

pgGrid: Uma implementação de fragmentação de dados para o pgcluster

Gustavo Tonini

20 de junho de 2009

Trabalho sob o título *pgGrid: Uma implementação de fragmentação de dados para o pgCluster*, defendido por Gustavo Tonini e aprovado para obtenção do grau de Bacharel em Ciências da Computação em 28 de maio de 2009, em Florianópolis, Estado de Santa Catarina, pela banca examinadora constituída pelos doutores:

Prof. Dr. Frank Siqueira
Orientador

Prof. Dr. Ronaldo Mello

Prof. Dr. Renato Fileto

Gustavo Tonini

pgGrid: Uma implementação de fragmentação de dados para o pgcluster

Florianópolis

Maio de 2009

Gustavo Tonini

pgGrid: Uma implementação de fragmentação de dados para o pgcluster

Orientador:
Frank Siqueira

UNIVERSIDADE FEDERAL DE SANTA CATARINA

Florianópolis
Maio de 2009

Sumário

Lista de abreviaturas e siglas

Lista de Figuras

Resumo

Abstract

1	Introdução	p. 10
1.1	Objetivo geral	p. 10
1.2	Objetivos específicos	p. 11
1.3	Justificativa	p. 11
1.4	Metodologia	p. 12
1.5	Organização do texto	p. 12
2	Sistemas distribuídos	p. 13
2.1	Características	p. 13
2.1.1	Replicação e confiabilidade	p. 13
2.1.2	Escalabilidade	p. 14
2.1.3	Disponibilidade	p. 14
2.2	Arquiteturas	p. 14
3	Bancos de dados distribuídos	p. 16
3.1	Modelos arquiteturais para BDD	p. 16
3.1.1	Autonomia	p. 16

3.1.2	Distribuição	p. 16
3.1.3	Heterogeneidade	p. 17
3.2	Arquiteturas	p. 17
3.2.1	Cliente/servidor	p. 17
3.2.2	<i>Peer to peer</i> (P2P)	p. 18
3.3	Classificação	p. 18
3.3.1	Mestre/escravo (<i>Master-slave</i>)	p. 19
3.3.2	Multimaster	p. 19
3.3.3	Sincronismo e assincronismo	p. 19
3.3.4	Replicação total ou parcial	p. 20
3.4	Tipos de fragmentação	p. 20
3.4.1	Fragmentação horizontal	p. 20
3.4.2	Fragmentação vertical	p. 21
3.4.3	Fragmentação híbrida	p. 21
3.5	Alocação dos fragmentos	p. 21
3.6	Arquiteturas para integração dos dados	p. 22
3.7	Regras de uma fragmentação correta	p. 22
3.8	Controle distribuído de concorrência	p. 23
3.8.1	O algoritmo MVCC	p. 23
3.9	Protocolos de confiabilidade distribuída	p. 24
3.9.1	O protocolo 2PC	p. 24
3.10	Sistema gerenciador de bancos de dados distribuídos	p. 25
4	Exemplos de SGBDs distribuídos	p. 27
4.1	PostgreSQL e pgCluster	p. 27
4.2	DRDA - Uma arquitetura distribuída para BD relacionais	p. 28
4.3	Implementação do DRDA no SGBD DB2	p. 29

4.4	MySQL Cluster	p. 30
4.5	Oracle Real Application Clusters (RAC)	p. 32
4.5.1	Cache Fusion	p. 32
4.6	Comparativo das tecnologias	p. 33
5	Projeto do pgGrid	p. 34
5.1	Adição da extensão à gramática SQL	p. 34
5.2	Modificação nos catálogos e <i>bootstrap</i>	p. 36
5.2.1	O catálogo de sites (<i>pg_grid_site</i>)	p. 37
5.2.2	Os catálogo de fragmentos (<i>pg_grid_fragment</i>)	p. 37
5.2.3	O catálogo de alocação dos fragmentos (<i>pg_grid_allocation</i>)	p. 37
5.3	Restrições semânticas	p. 38
5.4	Processamