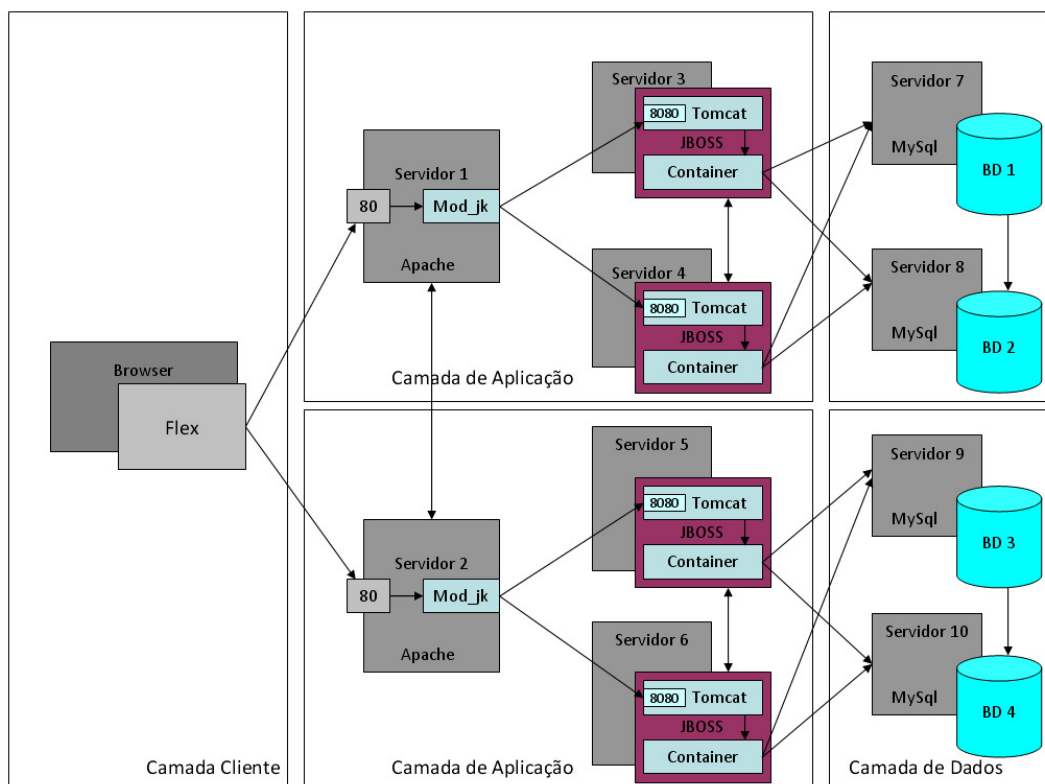


Figura 13: Arquitetura proposta  
Fonte: Adaptado por Andrade, Rafael

Esta arquitetura está dividida em quatro camadas, elas são a camada Cliente, camada de Aplicação e camada de Dados. Cada camada será melhor especificada nos próximos tópicos.

### 3.5.1 Camada Cliente

A camada de Cliente consiste na parte aonde usuário final realmente participa, ou seja, onde ficaria o “consumidor” da aplicação, por se tratar de uma aplicação web, o consumidor dela é o próprio browser do usuário, fazendo assim que para acessar a aplicação ele necessite somente uma conexão a internet. A camada de cliente envia sempre dois requests para os servidores replicados, para garantir que o mesmo request seja recebido pelos dois.

### 3.5.2 Camada de Aplicação

A camada de Aplicação consiste em dois servidores que contém configurado neles o servidor HTTP Apache, o servidor Apache está no mercado desde 1996, e atualmente é o servidor mais estável e mais utilizado, sempre se mantendo atualizado com os padrões HTTP. Juntamente com o servidor Apache existe seu complemento o mod\_jk, que consiste em um complemento usado para se trabalhar com o servidor Apache juntamente com servlets, especialmente servlets Java.

O mod\_jk é um conector usado pelo Apache para se conectar ao servidor de aplicação Apache Tomcat usando o protocolo de comunicação AJP. Neste servidor ocorre o load balancing das aplicações, ou seja, ele seleciona qual dos servidores irá retornar a resposta para o browser do usuário. O mod\_jk trabalha com loading balance feito por Java, aonde ele tem a listagem de servidores disponíveis para acesso e a partir de um algoritmo chamado Round Robin ele seleciona o mais apropriado para dar a resposta a esse usuário. Nele também é configurada uma opção chamada sticky sessions, que se trata de uma configuração que diz que um request feito por algum browser será sempre respondido pelo mesmo servidor, isso é utilizado para diminuir o overheading entre os servidores de aplicação para replicar os dados das sessões dos usuários.

Estes servidores trabalham em cluster utilizando o Jgroups que será melhor explicado abaixo, eles trabalham em cluster de forma que mantém todos os dados de todas as sessões de usuários que estão usando o sistema em todos eles, ou seja, ocorre uma replicação das sessões de usuários, esse procedimento é chamado de State Replication.

Os servidores mantêm uma lista de todos ips onde estão os outros servidores, e assim ele enviam requisições ips para os outros com os dados necessários referentes às sessões dos usuários. Isso garante que em caso de falha em alguns dos servidores, qualquer outro poderá continuar tratando a sessão do usuário corretamente sem que o mesmo perceba a falha ocorrida.

Além dos servidores com o Apache, na camada de Aplicação contém mais dois servidores com o container Jboss, trabalhando em cluster, comunicando-se entre si. O container Jboss. O Jboss contém internamente a ele, o servidor de aplicações Tomcat, também desenvolvido pela Apache, o Tomcat trata-se do servidor de aplicação mais utilizado no mercado em conjunto com o Jboss, pois trabalha de acordo com as

especificações W3C, e também trabalha de forma otimizada com o servidor HTTP Apache. Nestes servidores é onde a aplicação está, ou seja, é neste servidor aonde ocorre o deploy da aplicação.

O cliente final nunca vê este servidor, ele somente se comunica com os servidores onde está configurado o servidor Apache, e o mesmo redireciona para estes servidores.

### 3.5.2.1 Jgroups

Jgroups é uma ferramenta para envio de mensagens multicast confiável. Ele se baseia na comunicação P2P entre os nodes dos clusters, ele pode ser baseado nos seguintes protocolos de comunicação: UDP, TCP e JMS. A principal característica do mesmo é a garantia de envio das mensagens entre os nodes, em caso de falha de alguma mensagem, ela será reenviada.

Jgroups também trabalha com o grupo, ou seja, ele controla todo o cluster, mantendo sempre uma lista atualizada de quem está conectado ou não no cluster, e quando há entrada de algum novo node, ele avisa a todos os outros para atualizarem sua listagem.

A arquitetura do Jgroups é especificada conforme mostrado abaixo:

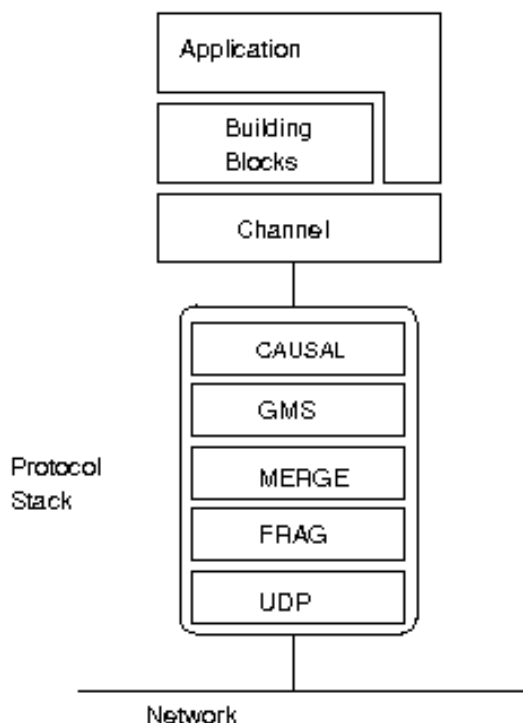


Figura 14: Arquitetura Jgroups

Fonte: <http://www.jgroups.org/manual/html/ch01.html#ArchitectureFig>



A arquitetura consiste em três partes:

A API channel de comunicação entre a aplicação e o Jgroups.

O Building Blocks que servem para facilitar o uso do Jgroups e dar uma maior abstração para quem usar o framework. Nesta camada também é aonde é implementado os protocolos de garantia de envio das mensagens.

E o Protocol Stack, que consiste na implementação da comunicação especificada pelo canal.

Jgroups é atualmente o framework mais utilizado para a comunicação confiável entre grupos, utilizado no Jboss, Tomcat, Jetty, entre outros casos de sucesso.

### **3.5.3 Camada de Banco de Dados**

A camada de Banco de dados consiste em mais dois servidores aonde somente roda o banco de dados Mysql. Estes servidores se comunicam entre si de forma master-slave, para garantir a replicação de dados sempre atualizada. Ou seja, é criado um cluster de banco de dados utilizando Mysql.

A camada de Aplicação se comunica com esta camada utilizando JDBC, que se trata de uma interface para conectar Java a banco de dados. A camada de aplicação ao tentar se comunicar com o banco de dados, caso o servidor principal esteja fora do ar, a aplicação se comunica com o próximo servidor de banco de dados do cluster.

### **3.5.4 Configuração da arquitetura proposta**

Para configurar os servidores utilizados na arquitetura proposta foram feitos os seguintes passos.

#### *3.5.4.1 Configurações dos servidores de aplicação (servidor 3, servidor 4, servidor 5, servidor 6)*

Os servidores do cluster serão chamados de node 1 e node 2, e neles rodará os servidores de aplicação (Jboss), o IP interno do servidor 3 é 192.168.1.159, do servidor 4 é 192.168.1.158, do servidor 5 é 192.168.1.8 e do servidor 6 é 192.168.1.2.

Primeiramente para rodar um cluster no servidor Jboss é somente necessário iniciá-lo com a configuração all que assim ele terá as configurações necessárias para o cluster.

Serão formados dois cluster, um utilizando o servidor 3 e servidor 4 e outro utilizando o servidor 5 e servidor 6, portanto somente essas maquinas se comunicarão entre si, e assim o servidor 3 não terá comunicação com servidor 5 e assim por diante.

Após isso, é necessário configurar os dados referentes à localização, nome e outros dados de cada node do cluster, e para isso, o Jboss utiliza o Jgroups, e para a configuração deste cluster foram usadas as seguintes configurações. Estas configurações são feitas no arquivo JBOSS\_HOME\server\all\deploy\ cluster-service.xml.

```
<Config>
  <TCP bind_addr="{jboss.bind.address}" start_port="7800" loopback="true"
    tcp_nodelay="true"
    rcv_buf_size="2000000"
    send_buf_size="640000"
    discard_incompatible_packets="true"
    enable_bundling="false"
    max_bundle_size="64000"
    max_bundle_timeout="30"
    use_incoming_packet_handler="true"
    use_outgoing_packet_handler="false"
    down_thread="false" up_thread="false"
    use_send_queues="false"
    sock_conn_timeout="300"
    skip_suspected_members="true"/>
  <TCPPING initial_hosts="{jboss.bind.address}[7800],192.168.1.153[7800]" port_range="3"
    timeout="3000"
    down_thread="false" up_thread="false"
    num_initial_members="2"/>
  <MERGE2 max_interval="100000"
    down_thread="false" up_thread="false" min_interval="20000"/>
  <FD_SOCKET down_thread="false" up_thread="false"/>
  <FD timeout="10000" max_tries="5" down_thread="false" up_thread="false" shun="true"/>
  <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
  <pbcast.NAKACK max_xmit_size="60000"
    use_mcast_xmit="false" gc_lag="0"
    retransmit_timeout="300,600,1200,2400,4800"
    down_thread="false" up_thread="false"
    discard_delivered_msgs="true"/>
  <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
    down_thread="false" up_thread="false"
    max_bytes="400000"/>
  <pbcast.GMS print_local_addr="true" join_timeout="3000"
    down_thread="false" up_thread="false"
    join_retry_timeout="2000" shun="true"
    view_bundling="true"/>
  <pbcast.STATE_TRANSFER down_thread="false" up_thread="false" use_flush="false"/>
</Config>
```

Figura 15: Configuração do Node 1 do cluster

Fonte: Adaptado por ANDRADE, Rafael

Para a aplicação do cluster, foi escolhido a opção de utilizar as configurações usando TCP, pois apesar de ter uma carga maior na rede, ele garante melhor a entrega dos pacotes entre os servidores.

As principais configurações usadas serão explicadas abaixo:

TCP - identifica que o sistema usará o protocolo TCP para fazer a comunicação entre os nodes;

TCPPING – identifica o protocolo usado para fazer a busca dos servidores disponíveis para o Cluster;

MERGE2 – é usado para que em caso de falha da rede, os coordenadores dos grupos possam restaurar o cluster;

FD SOCK – identifica o tipo de detecção de falha usado no cluster;

VERIFY\_SUSPECT – identifica o que leva um cluster a identificar outro como suspeito de estar em falha;

pbcast.NAKACK – Identifica a forma de garantia de entrega das mensagens usadas no cluster;

pbcast.GMS – identifica a forma usada para garantir a entrada de novos nodes no cluster;

pbcast.STABLE – Identifica que está sendo usado um garbage collector para todo cluster, pois se for guardar todas as mensagens enviadas entre os nodes, poderá ocorrer uma falta de memória nos nodes;

pbcast.STATE\_TRANSFER – identifica que cada node novo que entrar no cluster irá receber o estado do cluster.

E assim o primeiro servidor está pronto para rodar em cluster, faltando somente configurar o segundo servidor do cluster, o node 2.

Para configurar o node 2 do cluster, se utiliza as mesmas configurações do node 1, mudando somente o IP de identificação dos servidores, e assim sua configuração ficará conforme mostrado a seguir:

```

<Config>
  <TCP bind_addr="192.168.1.158" start_port="7800" loopback="true"
    tcp_nodelay="true"
    recv_buf_size="20000000"
    send_buf_size="640000"
    discard_incompatible_packets="true"
    enable_bundling="false"
    max_bundle_size="64000"
    max_bundle_timeout="30"
    use_incoming_packet_handler="true"
    use_outgoing_packet_handler="false"
    down_thread="false" up_thread="false"
    use_send_queues="false"
    sock_conn_timeout="300"
    skip_suspected_members="true"/>
  <TCPPING initial_hosts="192.168.1.158[7800],192.168.1.159[7800]" port_range="3"
    timeout="3000"
    down_thread="false" up_thread="false"
    num_initial_members="3"/>
  <MERGE2 max_interval="100000"
    down_thread="false" up_thread="false" min_interval="20000"/>
  <FD_SOCK down_thread="false" up_thread="false"/>
  <FD timeout="10000" max_tries="5" down_thread="false" up_thread="false" shun="true"/>
  <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
  <pbcast.NAKACK max_xmit_size="60000"
    use_mcast_xmit="false" gc_lag="0"
    retransmit_timeout="300,600,1200,2400,4800"
    down_thread="false" up_thread="false"
    discard_delivered_msgs="true"/>
  <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
    down_thread="false" up_thread="false"
    max_bytes="400000"/>
  <pbcast.GMS print_local_addr="true" join_timeout="3000"
    down_thread="false" up_thread="false"
    join_retry_timeout="2000" shun="true"
    view_bundling="true"/>
  <pbcast.STATE_TRANSFER down_thread="false" up_thread="false" use_flush="false"/>
</Config>

```

Figura 16: Configuração do Node 2 do cluster.  
 Fonte: Adaptado por ANDRADE, Rafael

Com estas configurações citadas acima, estará pronto a configuração do cluster, e ao iniciar os servidores, poderá se verificar ambos se comunicando. A figura a seguir exemplifica essa comunicação:

```

12:37:57,375 INFO [DefaultPartition] Initializing
12:37:57,468 INFO [STDOUT]
-----
GMS: address is 192.168.1.159:7800
-----
12:38:02,453 WARN [ConnectionTable] peer closed connection, trying to re-send message
12:38:02,453 ERROR [ConnectionTable] 2nd attempt to send data failed too
12:38:02,453 INFO [DefaultPartition] Number of cluster members: 2
12:38:02,453 INFO [DefaultPartition] Other members: 1
12:38:02,453 INFO [DefaultPartition] Fetching state (will wait for 30000 milliseconds):
12:38:02,687 INFO [DefaultPartition] state was retrieved successfully (in 234 milliseconds)
12:38:02,796 INFO [HANamingService] Started ha-jndi bootstrap jnpPort=1100, backlog=50, bindAddress=/0.0.0.0
12:38:02,812 INFO [DetachedHANamingService$AutomaticDiscovery] Listening on /0.0.0.0:1102, group=230.0.0.4, HA-JNDI address=192.168.1.159:1100
12:38:03,031 INFO [TreeCache] No transaction manager lookup class has been defined. Transactions cannot be used
12:38:05,468 INFO [STDOUT]
-----
GMS: address is 192.168.1.159:2105
-----

```

Figura 17: Exemplo do Cluster para node que está entrando  
Fonte: Adaptado por ANDRADE, Rafael

```

12:40:39,718 INFO [DefaultPartition] I am (192.168.1.153:1098) received membershipChanged event:
12:40:39,718 INFO [DefaultPartition] Dead members: 1 ([192.168.1.159:1098])
12:40:39,718 INFO [DefaultPartition] New Members : 0 ([])
12:40:39,718 INFO [DefaultPartition] All Members : 1 ([192.168.1.153:1098])
12:40:40,031 INFO [TreeCache] viewAccepted(): [192.168.1.153:1040;2] [192.168.1.153:1040]
12:40:55,093 INFO [TreeCache] viewAccepted(): [192.168.1.153:1040;3] [192.168.1.153:1040, 192.168.1.159:2131]
12:40:55,281 INFO [TreeCache] locking the subtree at / to transfer state
12:40:55,281 INFO [StateTransferGenerator_140] returning the state for tree rooted in / (1024 bytes)
12:41:02,921 INFO [DefaultPartition] New cluster view for partition DefaultPartition (id: 3, delta: 1) : [192.168.1.153:1098, 192.168.1.159:1098]
12:41:02,921 INFO [DefaultPartition] I am (192.168.1.153:1098) received membershipChanged event:
12:41:02,921 INFO [DefaultPartition] Dead members: 0 ([])
12:41:02,921 INFO [DefaultPartition] New Members : 1 ([192.168.1.159:1098])
12:41:02,921 INFO [DefaultPartition] All Members : 2 ([192.168.1.153:1098, 192.168.1.159:1098])
12:41:07,156 INFO [TreeCache] viewAccepted(): [192.168.1.153:1053;3] [192.168.1.153:1053, 192.168.1.159:2153]
12:41:09,531 INFO [TreeCache] viewAccepted(): [192.168.1.153:1057;3] [192.168.1.153:1057, 192.168.1.159:2158]

```

Figura 18: Exemplo de um coordenador do Cluster  
Fonte: Adaptado por ANDRADE, Rafael

Lembrando que para iniciar os servidores é necessário somente ir na pasta `JBOSS_HOME\bin` e executar o comando `run -c all -b 0.0.0.0`.

O comando `-c all` identifica que será rodado o servidor com todas as configurações de cluster e cache, e a identificação `-b 0.0.0.0` se fez necessário para que as máquinas se identifiquem na rede, sem isso o cluster não se “encontra”.

Para adicionar novas aplicações no cluster, a melhor forma de fazer é usar o Farm Deploy do Jboss, pois ele garante que todos os sistemas rodando em todos os nós do cluster serão iguais. Ele funciona de forma que ao colocar uma aplicação na

pasta JBOSS\_HOME\server\all\farm, e assim ele inicializará a aplicação e também enviara esta aplicação para todos nodes do cluster, para que assim o outros nodes também tenham a aplicação.

E assim o clusterização de servidores estará pronta. As configurações mostradas se repetem para o servidor 5 e servidor 6, por isso não foram mostradas.

#### *3.5.4.2 Configuração dos servidores de Banco de dados*

Para configurar a replicação de dados nos banco de dados, foram feitos alguns passos, que serão explicados abaixo.

##### **3.5.4.2.1 Configuração do Master**

Ao iniciar a configuração de replicação de dados no Mysql, é necessário primeiramente configurar o servidor que será considerado o “master”, ou seja, todas as requisições da aplicação serão enviadas para ele, e ele repassará essas requisições para os bancos de dados “slaves”. Devido um dos requisitos inicialmente especificados para o sistema, todos servidores deverão ser Windows, por isso as configurações mostradas são para servidores Windows, tendo alguma diferença para servidores usando Linux.

Por convenção, vamos chamar a pasta aonde foi instalado o Mysql de MYSQL\_HOME.

Assim, inicia-se alterando o arquivo my.ini, que fica no diretório MYSQL\_HOME\MySql Server 5.1\my.ini, neste arquivo estão as principais configurações do banco de dados, como o tipo de dados que serão usado, charset, entre outros. Inicialmente é necessário configurar o banco para que use a rede, portanto se fez necessário retirar as seguintes linhas da configuração:

```
#skip-networking  
#bind-address = 127.0.0.1
```

Após isso, é necessário dizer ao banco qual database ele deverá ter logs (para posteriormente serem lidos pelos servidores slaves). Como neste trabalho desejamos

configurar a replicação para a base “direseg”, as configurações de log ficaram como mostra a imagem abaixo:

```
long-bin = C:/mysql-bin.log
binlog-do-db=direseg
server-id=1
```

Após alterar estas configurações, é necessário reiniciar o banco de dados, para que assim ele comece a utilizar o log. Seguido a estas configurações, é necessário obter as informações sobre o arquivo de log usado pelo banco de dados “master”, para poder posteriormente configurar os “slaves”, e isso se faz usando os seguintes comandos no Mysql:

```
USE direseg;
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
```

Ao executar isto, foi obtida a seguinte resposta:

```
Microsoft Windows XP [versão 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\rafael>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.34-community-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000002 |      106 | direseg      |                    |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _
```

Figura 19: Exemplo tabela mysql  
Fonte: Adaptado por ANDRADE, Rafael

É importante anotar os dados que aparecem, pois eles deverão ser usados posteriormente para configurar os slaves.

Após isto, o banco de dados “master” da replicação está pronto. Agora falta somente configurar os slaves.

### 3.5.4.2.2 Configuração dos Slaves

Para configurar os “slaves”, primeiramente é necessário criar o database direseg nos mesmo, e após isso, é necessário alterar as configurações do arquivo my.ini para informar os dados referentes a replicação e localização do servidor “máster”. A figura abaixo ilustrar as configurações usadas no servidor “slave”.

```
server-id=2
master-host=192.168.1.153
master-user=root
master-password=jexpadm
master-connect-retry=60
replicate-do-db=direseg
```

Figura 20: Configuração do banco de dados “slave”  
Fonte: Adaptado por ANDRADE, Rafael

Após aplicar as configurações, é necessário reiniciar o banco de dados. E assim, para que o “slave” esteja em conformidade com o “máster”, se utiliza o seguinte comando:

```
LOAD DATA FROM MASTER;
```

E assim, o “slave” terá todos os dados do banco “master”, dados somente do schema “direseg”, pois é o único que interessa para no momento.

Após isso, é necessário para a thread Slave do banco de dados, para atualizar as informações referentes ao arquivo de log e posição do arquivo do máster (os dados obtidos anteriormente com o comando SHOW MASTER STATUS;), e para isso foi executado o comando:

```
SLAVE STOP;
```

E logo após o comando:

```
CHANGE MASTER TO MASTER_HOST='192.168.1.153', MASTER_USER='root',
MASTER_PASSWORD='jexpadm', MASTER_LOG_FILE='mysql-bin.000002',
MASTER_LOG_POS=106;
```



Este comando adiciona ou altera as configurações dos referentes ao banco de dados máster. E as variáveis usadas são:

- MASTER\_HOST – IP da máquina que possui o banco de dados máster
- MASTER\_USER – usuário que está na máquina máster, que será usado para obter os dados.
- MASTER\_PASSWORD – senha do usuário.
- MASTER\_LOG\_FILE – qual arquivo está sendo usado pelo máster para salvar todas as transações feitas nele.
- MASTER\_LOG\_POS – posição a qual o arquivo será lido para manter sempre atualizado.

Com estas configurações prontas, é apenas necessário executar o comando:

```
START SLAVE;
```

E assim a replicação entre os servidores está pronta, todas as requisições e transações que ocorrerem no banco de dados máster será replicada para os slaves.

Lembrando que para fazer a replicação, é necessário uma rede local, e a porta 3306 deverá estar liberada no firewall, pois o firewall do Windows bloqueia esta porta para conexões externas.

E com estas configurações ficará pronta a replicação de dados usando Mysql. Lembrando que as configurações se repetem nos dois nós do cluster, alterando somente o ip das máquinas usadas.

### **3.5.5 Avaliação de Desempenho**

Visando verificar o desempenho da infra-estrutura proposta foram executados testes em uma rede local de 10mbps compostas por computadores Intel Dual Core de 1.6 GHz com 2gb de memória RAM e sistema operacional Microsoft Windows XP. Os computadores foram configurados conforme mostrado na arquitetura proposta mostrada anteriormente.

A primeira análise de desempenho feita nesta arquitetura foi o tempo de resposta a uma requisição enviada ao servidor comparada com o número de serviços disponíveis no cluster. A mensagem enviada para o servidor foi uma requisição HTTP,

e a resposta do servidor é sempre uma resposta de contém 0 bytes, tendo somente os bytes do protocolo HTTP. De acordo com o gráfico abaixo, quando se tem somente um serviço o tempo de resposta do servidor é de 40ms(milissegundos), e quando se coloca mais um serviço no cluster, o número praticamente dobra, passando a ~70ms, e assim se percebe um crescimento com o número de serviços. Os testes foram executados com mensagens de tamanho de 0-256k-512k bytes, e os resultados completos estão mais especificados abaixo.

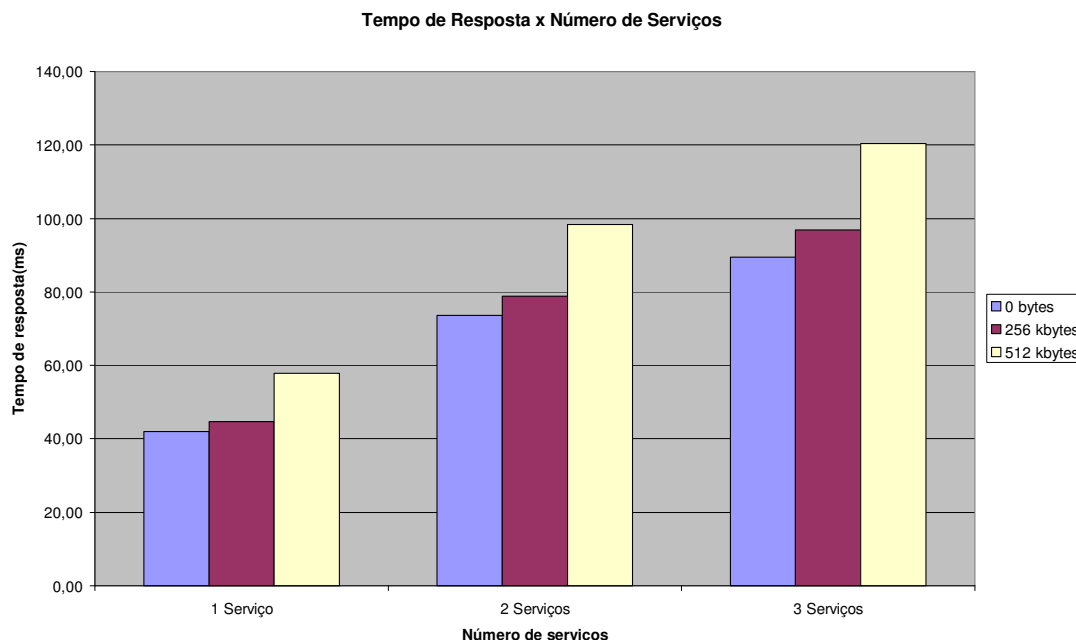


Figura 21: Desempenho do serviço  
Fonte: Adaptado por Andrade, Rafael

Outra análise de desempenho feita foi a de tempo de resposta considerando o número de clientes fazendo a requisição ao servidor. A mensagem a ser enviada foi de 512kbytes, que seria a simulação de uma listagem no servidor. O número de cliente variou entre 10 e 80 clientes. Também foram feitos de acordo com o número de serviços disponíveis. Para fazer estes testes, foi implementado um client em Java, que cria threads de disparo de requisições para o servidor, cada thread envia 100 requisições ao servidor, e após isso é calculado o tempo médio de resposta entre essas requisições. O único problema identifica nesta abordagem é que o Java tem um tempo de resposta maior do que o Flex, por isso os dados ficaram acima do que

realmente seria calculado usando clients em Flex. Os resultados obtidos estão mostrados no gráfico abaixo

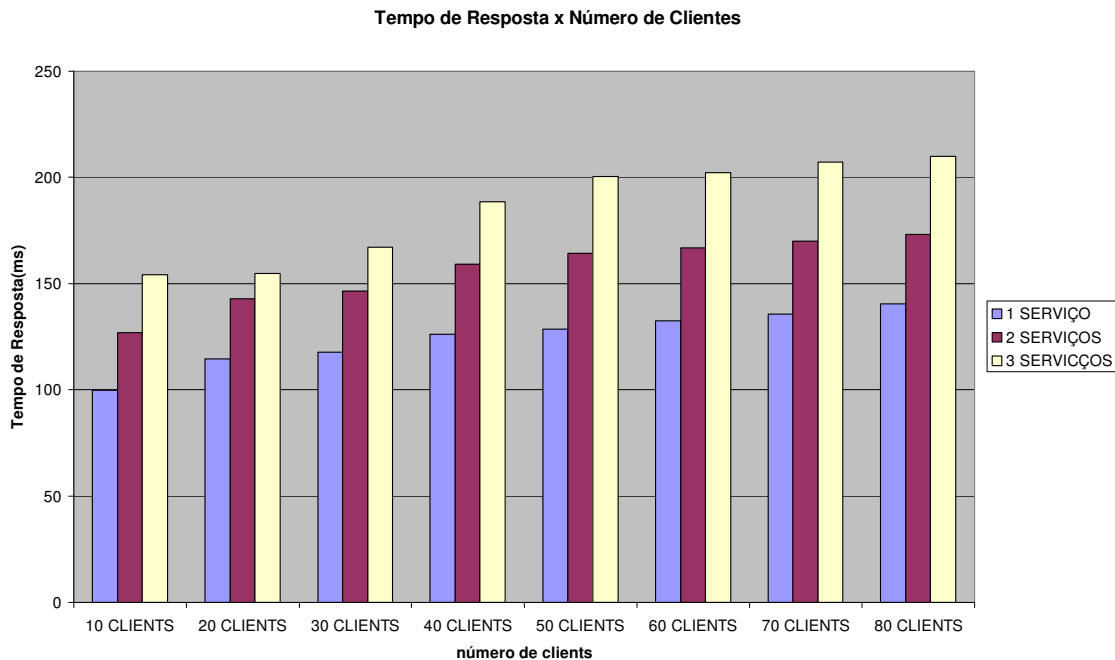


Figura 22: Desempenho do serviço x Número de Clientes  
Fonte: Adaptado por Andrade, Rafael

Como pode ser visto no gráfico, a tempo de resposta aumenta de forma estável de acordo com o número de clientes enviando requisições, pode-se comprovar uma média de aumento de cerca de 15% no tempo de resposta para cada aumento de 10 Clientes enviando requisições, o que é considerado um desempenho muito bom para aplicações web. Sendo que uma requisição chegando até 80ms é praticamente imperceptível para o ser humano, e o maior tempo de resposta obtido é ~200ms.

Não é possível calcular especificamente o overhead causado para detector de falhas usado no servidor Jboss, mas de acordo com o que foi especificado na documentação do próprio servidor, o overhead causado pelos detectores de falhas é praticamente nulo, aumentando de 2 a 5 por cento somente o tempo de resposta a uma requisição.

#### 4 TRABALHOS FUTUROS

Para o futuro, pretendo fazer maior garantia de tolerância a falhas, não somente a falhas de hardware, como também a falhas de desenvolvimento como outras, para assim ter uma garantia maior na confiabilidade do sistema.

A aplicação em desenvolvimento está em fase *alpha*, e em desenvolvimento, portanto um dos principais trabalhos futuros será a finalização da mesma.

Outro trabalho futuro é o desenvolvimento de técnicas de replicação de servidores em máquinas virtuais, diminuindo assim o custo de implantação da arquitetura proposta neste trabalho.

## 5 CONSIDERAÇÕES FINAIS

Atualmente o desenvolvimento de aplicação Web está cada vez maior, e diante desta nova realidade, estão sendo necessárias novas formas de garantia para garantir a confiabilidade desses sistemas.

Como a web é ainda uma forma de utilização pouco confiável, muito sujeita a falhas, como quedas de conexão, quedas de luz, entre outras, as garantias de tolerância a essas falhas em sistemas web está se tornando mais necessária.

O trabalho mostra uma forma de garantir a tolerância a falhas, que se mostra bastante confiável, que garante que em caso de falhas de qualquer um dos servidores, a transferência para outro servidor ficará invisível para o usuário final.

A replicação de dados utilizada mostrou-se interessante para garantir à tolerância a falha em algum dos servidores de banco de dados, pois em caso de falha, pode-se automaticamente conectar-se a outro, e este terá os dados idênticos ao outro.

A forma utilizada para a garantia a falhas mostrou-se eficiente, porém, é uma forma com um custo elevado para desenvolvimento, tanto com *hardwares*, pois a arquitetura proposta necessita muitas máquinas na mesma, como também custo de implementação, pois não existe quase material sobre os temas escolhidos, e isso faz com que o custo de busca e implementação destas tecnologias se torne custoso.

Por se tratar de um sistema que terá usuários reais finais, o processo de desenvolvimento do mesmo foi tratado como um projeto de Sistema, por isso foram desenvolvidos documentos e outras formas de evidência para garantir a qualidade do mesmo, o que ajudou bastante no desenvolvimento do mesmo, que mesmo em fase *alpha* de seu desenvolvimento se mostra bastante maduro.

## 6 REFERÊNCIAS

WIKIPEDIA. Disponível em:

[http://en.wikipedia.org/wiki/Replication\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Replication_(computer_science)). Acesso em: 12 jul. 2008.

CLUSTER BANCO DE DADOS. Disponível em:

<http://www.docs.mussicorp.net/guiacuster/clusterbancodedados.php>. Acesso em: 12 jul. 2008

REPLICATION AND DATABASE MIRRORING. Disponível em:

<http://msdn.microsoft.com/en-us/library/ms151799.aspx>. Acesso em: 22 jul. 2009.

DISTRIBUTED DATABASE. Disponível em:

[http://en.wikipedia.org/wiki/Distributed\\_database](http://en.wikipedia.org/wiki/Distributed_database). Acesso em 15 ago. 2008

Oracle9i Database Administrator's Guide, Disponível em:

[http://www.cs.uvm.edu/oracle9doc/server.901/a90117/ds\\_conce.htm](http://www.cs.uvm.edu/oracle9doc/server.901/a90117/ds_conce.htm), Acesso em: 22 set. 2008

Oracle® Database Advanced Replication 10g Release 1 (10.1). Disponível em:

<http://stanford.edu/dept/itss/docs/oracle/10g/server.101/b10732/toc.htm>. Acesso em: 30 set. 2008

Oracle Data Replication and Integration. Disponível em:

<http://www.oracle.com/technology/products/dataint/index.html>. Acesso em: 23 abr. 2009.

Oracle Replication. Disponível em:

[http://searchoracle.techtarget.com/news/article/0,289142,sid41\\_gci905154,00.html?Exclusive=True](http://searchoracle.techtarget.com/news/article/0,289142,sid41_gci905154,00.html?Exclusive=True). Acesso em 23 abr. 2009.

Database Oracle Advanced Replication Tutorial. Disponível em:

<http://www.roseindia.net/software-tutorials/detail/13572>. Acesso em: 23 abr. 2009.

Do It Yourself (DIY) Oracle replication. Disponível em:

<http://it.toolbox.com/blogs/data-ruminations/do-it-yourself-diy-oracle-replication-12894>. Acesso em: 23 abr. 2009-10-26

Ian Gilfillan. Database Replication in Mysql. Disponível em: <http://www.databasejournal.com/features/mysql/article.php/3355201/Database-Replication-in-MySQL.htm>. Acesso em 28 abr. 2009.

Step-by-Step how to Setup MYSQL Database Replication. Disponível em: <http://aciddrop.com/2008/01/10/step-by-step-how-to-setup-mysql-database-replication/>. Acesso em: 28 abr. 2009.

Mysql Replication. Disponível em: <http://dev.mysql.com/doc/refman/5.0/en/replication.html>. Acesso em: 28 abr. 2009.

Falko Timme. How To Set Up Database Replication In Mysql. Disponível em: [http://www.howtoforge.com/mysql\\_database\\_replication](http://www.howtoforge.com/mysql_database_replication). Acesso em: 28 abr. 2009.

Daniel. Setting up database replication on Mysql. Disponível em: <http://www.gra2.com/article.php/setting-up-database-replication-on-mysql>. Acesso em: 30 abr. 2009.

Database Replication In Mysql. Disponível em: <http://librenix.com/?inode=8118>. Acesso em: 30 abr. 2009.

Slony-I 1.0.5 (PostgreSQL Database Replication). Disponível em: <http://www.postgresql.org/about/news.233>. Acesso em: 30 abr. 2009.

PostgreSQL - Replication and High Availability. Disponível em: <http://edoceo.com/liber/db-postgresql-replication>. Acesso em: 30 abr. 2009.

PGCluster. Disponível em: <http://pgcluster.projects.postgresql.org/>. Acesso em: 30 abr. 2009.

Slony-I. Disponível em: <http://slony1.projects.postgresql.org/>. Acesso em: 30 abr. 2009.

DBBalancer. Disponível em: <http://sourceforge.net/projects/dbbalancer/>. Acesso em 30 abr. 2009.

PgPool. Disponível em: <http://pgpool.projects.postgresql.org/>. Acesso em 30 abr. 2009.

PostgreSQL table comparator. Disponível em: <http://pg-comparator.projects.postgresql.org/>. Acesso em 30 abr. 2009.

Database replication under Postgres. Disponível em:  
<http://archives.postgresql.org/pgsql-interfaces/1999-12/msg00191.php>. Acesso em 5 mai. 2009.

Database Replication. Disponível em:  
<http://www.cs.mcgill.ca/~kemme/disl/replication.html>, Acesso em: 5 mai. 2009.

PostgreSQL + Replication. Disponível em:  
<http://www.commandprompt.com/products/mammothreplicator/>. Acesso em: 5 mai. 2009.

Irina Sourikova. PostgreSQL Replicator. Disponível em:  
[http://www.google.com.br/url?sa=t&source=web&ct=res&cd=9&url=http%3A%2F%2Fwww.rhic.bnl.gov%2FRCF%2FUserInfo%2FMeetings%2FTechnology%2FIrina\\_Sourikova\\_techMeet.ppt&ei=U70YSfuzllyi8gTM1sCfCw&usg=AFQjCNHdOqqGew5a31NxDnRS77YI8DMx2w&sig2=pmHi-11Ra4vvvNA5e7moAA](http://www.google.com.br/url?sa=t&source=web&ct=res&cd=9&url=http%3A%2F%2Fwww.rhic.bnl.gov%2FRCF%2FUserInfo%2FMeetings%2FTechnology%2FIrina_Sourikova_techMeet.ppt&ei=U70YSfuzllyi8gTM1sCfCw&usg=AFQjCNHdOqqGew5a31NxDnRS77YI8DMx2w&sig2=pmHi-11Ra4vvvNA5e7moAA). Acesso em: 10 mai. 2009.

Fault-Tolerant system. Disponível em: [http://en.wikipedia.org/wiki/Fault-tolerant\\_system](http://en.wikipedia.org/wiki/Fault-tolerant_system). Acesso em: 12 mai. 2009.

Walter Heimerdinger , Charles B. Weinstock. Conceptual Framework for System Fault Tolerance, A. Disponível em:  
<http://www.sei.cmu.edu/library/abstracts/reports/92tr033.cfm>. Acesso em: 12 mai. 2009.

Fault Tolerance Concepts. Disponível em:  
[http://hissa.nist.gov/chissa/SEI\\_Framework/framework\\_8.html](http://hissa.nist.gov/chissa/SEI_Framework/framework_8.html). Acesso em: 12 mai. 2009.

Pankaj Jalote. Fault Tolerance in Distributed Systems. 1Ed. 2006.

Taisy Weber. Conceitos de Dependabilidade. Disponível em:  
<http://www.inf.ufrgs.br/~taisy/disciplinas/textos/ConceitosDependabilidade.PDF>. Acesso em: 12 mai. 2009.

Jboss 4 Clustering Guide. Disponível em:  
<http://www.google.com.br/url?sa=t&source=web&ct=res&cd=2&url=http%3A%2F%2Fdocs.jboss.org%2Fjbossas%2Fguides%2Fclusteringguide%2Fr2%2Fen%2Fpdf%2Fjboss4-clustering.pdf&ei=Db8YSeeYKaCm8ASpvtmTCw&usg=AFQjCNGhTq0da0et4y107ANECQHxC1XHmQ&sig2=Lv0DMdrTd71O-hUuxUzl4w>. Acesso em: 22 mai. 2009.



Jboss Clustering. Disponível em:

<http://docs.jboss.org/jbossas/jboss4guide/r4/html/cluster.chapt.html>. Acesso em: 22 mai. 2009.

Ivelin Ivanov. J2EE Clustering with Jboss. Disponível em:

[http://onjava.com/pub/a/onjava/2003/08/20/jboss\\_clustering.html?page=1](http://onjava.com/pub/a/onjava/2003/08/20/jboss_clustering.html?page=1).

Acesso em: 26 mai. 2009.

Samson Kittoli. JBoss AS5 Clustering Guide. Disponível em:

<http://www.jboss.org/community/wiki/jbossas5clusteringguide>. Acesso em: 26 mai. 2009.

Brian Stansberry, Galder Zamarreno. JBoss Application Server Clustering Guide.

Disponível em: [http://www.jboss.org/file-](http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Clustering_Guide/beta422/html/index.html)

[access/default/members/jbossas/freezone/docs/Clustering\\_Guide/beta422/html/index.html](http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Clustering_Guide/beta422/html/index.html). Acesso em: 26 mai. 2009.

Web 2.0. Disponível em: [http://pt.wikipedia.org/wiki/Web\\_2.0](http://pt.wikipedia.org/wiki/Web_2.0). Acesso em: 30

ago. 2009.

Tim O'Reilly. What Is Web 2.0. Disponível em:

<http://oreilly.com/web2/archive/what-is-web-20.html>. Acesso em: 30 ago. 2009.

## **7 ANEXOS**

Anexo A – Análise de Requisitos

**ANEXO A**

# **DIR\_WEB**

## **Análise de Requisitos**

Versão:01.00

Data:10 maio, 2009

Identificador do documento:documento\_requisitos

Localização:docs/requisitos

## Histórico de revisões

Versão (XX.YY)	Data (DD/MMM/YYYY)	Autor	Descrição	Localização
01.00	10/05/2009	Rafael Andrade	Criação	Docs/requisitos

## **Introdução**

---

### **Propósito**

Esse documento apresenta o conjunto de requisitos funcionais e não funcionais do projeto. A especificação de requisitos envolve o conjunto de necessidades que devem ser atendidas para que os objetivos do projeto sejam plenamente atingidos, e as restrições que devem ser observadas durante o processo de desenvolvimento. Ela deve também estabelecer o relacionamento entre estes objetivos e restrições.

### **Público Alvo**

Esse documento destina-se aos analistas de sistemas, responsáveis pela concepção tecnológica e modelagem da solução do projeto e aos desenvolvedores.

## **Análise da Necessidade**

---

### **Descrição do Projeto**

Projeto para administração e Pedido de novos seguros para a corretora Direseg.

### **Objetivos**

Desenvolver uma aplicação que tenha todas funcionalidades referentes a administração, cotação, relatórios para a corretora Direseg.

### **Metas**

O projeto quando concluído deverá atender os seguintes critérios:

- a) Os usuários das vendedoras de automóveis poderão pedir cotações em tempo real para o seguro dos autos vendidos.
- b) Os usuários da corretora poderão calcular e enviar a resposta aos pedidos em tempo real.
- c) Os usuários da corretora poderão cadastrar seguros de todos seus clientes.
- d) Os usuários da corretora poderão cadastrar produtores para fazer parcerias.
- e) Os usuários das produtoras poderão ver relatórios de produção, e também de relatórios de seguros.

### **Justificativas**

Atualmente verificou-se a demanda para um serviço unificado de administração da corretora e também para pedidos de novos seguros.

## Requisitos Funcionais

---

### Módulo Administrador

<b>RQ0001_00</b>	O sistema deve permitir que todos funcionários da corretora tenha login próprio.		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Marcelo		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0002_00</b>	O sistema deve listar os usuários cadastrados		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed – Rodrigo		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0003_00</b>	O sistema deve permitir que os administradores vejam todos pedidos de cotação, com filtros.		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed – Rodrigo		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0004_00</b>	O sistema deve permitir o cadastro de Ramos		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0005_00</b>	O sistema deve permitir o cadastro de Tipos de cliente		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0006_00</b>	O sistema deve permitir o cadastro de Segurados		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0007_00</b>	O sistema deve permitir o cadastro de Seguros		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0008_00</b>	O sistema deve permitir o cadastro de Produtores		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0009_00</b>	O sistema deve permitir o cadastro de usuários para produtores		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	12/03/2009
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0010_00</b>	O sistema deve permitir o cadastro de Segadoras		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0011_00</b>	O sistema deve permitir o cadastro de Bancos		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0012_00</b>	O sistema deve permitir o cadastro de Perfis de usuário		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0013_00</b>	O sistema deve permitir o cadastro de Comissões de Seguro		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0014_00</b>	O sistema deve permitir o cadastro de Pagamentos de Parcelas de Seguros		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0015_00</b>	O sistema deve permitir o cadastro de Pagamentos de Parcelas de Seguros		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		

<b>RQ0016_00</b>	O sistema deve permitir o cadastro de Perfil de Segurados		
<b>Autor</b>	Rafael Andrade		
<b>Solicitante</b>	Portal Unimed - Vivian		
<b>Prioridade</b>	Alta		
<b>Status</b>	Pendente		



<b>RQ0017_00</b>	O sistema deve permitir o cadastro Coberturas para os Ramos de Seguro
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0018_00</b>	O sistema deve permitir o cadastro de Sinistros
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0019_00</b>	O sistema deve permitir o cadastro e envio de Solicitação entre os usuários do sistema
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0020_00</b>	O sistema deve gerar relatórios de produção, filtrados por Produtor, Ramo, Seguradora, Cliente entre outros
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0021_00</b>	O sistema deve gerar relatórios de clientes, produtores, seguradoras e de todos cadastros do sistema.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0022_00</b>	O sistema deve gerar relatórios de comissão, por data.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

## Módulo Concessionária

<b>RQ0023_00</b>	O sistema deve permitir o cadastro de Pedidos de cotação de seguros
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0024_00</b>	O sistema deve permitir o envio de arquivos anexados aos pedidos de cotação
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0025_00</b>	O sistema deve permitir visualizar as cotações prontas recebidas.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0026_00</b>	O sistema deve gerar relatório de produção do produtor.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0027_00</b>	O sistema deve gerar relatório de comissão do produtor, por data.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

### Módulo Administradora

<b>RQ0028_00</b>	O sistema deve permitir o cadastro de Pedidos de cotação de seguros
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0029_00</b>	O sistema deve permitir o envio de arquivos anexados aos pedidos de cotação
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0030_00</b>	O sistema deve permitir visualizar as cotações prontas recebidas.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0031_00</b>	O sistema deve gerar relatório de produção do produtor.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

<b>RQ0032_00</b>	O sistema deve gerar relatório de comissão do produtor, por data.
<b>Autor</b>	Rafael Andrade
<b>Solicitante</b>	Portal Unimed - Vivian
<b>Prioridade</b>	Alta
<b>Status</b>	Pendente

## Requisitos Não-Funcionais

---

### Confiabilidade e Robustez

<b>RQ0033_00</b>	O sistema deverá rodar no servidor Jboss 4.0.2		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	13/03/2009
<b>Solicitante</b>	Portal Unimed – Rodrigo		
<b>Prioridade</b>	-		
<b>Status</b>	Pendente		

<b>RQ0034_00</b>	O sistema deverá rodar em SO Windows		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	13/03/2009
<b>Solicitante</b>	Portal Unimed – Rodrigo		
<b>Prioridade</b>	-		
<b>Status</b>	Pendente		

### Interação e Usabilidade

<b>RQ0035_00</b>	O sistema deverá ser desenvolvido utilizando a tecnologia Flex		
<b>Autor</b>	Rafael Andrade	<b>Data</b>	13/03/2009
<b>Solicitante</b>	Portal Unimed – Rodrigo		
<b>Prioridade</b>	-		
<b>Status</b>	Pendente		

Anexo B – Artigo

## Uma Infraestrutura para Tolerância a Falhas em Sistemas para Web 2.0

Rafael de Souza Andrade

Centro Tecnológico – Universidade Federal de Santa Catarina (UFSC)  
CEP 88040-900 – Florianópolis – SC – Brazil

Departamento de Informática e Estatística  
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brazil  
rafa86@inf.ufsc.br

**Resumo.** *Este artigo mostra uma técnica de se garantir uma melhor tolerância a falhas para aplicações web, principalmente aplicações web 2.0. Para garantir esta maior tolerância, foram utilizadas técnicas de clusterização de servidores de aplicação Jboss, juntamente com replicação de banco de dados utilizando Mysql. Adicionalmente o artigo faz verificações de desempenho na arquitetura proposta, para verificar seu uso real.*

**Palavras chave:** *Web 2.0, Tolerância a Falhas, Clusterização de Jboss, Mysql.*

**Abstract.** *This article shows a technique to ensure a better fail tolerance for web applications, especially web 2.0 applications. To ensure that greater tolerance were used techniques of clustering application server Jboss, along with database replication data using Mysql. Additionally the article makes checks of performance in the proposed architecture, to verify actual usage.*

**Keywords:** *Web 2.0, Fault-Tolerance, Jboss Clustering, Mysql.*

### 1 Introdução

Estamos presenciando um crescimento cada vez maior da Internet, aonde praticamente tudo em nossas vidas pode ser feito pela mesma. E isso fez com que surgisse uma forte demanda de novas plataformas para desenvolvimento de aplicações Web, por exemplo, Internet banking, e-commerce, e-learning e etc. Todos estes são exemplos reais que aplicações web têm crescido muito e que continuarão assim no futuro.

Atualmente estamos vivendo a fase em que todos sistemas, além de robustos e confiáveis, devem ser amigáveis para os usuários finais. A chamada Web 2.0, aonde a interação do usuário com seus sistemas se torna fácil, amigável e muito mais interessante, é um exemplo dessa nova tendência.

Porém, atualmente a maioria das aplicações desenvolvidas não oferece suporte com garantias de tolerância a falhas. Como por exemplo, se um de seus servidores desliga por alguma falha, o sistema deixa de funcionar. Desta forma, a comunidade científica da área tem proposto diversas soluções tecnológicas para desenvolvimento de aplicações tolerante a faltas para garantir que mesmo em caso de algumas falhas parciais os sistemas possam continuar operacionais.

Garantias de tolerância a faltas tem sido muito estudadas, mas somente para sistemas críticos, e hoje em dia tem aumentado a necessidade de se ter essas garantias em sistemas web, para o funcionamento correto dos mesmos.

Este artigo visa realizar um estudo de técnicas de tolerância a faltas para sistemas web 2.0. A arquitetura proposta neste trabalho garantirá a tolerância a faltas, principalmente faltas de parada (*crash*), onde acontecem erros indesejáveis ou até mesmo perdas devido ao desligamento acidental de sistemas críticos.

## 2 Aplicações Web

Aplicações web são as aplicações desenvolvidas para serem usadas nos browsers dos usuários, sem a necessidade de nenhum outro “programa” específico, somente um acesso a internet, e devido a essa simplicidade este tipo de modelo vem se popularizando cada vez mais entre os usuários da Internet.

Atualmente a maioria das soluções usadas pelas empresas tendem a ser web, devido ao fato de serem aplicações centralizadas em um único servidor, e fazendo com que qualquer usuário tenha acesso a ele facilmente através de uma URI, que consiste no endereço onde fica localizada a aplicação na web.

Estas aplicações tem se popularizado mais na última década, devido a melhoria na infraestrutura das redes e da web em si, permitindo altas velocidades de acesso para sistemas, fazendo assim com que se pudesse criar aplicações cada vez mais robustas e mais completas.

Porém, as aplicações web estão atualmente sofrendo grandes alterações, pois além de completas e robustas, os usuários estão demandando que as mesmas sejam cada vez mais intuitivas e fáceis de usar, e isso tem se tornado um ponto vital para aplicações web.

E nesta tendência estamos vivendo a web 2.0, que se baseia completamente na ideia de que os sistemas devem ter um alta usabilidade, ou seja, devem ser fáceis de usar, e sempre intuitivos para os usuários, além de serem cada vez mais bem elaborados, onde as novas tecnologias tem se mostrado muito eficiente.

Uma das tecnologias atuais mais conhecidas para a web 2.0 é o Flex, uma linguagem de programação desenvolvida pela Adobe, que é baseada em *Action Script* para programar a parte de interface com o usuário. O Flex forma arquivos swf(Flash) para gerar a interação com o usuário final, disponibilizando assim uma melhor interface para o usuário.

## 3 Tolerância a faltas em aplicações web 2.0

A maioria das aplicações web 2.0 não tem suporte para tolerância a faltas, tornando assim seus serviços vulneráveis a qualquer falha, de hardware ou software. O que torna suas aplicações não confiáveis, pois caso algum servidor falhe a aplicação se torna indisponível para o usuário.

A arquitetura proposta se baseia no servidor de aplicação Jboss juntamente com Banco de dados MySQL. Ela se propõe a criar clusters de servidores de aplicação juntamente com clusters de banco de dados, garantindo assim que caso algum dos servidores caia a aplicação não “trave” para o usuário.

O servidor de aplicações Jboss se trata do servidor de aplicação mais usado para aplicações web Java no mercado, está no mercado a muitos anos e é muito estável. Ele oferece suporte para clusterização, cache entre outras funcionalidades que o garantem como um dos melhores servidores do mercado.

### 3.1 Clusterização de Servidores de Aplicação(Jboss)

Clusterização é a possibilidade de poder rodar aplicações em diversos servidores, com esta forma, uma aplicação poderá continuar rodando mesmo em caso de falha de algum dos servidores disponíveis dentro do cluster.

Para fazer clusterização com o Jboss, a forma mais simples é disponibilizar vários servidores dentro de uma mesma rede, e rodá-los com a configuração `-c all`. Assim todos iniciam dentro de um cluster desta rede.

Através da clusterização é possível garantir que as aplicações tenham uma disponibilidade muito maior na rede, pois ela garante que mesmo com a falha de algum servidor, a aplicação continuará rodando transparentemente para o usuário final.

A clusterização do jboss utiliza o framework Jgroups para fazer o controle do cluster, tendo sempre uma máquina considerada o *líder* e as outras enviando requisições a ele.

Jgroups é um framework para comunicação multicast confiável, ou seja, é um framework que trabalha com grupos de máquinas trabalhando em conjunto e contém funções de criação, deleção de grupos, assim como entrar e sair dos mesmos, e também envia avisos a todos os componentes do grupo sobre saída ou entrada de um componente neste grupo.

Este framework garante que as mensagens trocadas entre os servidores do cluster realmente sejam entregues entre eles, pois em caso de falha existem formas de garantir o envio, como reenvio ou então confirmação de recebimento do mesmo.

A arquitetura utilizada para este artigo será melhor especificada na seção 4.

## 3.2 Banco de Dados Mysql

O banco de dados Mysql é atualmente o banco de dados *open source* mais usado no mercado, demonstrando grande maturidade e também um ótimo desempenho. Foi escolhido utilizar este banco devido ao fato de ser o único banco de dados *open source* com características de replicação e clusterização de dados nativo, tendo em vista que os outros bancos estudados também tem essa possibilidade, porém são soluções desenvolvidas por terceiros, sem ter a garantia de funcionamento das mesmas.

Para instalação do banco de dados foram usadas configurações padrões de instalação. A partir disso foram feitas as configurações do cluster de banco de dados conforme mostrado abaixo.

## 3.3 Clusterização de MySQL

MySQL Cluster é um banco de dados de alta disponibilidade construído utilizando uma arquitetura única e interface SQL padrão. O sistema consiste de um número de processos de comunicação, ou nós que podem ser distribuídas através de máquinas para garantir a disponibilidade contínua em caso de falha de servidor ou rede. MySQL Cluster usa um mecanismo de armazenagem, que consiste em um conjunto de dados nós para armazenamento de dados que pode ser acessado através do padrão SQL com o MySQL ou através do NDB API para acesso em tempo real.

A API NDB é uma linguagem de programação para aplicações orientada a objetos para o Cluster Mysql que implementa indexes, busca, transações e tratamentos de eventos.

O cluster de Mysql garante a tolerância a faltas em qualquer nó do cluster, para isso, ele se auto reconfigura para garantir seu funcionamento.

## 4 Arquitetura proposta

Para a garantia a tolerância a faltas de hardware, a solução mais utilizada atualmente é a clusterização ou replicação de hardware, que apesar de ser custosa, pois necessidade redundância de hardwares se mostra eficiente neste ponto.

A arquitetura proposta neste artigo se baseia em clusterização de servidores de aplicação Jboss juntamente com replicação de dados Mysql, conforme mostrado figura abaixo.

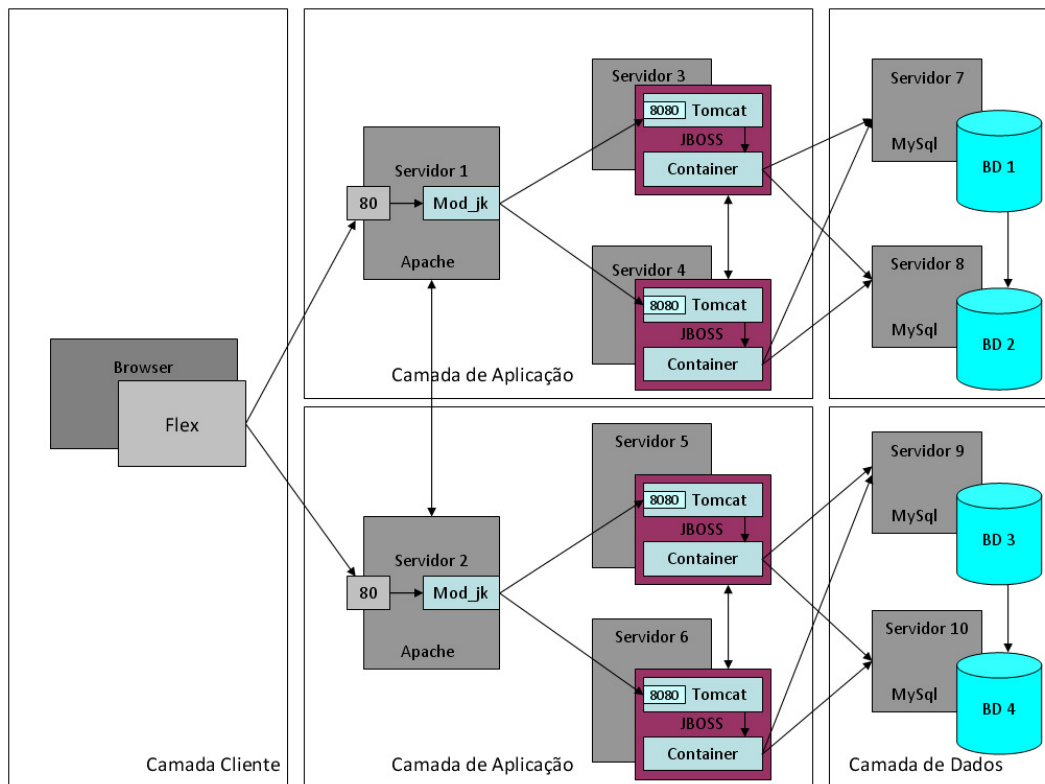


Figura 1: Arquitetura proposta  
Fonte:

Esta arquitetura está dividida em três camadas, que são a camada Cliente, camada de Aplicação e camada de Dados. Cada camada será melhor especificada nas próximas seções.

#### 4.1 Camada Cliente

A **camada de Cliente** consiste na parte aonde usuário final realmente participa, ou seja, onde ficaria o “consumidor” da aplicação, por se tratar de uma aplicação web, o consumidor dela é o próprio browser do usuário, fazendo assim que para acessar a aplicação ele necessite somente uma conexão a internet. A camada de cliente envia sempre dois *requests* para os servidores replicados, para garantir que o mesmo *request* seja recebido pelos *backups*.

#### 4.2 Camada de Aplicação

A **camada de Aplicação** consiste em dois servidores que contém configurado neles o servidor HTTP Apache, o servidor Apache está no mercado desde 1996, e atualmente é o servidor mais estável e mais utilizado, sempre se mantendo atualizado com os padrões HTTP. Juntamente com o servidor Apache existe seu complemento o mod\_jk, que consiste em um complemento usado para se trabalhar com o servidor Apache juntamente com servlets, especialmente servlets Java. O mod\_jk é um conector usado pelo Apache para se conectar ao servidor de aplicação Apache Tomcat usando o protocolo de comunicação AJP. Neste servidor ocorre o *load balancing* das aplicações, ou seja, ele seleciona qual dos servidores irá

retornar a resposta para o *browser* do usuário. O mod\_ajk trabalha com *loading balance* feito por java, aonde ele tem a listagem de servidores disponíveis para acesso e a partir de um algoritmo chamado *Round Robin* ele seleciona o mais apropriado para dar a resposta a esse usuário. Nele também é configurado uma opção chamada *sticky sessions*, que se trata de uma configuração que diz que um request feito por algum browser será sempre respondido pelo mesmo servidor, isso é utilizado para diminuir o *overheading* entre os servidores de aplicação para replicar os dados das sessões dos usuários.

Estes servidores trabalham em cluster utilizando o Jgroups que será melhor explicado abaixo, eles trabalham em cluster de forma que mantém todos os dados de todas as sessões de usuários que estão usando o sistema em todos eles, ou seja, ocorre uma replicação das sessões de usuários, ele procedimento é chamado de *State Replication*.

Os servidores mantém uma lista de todos ips onde estão os outros servidores, e assim ele enviam requisições ips para os outros com os dados necessários referentes as sessões dos usuários. Isso garante que em caso de falha em alguns dos servidores, qualquer outro poderá continuar tratando a sessão do usuário corretamente sem que o mesmo perceba a falha ocorrida.

Além dos servidores com o Apache, na camada de Aplicação contém mais dois servidores com o container Jboss, trabalhando em cluster, comunicando-se entre si. O container Jboss. O jboss contém internamente a ele, o servidor de aplicações Tomcat, também desenvolvido pela Apache, o Tomcat trata-se do servidor de aplicação mais utilizado no mercado em conjunto com o Jboss, pois trabalha de acordo com as especificações W3C, e também trabalha de forma otimizada com o servidor HTTP Apache. Nestes servidores é aonde a aplicação está, ou seja, é neste servidor aonde ocorre o deploy da aplicação.

O cliente final nunca ve este servidor, ele somente se comunica com os servidores aonde está configurado o servidor Apache, e o mesmo redireciona para estes servidores.

#### **4.2.1 Jgroups**

Jgroups é uma ferramenta para envio de mensagens multicast confiável. Ele se baseia na comunicação P2P entre os nodes dos clusters, ele pode ser baseado nos seguintes protocolos de comunicação: UDP, TCP e JMS. A principal característica do mesmo é a garantia de envio das mensagens entre os nodes, em caso de falha de alguma mensagem, ela será reenviada.

Jgroups também trabalha com o grupo, ou seja, ele controla todo o cluster, mantendo sempre uma lista atualizada de quem está conectado ou não no cluster, e quando há entrada de algum novo node, ele avisa a todos os outros para atualizarem sua listagem.

A arquitetura do Jgroups é especificada conforme mostrado abaixo:



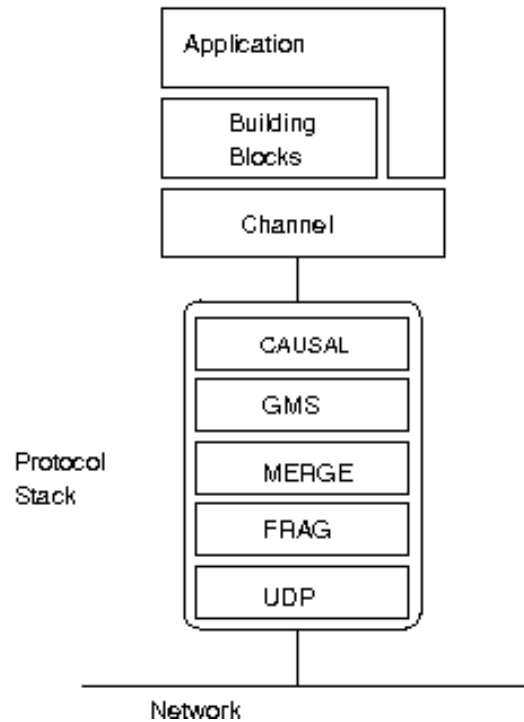


Figura 2: Arquitetura Jgroups

Fonte: <http://www.jgroups.org/manual/html/ch01.html#ArchitectureFig>

A arquitetura consiste em três partes:

1. A API channel de comunicação entre a aplicação e o Jgroups.
2. O Building Blocks que servem para facilitar o uso do Jgroups e dar uma maior abstração para quem usar o framework. Nesta camada também é aonde é implementado os protocolos de garantia de envio das mensagens.
3. E o Protocol Stack, que consiste na implementação da comunicação especificada pela canal.

Jgroups é atualmente o framework mais utilizado para a comunicação confiável entre grupos, utilizado no Jboss, Tomcat, Jetty, entre outros casos de sucesso.

#### 4.3 Camada de Banco de Dados

A **camada de Banco de dados** consiste em mais dois servidores aonde somente roda o banco de dados Mysql. Estes servidores se comunicam entre si de forma master-slave, para garantir a replicação de dados sempre atualizada. Ou seja, é criado um cluster de banco de dados utilizando Mysql.

A camada de Aplicação se comunica com esta camada utilizando JDBC, que se trata de uma interface para conectar Java a banco de dados. A camada de aplicação ao tentar se comunicar com o banco de dados, caso o servidor principal esteja fora do ar, a aplicação se comunica com o proximo servidor de banco de dados do cluster.

## 5 Desenvolvimento

Nos próximos tópicos serão explicados os procedimentos desenvolvidos para o funcionamento da arquitetura que está sendo proposta neste artigo

### 5.1 Configuração da arquitetura proposta

Para configurar os servidores utilizados na arquitetura proposta foram feitos os seguintes passos.

#### 5.1.1 Configurações dos servidores de aplicação(servidor 3, servidor 4, servidor 5, servidor 6)

Os servidores do cluster serão chamados de node 1 e node 2, e neles rodará os servidores de aplicação (jboss), o IP interno do servidor 3 é 192.168.1.159, do servidor 4 é 192.168.1.158, do servidor 5 é 192.168.1.8 e do servidor 6 é 192.168.1.2. Primeiramente para rodar um cluster no servidor jboss é somente necessário inicia-lo com a configuração all que assim ele terá as configurações necessárias para o cluster.

Serão formados dois cluster, um utilizando o servidor 3 e servidor 4 e outro utilizando o servidor 5 e servidor 6, portanto somente essas maquinas se comunicarão entre si, e assim o servidor 3 não terá comunicação com servidor 5 e assim por diante.

Após isso, é necessário configurar os dados referentes a localização, nome e outros dados de cada node do cluster, e para isso, o jboss utiliza o Jgroups, e para a configuração deste cluster foram usadas as seguintes configurações. Estas configurações são feitas no arquivo `JBOSS_HOME\server\all\deploy\cluster-service.xml`.

```

<Config>
  <TCP bind_addr="{jboss.bind.address}" start_port="7800" loopback="true"
    tcp_nodelay="true"
    rcv_buf_size="2000000"
    send_buf_size="640000"
    discard_incompatible_packets="true"
    enable_bundling="false"
    max_bundle_size="64000"
    max_bundle_timeout="30"
    use_incoming_packet_handler="true"
    use_outgoing_packet_handler="false"
    down_thread="false" up_thread="false"
    use_send_queues="false"
    sock_conn_timeout="300"
    skip_suspected_members="true"/>
  <TCPPING initial_hosts="{jboss.bind.address}[7800],192.168.1.153[7800]" port_range="3"
    timeout="3000"
    down_thread="false" up_thread="false"
    num_initial_members="2"/>
  <MERGE2 max_interval="100000"
    down_thread="false" up_thread="false" min_interval="20000"/>
  <FD_SOCKET down_thread="false" up_thread="false"/>
  <FD timeout="10000" max_tries="5" down_thread="false" up_thread="false" shun="true"/>
  <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
  <pbcast.NAKACK max_xmit_size="60000"
    use_mcast_xmit="false" gc_lag="0"
    retransmit_timeout="300,600,1200,2400,4800"
    down_thread="false" up_thread="false"
    discard_delivered_msgs="true"/>
  <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
    down_thread="false" up_thread="false"
    max_bytes="400000"/>
  <pbcast.GMS print_local_addr="true" join_timeout="3000"
    down_thread="false" up_thread="false"
    join_retry_timeout="2000" shun="true"
    view_bundling="true"/>
  <pbcast.STATE_TRANSFER down_thread="false" up_thread="false" use_flush="false"/>
</Config>

```

Figura 3: Configuração do Node 1 do cluster  
 Fonte: Adaptado por ANDRADE, Rafael

Para a aplicação do cluster, foi escolhido a opção de utilizar as configuraçãoe usando TCP, pois apesar de ter uma carga maior na rede, ele garante melhor a entrega dos pacotes entre os servidores.

As principais configurações usadas serão explicadas abaixo:

- TCP - identifica que o sistema usará o protocolo TCP para fazer a comunicação entre os nodes;
- TCPPING – identifica o protocolo usado para fazer a busca dos servidores disponíveis para o Cluster;
- MERGE2 – é usado para que em caso de falha da rede, os coordenadores dos grupos possam restaurar o cluster;
- FD\_SOCKET – identifica o tipo de detecção de falha usado no cluster;
- VERIFY\_SUSPECT – identifica o que leva um cluster a identificar outro como suspeito de estar em falha;
- pbcast.NAKACK – Identifica a forma de garantia de entrega das mensagens usadas no cluster;

- pbcast.GMS – identifica a forma usada para garantir a entrada de novos nodes no cluster;
- pbcast.STABLE – Identifica que está sendo usado um garbage collector para todo cluster, pois se for guardar todas as mensagens enviadas entre os nodes, poderá ocorrer uma falta de memória nos nodes;
- pbcast.STATE\_TRANSFER – identifica que cada node novo que entrar no cluster irá receber o estado do cluster.

E assim o primeiro servidor está pronto para rodar em cluster, faltando somente configurar o segundo servidor do cluster, o node 2.

Para configurar o node 2 do cluster, se utiliza as mesmas configurações do node 1, mudando somente o IP de identificação dos servidores, e assim sua configuração ficará conforme mostrado a seguir:

```
<Config>
  <TCP bind_addr="192.168.1.158" start_port="7800" loopback="true"
    tcp_nodelay="true"
    rcv_buf_size="20000000"
    send_buf_size="640000"
    discard_incompatible_packets="true"
    enable_bundling="false"
    max_bundle_size="64000"
    max_bundle_timeout="30"
    use_incoming_packet_handler="true"
    use_outgoing_packet_handler="false"
    down_thread="false" up_thread="false"
    use_send_queues="false"
    sock_conn_timeout="300"
    skip_suspected_members="true"/>
  <TCPPING initial_hosts="192.168.1.158[7800],192.168.1.159[7800]" port_range="3"
    timeout="3000"
    down_thread="false" up_thread="false"
    num_initial_members="3"/>
  <MERGE2 max_interval="100000"
    down_thread="false" up_thread="false" min_interval="20000"/>
  <FD_SOCKET down_thread="false" up_thread="false"/>
  <FD_TIMEOUT="10000" max_tries="5" down_thread="false" up_thread="false" shun="true"/>
  <VERIFY_SUSPECT timeout="1500" down_thread="false" up_thread="false"/>
  <pbcast.NAKACK max_xmit_size="60000"
    use_mcast_xmit="false" gc_lag="0"
    retransmit_timeout="300,600,1200,2400,4800"
    down_thread="false" up_thread="false"
    discard_delivered_msgs="true"/>
  <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
    down_thread="false" up_thread="false"
    max_bytes="400000"/>
  <pbcast.GMS print_local_addr="true" join_timeout="3000"
    down_thread="false" up_thread="false"
    join_retry_timeout="2000" shun="true"
    view_bundling="true"/>
  <pbcast.STATE_TRANSFER down_thread="false" up_thread="false" use_flush="false"/>
</Config>
```

Figura 4: Configuração do Node 2 do cluster.

Fonte: Adaptado por ANDRADE, Rafael

Com estas configurações citadas acima, estará pronto a configuração do cluster, e ao iniciar os servidores, poderá se verificar ambos se comunicando. A figura a seguir exemplifica essa comunicação:

```

12:37:57,375 INFO [DefaultPartition] Initializing
12:37:57,468 INFO [STDOUT]
-----
GMS: address is 192.168.1.159:7800
-----
12:38:02,453 WARN [ConnectionTable] peer closed connection, trying to re-send m
sg
12:38:02,453 ERROR [ConnectionTable] 2nd attempt to send data failed too
12:38:02,453 INFO [DefaultPartition] Number of cluster members: 2
12:38:02,453 INFO [DefaultPartition] Other members: 1
12:38:02,453 INFO [DefaultPartition] Fetching state (will wait for 30000 millis
econds):
12:38:02,687 INFO [DefaultPartition] state was retrieved successfully (in 234 m
illiseconds)
12:38:02,796 INFO [HANamingService] Started ha-jndi bootstrap jnpPort=1100, bac
klog=50, bindAddress=/0.0.0.0
12:38:02,812 INFO [DetachedHANamingService$AutomaticDiscovery] Listening on /0.
0.0.0:1102, group=230.0.0.4, HA-JNDI address=192.168.1.159:1100
12:38:03,031 INFO [TreeCache] No transaction manager lookup class has been defi
ned. Transactions cannot be used
12:38:05,468 INFO [STDOUT]
-----
GMS: address is 192.168.1.159:2105
-----

```

Figura 5: Exemplo do Cluster para onde está entrando

Fonte: Adaptado por ANDRADE, Rafael

```

12:40:39,718 INFO [DefaultPartition] I am (192.168.1.153:1098) received members
hipChanged event:
12:40:39,718 INFO [DefaultPartition] Dead members: 1 ([192.168.1.159:1098])
12:40:39,718 INFO [DefaultPartition] New Members : 0 ([])
12:40:39,718 INFO [DefaultPartition] All Members : 1 ([192.168.1.153:1098])
12:40:40,031 INFO [TreeCache] viewAccepted(): [192.168.1.153:1040!2] [192.168.1
.153:1040]
12:40:55,093 INFO [TreeCache] viewAccepted(): [192.168.1.153:1040!3] [192.168.1
.153:1040, 192.168.1.159:2131]
12:40:55,281 INFO [TreeCache] locking the subtree at / to transfer state
12:40:55,281 INFO [StateTransferGenerator_140] returning the state for tree roo
ted in /(<1024 bytes)
12:41:02,921 INFO [DefaultPartition] New cluster view for partition DefaultPart
ition (id: 3, delta: 1) : [192.168.1.153:1098, 192.168.1.159:1098]
12:41:02,921 INFO [DefaultPartition] I am (192.168.1.153:1098) received members
hipChanged event:
12:41:02,921 INFO [DefaultPartition] Dead members: 0 ([])
12:41:02,921 INFO [DefaultPartition] New Members : 1 ([192.168.1.159:1098])
12:41:02,921 INFO [DefaultPartition] All Members : 2 ([192.168.1.153:1098, 192.
168.1.159:1098])
12:41:07,156 INFO [TreeCache] viewAccepted(): [192.168.1.153:1053!3] [192.168.1
.153:1053, 192.168.1.159:2153]
12:41:09,531 INFO [TreeCache] viewAccepted(): [192.168.1.153:1057!3] [192.168.1
.153:1057, 192.168.1.159:2158]

```

Figura 6: Exemplo de um coordenador do Cluster

Fonte: Adaptado por ANDRADE, Rafael

Lembrando que para iniciar os servidores é necessário somente ir na pasta `JBOSS_HOME/bin` e executar o comando `run -c all -b 0.0.0.0`.

O comando `-c all` identifica que será rodado o servidor com todas as configurações de cluster e cache, e a identificação `-b 0.0.0.0` se fez necessário para que as máquinas se identifiquem na rede, sem isso o cluster não se “encontra”.

Para adicionar novas aplicações no cluster, a melhor forma de fazer é usar o Farm Deploy do Jboss, pois ele garante que todos os sistemas rodando em todos os nós do cluster serão iguais. Ele funciona de forma que ao colocar uma aplicação na pasta `JBOSS_HOME/server/all/farm`, e assim ele inicializará a aplicação e também enviara esta

aplicação para todos nodes do cluster, para que assim o outros nodes também tenham a aplicação.

E assim o clusterização de servidores estará pronta. As configurações mostradas se repetem para o servidor 5 e servidor 6, por isso nao foram mostradas.

### 5.1.2 Configuração dos servidores de Banco de dados

Para configurar a replicação de dados nos banco de dados, foram feitos alguns passos, que serão explicados abaixo.

#### Configuração do Master

Ao iniciar a configuração de replicacao de dados no mysql, é necessário primeiramente configurar o servidor que será considerado o “master”, ou seja, todas as requisições da aplicação serão enviados para ele, e ele repassará essas requisições para os bancos de dados “slaves”. Devido um dos requisitos inicialmente especificados para o sistema, todos servidores deverão ser Windows, por isso as configurações mostradas são para servidores Windows, tendo alguma diferença para servidores usando Linux.

Por convenção, vamos chamar a pasta aonde foi instalado o Mysql de MYSQL\_HOME.

Assim, inicia-se alterando o arquivo my.ini, que fica no diretório MYSQL\_HOME\MySQL Server 5.1\my.ini, neste arquivo estão as principais configurações do banco de dados, como o tipo de dados que serão usado, charset, entre outros. Inicialmente é necessário configurar o banco para que use a rede, portanto se fez necessário retirar as seguintes linhas da configuração:

```
#skip-networking
#bind-address = 127.0.0.1
```

Figura 7: Configurando o networking  
Fonte: Adaptado por ANDRADE, Rafael

Após isso, é necessário dizer ao banco qual database ele deverá ter logs (para posteriormente serem lidos pelos servidores slaves). Como neste trabalho desejamos configurar a replicação para a base “direseg”, as configurações de log ficaram como mostra a imagem abaixo:

```
Log-bin = C:/mysql-bin.log
Binlog-do-db=direseg
Server-id=1
```

Após alterar estas configurações, é necessário reiniciar o banco de dados, para que assim ele comece a utilizar o log. Seguido a estas configurações, é necessário obter as informações sobre o arquivo de log usado pelo banco de dados “master”, para poder posteriormente configurar os “slaves”, e isso se faz usando os seguintes comando no mysql:

```
USE direseg;
FLUSH TABLES WITH READ LOCK;
SHOW MASTER STATUS;
```

Ao executar isto, foi obtido a seguinte resposta

```

Microsoft Windows XP [versão 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\rafael>mysql -u root -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 4
Server version: 5.1.34-community-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File           | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+-----+-----+-----+-----+
| mysql-bin.000002 |      106 | direseg      |                   |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> _

```

Figura 8: Exemplo de resposta do Mysql  
 Fonte: Adaptado por ANDRADE, Rafael

É importante anotar os dados que aparecem, pois eles deverão ser usados posteriormente para configurar os slaves.

Após isto, o banco de dados “master” da replicação está pronto. Agora falta somente configurar os slaves.

## Configuração dos Slaves

Para configurar os “slaves”, primeiramente é necessário criar o database direseg nos mesmo, e após isso, é necessário alterar as configurações do arquivo my.ini para informar os dados referentes a replicação e localização do servidor “máster”. A figura abaixo ilustrar as configurações usadas no servidor “slave”.

```

server-id=2
master-host=192.168.1.153
master-user=root
master-password=j expadm
master-connect-retry=60
replicate-do-db=direseg

```

Figura 9: Configuração do banco de dados “slave”  
 Fonte: Adaptado por ANDRADE, Rafael

Após aplicar as configurações, é necessário reiniciar o banco de dados. E assim, para que o “slave” esteja em conformidade com o “máster”, se utiliza o seguinte comando:

```
LOAD DATA FROM MASTER;
```

E assim, o “slave” terá todos os dados do banco “master”, dados somente do schema “direseg”, pois é o único que interessa para no momento.

Após isso, é necessário para a thread Slave do banco de dados, para atualizar as informações referentes ao arquivo de log e posição do arquivo do máster (os dados obtidos anteriormente com o comando SHOW MASTER STATUS;), e para isso foi executado o comando:

```
SLAVE STOP;
```

E logo após o comando:

```
CHANGE MASTER TO MASTER_HOST='192.168.1.153', MASTER_USER='root',  
MASTER_PASSWORD='jexpadm', MASTER_LOG_FILE='mysql-bin.000002',  
MASTER_LOG_POS=106;
```

Este comando adiciona ou altera as configurações dos referentes ao banco de dados máster. E as variáveis usadas são:

- MASTER\_HOST – IP da máquina que possui o banco de dados máster
- MASTER\_USER – usuário que está na máquina máster, que será usado para obter os dados.
- MASTER\_PASSWORD – senha do usuário.
- MASTER\_LOG\_FILE – qual arquivo está sendo usado pelo máster para salvar todas as transações feitas nele.
- MASTER\_LOG\_POS – posição a qual o arquivo será lido para manter sempre atualizado.

Com estas configurações prontas, é apenas necessário executar o comando:

```
START SLAVE;
```

E assim a replicação entre os servidores está pronta, todas requisições e transações que ocorrerem no banco de dados máster será replicada para os slaves. Lembrando que para fazer a replicação, é necessário uma rede local, e a porta 3306 deverá estar liberada no firewall, pois o firewall do Windows bloqueia esta porta para conexões externas.

E com estas configurações ficará pronto a replicação de dados usando Mysql. Lembrando que as configurações se repetem nos dois nós do cluster, alterando somente o ip das máquinas usadas.

## 5.2 Avaliação de Desempenho

Visando verificar o desempenho da infra-estrutura proposta foram executados testes em uma rede local de 10Mbps compostas por computadores Intel Dual Core de 1.6 GHz com 2Gb de memória RAM e sistema operacional Microsoft Windows XP. Os computadores foram configurados conforme mostrado na arquitetura proposta mostrada anteriormente.

A primeira análise de desempenho feita nesta arquitetura foi o tempo de resposta a uma requisição enviada ao servidor comparada com o número de serviços disponíveis no cluster. A mensagem enviada para o servidor foi uma requisição HTTP, e a resposta do servidor é sempre uma resposta de contém 0 bytes, tendo somente os bytes do protocolo HTTP. De acordo com o gráfico abaixo, quando se tem somente um serviço o tempo de resposta do servidor é de 40ms(milisegundos), e quando se coloca mais um serviço no cluster, o número praticamente dobra, passando a ~70ms, e assim se percebe um crescimento com o número de serviços. Os testes foram executados com mensagens de tamanho de 0-256k-512k bytes, e os resultados completos estão mais especificados abaixo.



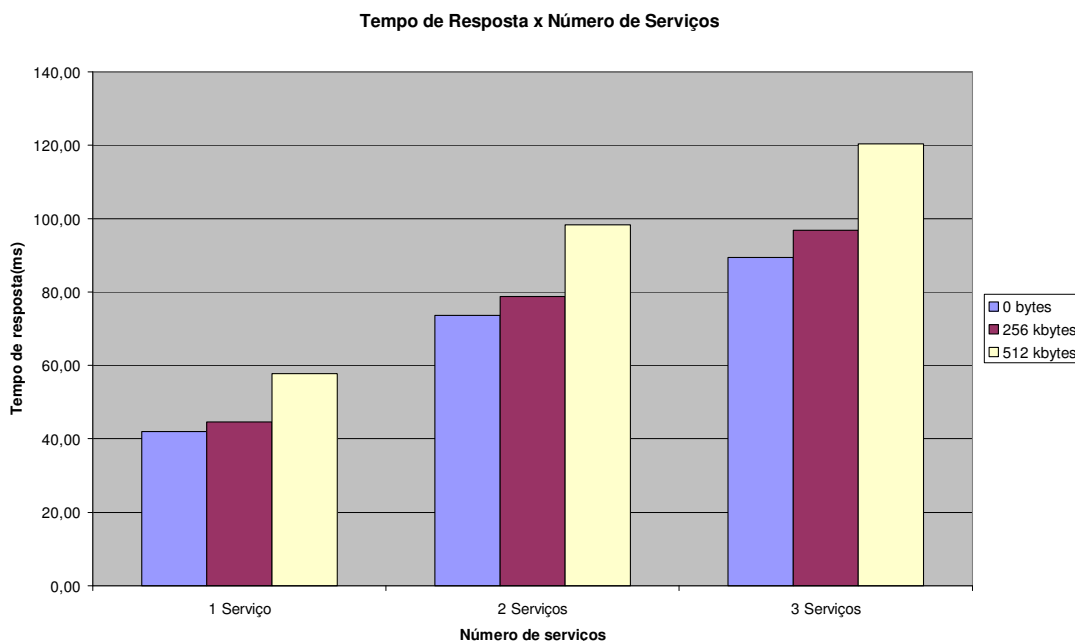


Figura 10: Desempenho do serviço.  
 Fonte: Adaptado por Andrade, Rafael

Outra análise de desempenho feita foi a de tempo de resposta considerando o número de clientes fazendo a requisição ao servidor. A mensagem a ser enviada foi de 512Kbytes, que seria a simulação de uma listagem no servidor. O número de cliente variou entre 10 e 80 clientes. Também foram feitos de acordo com o número de serviços disponíveis. Para fazer estes testes, foi implementado um client em Java, que cria threads de disparo de requisições para o servidor, cada thread envia 100 requisições ao servidor, e após isso é calculado o tempo médio de resposta entre essas requisições. O unico problema identifica nesta abordagem é que o java tem um tempo de resposta maior do que o Flex, por isso os dados ficaram acima do que realmente seria calculado usando clients em Flex. Os resultado obtidos estão mostrados no gráfico abaixo

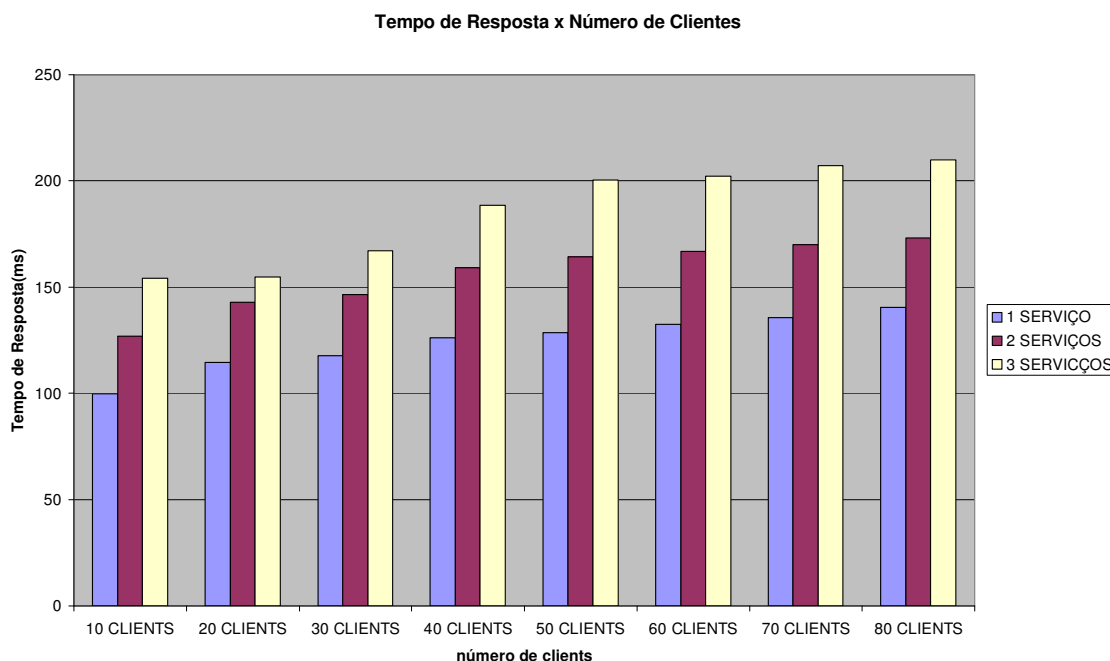


Figura 10: Desempenho do serviço x Número de Clientes.

Fonte: Adaptado por Andrade, Rafael

Como pode ser visto no gráfico, o tempo de resposta aumenta de forma estável de acordo com o número de clientes enviando requisições, pode-se comprovar uma média de aumento de cerca de 15% no tempo de resposta para cada aumento de 10 Clientes enviando requisições, o que é considerado um desempenho muito bom para aplicações web. Sendo que uma requisição chegando até 80ms é praticamente imperceptível para o ser humano, e o maior tempo de resposta obtido é ~200ms.

Não é possível calcular especificamente o overhead causado para detector de falhas usado no servidor Jboss, mas de acordo com o que foi especificado na documentação do próprio servidor, o overhead causado pelos detectores de falhas é praticamente nulo, aumentando de 2 a 5 por cento somente o tempo de resposta a uma requisição.

## 6 Conclusão

Este artigo mostrou uma proposta de arquitetura para se garantir a maior disponibilidade em aplicações Web 2.0, visto que atualmente, os usuários estão cobrando cada vez mais qualidade nos serviços oferecidos a eles, garantindo seu alto desempenho assim como a sua disponibilidade. A arquitetura proposta garante as falhas de hardware que possam acontecer no servidor, assim como também pode garantir as falhas de programação da própria aplicação, colocando-se aplicações programadas de formas diferentes em servidores diferentes.

De acordo com os testes executados pode-se perceber que a arquitetura apesar de usar bastante replicação de dados, se mostrou com um alto desempenho, garantindo assim sua qualidade para a Web 2.0 que cada vez mais necessita altos desempenhos para as aplicações.

Um dos maiores problemas encontrados nessa arquitetura é o custo de implantação da mesma, pois para garantir sua funcionalidade real é necessário muitos servidores diferentes.

## Referências Bibliográficas

WIKIPEDIA. Disponível em: [http://en.wikipedia.org/wiki/Replication\\_\(computer\\_science\)](http://en.wikipedia.org/wiki/Replication_(computer_science)). Acesso em: 12 jul. 2008.

CLUSTER BANCO DE DADOS. Disponível em:  
<http://www.docs.mussicorp.net/guiacuster/clusterbancodedados.php>. Acesso em: 12 jul. 2008

REPLICATION AND DATABASE MIRRORING. Disponível em: <http://msdn.microsoft.com/en-us/library/ms151799.aspx>. Acesso em: 22 jul. 2009.

DISTRIBUTED DATABASE. Disponível em: [http://en.wikipedia.org/wiki/Distributed\\_database](http://en.wikipedia.org/wiki/Distributed_database). Acesso em 15 ago. 2008

Oracle9i Database Administrator's Guide, Disponível em:  
[http://www.cs.uvm.edu/oracle9doc/server.901/a90117/ds\\_conce.htm](http://www.cs.uvm.edu/oracle9doc/server.901/a90117/ds_conce.htm), Acesso em: 22 set. 2008

Oracle® Database Advanced Replication 10g Release 1 (10.1). Disponível em:  
<http://stanford.edu/dept/itss/docs/oracle/10g/server.101/b10732/toc.htm>. Acesso em: 30 set. 2008

Oracle Data Replication and Integration. Disponível em:  
<http://www.oracle.com/technology/products/dataint/index.html>. Acesso em: 23 abr. 2009.

Oracle Replication. Disponível em:  
[http://searchoracle.techtarget.com/news/article/0,289142,sid41\\_gci905154,00.html?Exclusive=True](http://searchoracle.techtarget.com/news/article/0,289142,sid41_gci905154,00.html?Exclusive=True). Acesso em 23 abr. 2009.

Database Oracle Advanced Replication Tutorial. Disponível em: <http://www.roseindia.net/software-tutorials/detail/13572>. Acesso em: 23 abr. 2009.

Do It Yourself (DIY) Oracle replication. Disponível em: <http://it.toolbox.com/blogs/data-ruminations/do-it-yourself-diy-oracle-replication-12894>. Acesso em: 23 abr. 2009-10-26

Ian Gilfillan. Database Replication in Mysql. Disponível em:  
<http://www.databasejournal.com/features/mysql/article.php/3355201/Database-Replication-in-MYSQL.htm>. Acesso em 28 abr. 2009.

Step-by-Step how to Setup MYSQL Database Replication. Disponível em:  
<http://aciddrop.com/2008/01/10/step-by-step-how-to-setup-mysql-database-replication/>. Acesso em: 28 abr. 2009.

Mysql Replication. Disponível em: <http://dev.mysql.com/doc/refman/5.0/en/replication.html>. Acesso em: 28 abr. 2009.

Falko Timme. How To Set Up Database Replication In Mysql. Disponível em:  
[http://www.howtoforge.com/mysql\\_database\\_replication](http://www.howtoforge.com/mysql_database_replication). Acesso em: 28 abr. 2009.

Daniel. Setting up database replication on Mysql. Disponível em: <http://www.gra2.com/article.php/setting-up-database-replication-on-mysql>. Acesso em: 30 abr. 2009.

Database Replication In Mysql. Disponível em: <http://librenix.com/?inode=8118>. Acesso em: 30 abr. 2009.

Slony-I 1.0.5 (PostgreSQL Database Replication). Disponível em:  
<http://www.postgresql.org/about/news.233>. Acesso em: 30 abr. 2009.

PostgreSQL - Replication and High Availability. Disponível em: <http://edoceo.com/liber/db-postgresql-replication>. Acesso em: 30 abr. 2009.

PGCluster. Disponível em: <http://pgcluster.projects.postgresql.org/>. Acesso em: 30 abr. 2009.

Slony-I. Disponível em: <http://slony1.projects.postgresql.org/>. Acesso em: 30 abr. 2009.

DBBalancer. Disponível em: <http://sourceforge.net/projects/dbbalancer/>. Acesso em 30 abr. 2009.

PgPool. Disponível em: <http://pgpool.projects.postgresql.org/>. Acesso em 30 abr. 2009.

PostgreSQL table comparator. Disponível em: <http://pg-comparator.projects.postgresql.org/>. Acesso em 30 abr. 2009.

Database replication under Postgres. Disponível em: <http://archives.postgresql.org/pgsql-interfaces/1999-12/msg00191.php>. Acesso em 5 mai. 2009.

Database Replication. Disponível em: <http://www.cs.mcgill.ca/~kemme/disl/replication.html>, Acesso em: 5 mai. 2009.

PostgreSQL + Replication. Disponível em: <http://www.commandprompt.com/products/mammothreplicator/>. Acesso em: 5 mai. 2009.

Irina Sourikova. PostgreSQL Replicator. Disponível em: [http://www.google.com.br/url?sa=t&source=web&ct=res&cd=9&url=http%3A%2F%2Fwww.rhic.bnl.gov%2FRCF%2FUserInfo%2FMeetings%2FTechnology%2FIRina\\_Sourikova\\_techMeet.ppt&ei=U70YSfuzIIyi8gTM1sCfCw&usg=AFQjCNHdOqqGew5a31NxDnRS77YI8DMx2w&sig2=pmHi-11Ra4vvvNA5e7moAA](http://www.google.com.br/url?sa=t&source=web&ct=res&cd=9&url=http%3A%2F%2Fwww.rhic.bnl.gov%2FRCF%2FUserInfo%2FMeetings%2FTechnology%2FIRina_Sourikova_techMeet.ppt&ei=U70YSfuzIIyi8gTM1sCfCw&usg=AFQjCNHdOqqGew5a31NxDnRS77YI8DMx2w&sig2=pmHi-11Ra4vvvNA5e7moAA). Acesso em: 10 mai. 2009.

Fault-Tolerant system. Disponível em: [http://en.wikipedia.org/wiki/Fault-tolerant\\_system](http://en.wikipedia.org/wiki/Fault-tolerant_system). Acesso em: 12 mai. 2009.

Walter Heimerdinger , Charles B. Weinstock. Conceptual Framework for System Fault Tolerance, A. Disponível em: <http://www.sei.cmu.edu/library/abstracts/reports/92tr033.cfm>. Acesso em: 12 mai. 2009.

Fault Tolerance Concepts. Disponível em: [http://hissa.nist.gov/chissa/SEI\\_Framework/framework\\_8.html](http://hissa.nist.gov/chissa/SEI_Framework/framework_8.html). Acesso em: 12 mai. 2009.

Pankaj Jalote. Fault Tolerance in Distributed Systems. 1Ed. 2006.

Taisy Weber. Conceitos de Dependabilidade. Disponível em: <http://www.inf.ufrgs.br/~taisy/disciplinas/textos/ConceitosDependabilidade.PDF>. Acesso em: 12 mai. 2009.

Jboss 4 Clustering Guide. Disponível em: <http://www.google.com.br/url?sa=t&source=web&ct=res&cd=2&url=http%3A%2F%2Fdocs.jboss.org%2Fjbossas%2Fguides%2Fclusteringguide%2Fr2%2Fen%2Fpdf%2Fjboss4-clustering.pdf&ei=Db8YSeeYKaCm8ASpvtmTCw&usg=AFQjCNGhTq0da0et4y107ANECQHxC1XHmQ&sig2=Lv0DMdrTd710-hUuxUz14w>. Acesso em: 22 mai. 2009.

Jboss Clustering. Disponível em: <http://docs.jboss.org/jbossas/jboss4guide/r4/html/cluster.chapt.html>. Acesso em: 22 mai. 2009.

Ivelin Ivanov. J2EE Clustering with Jboss. Disponível em: [http://onjava.com/pub/a/onjava/2003/08/20/jboss\\_clustering.html?page=1](http://onjava.com/pub/a/onjava/2003/08/20/jboss_clustering.html?page=1). Acesso em: 26 mai. 2009.

Samson Kittoli. JBoss AS5 Clustering Guide. Disponível em: <http://www.jboss.org/community/wiki/jbossas5clusteringguide>. Acesso em: 26 mai. 2009.

Brian Stansberry, Galder Zamarreno. JBoss Application Server Clustering Guide. Disponível em: <http://www.jboss.org/file->

[access/default/members/jbossas/freezone/docs/Clustering\\_Guide/beta422/html/index.html](http://access/default/members/jbossas/freezone/docs/Clustering_Guide/beta422/html/index.html). Acesso em: 26 mai. 2009.

Web 2.0. Disponível em: [http://pt.wikipedia.org/wiki/Web\\_2.0](http://pt.wikipedia.org/wiki/Web_2.0). Acesso em: 30 ago. 2009.

Tim O'Reilly. What Is Web 2.0. Disponível em: <http://oreilly.com/web2/archive/what-is-web-20.html>. Acesso em: 30 ago. 2009.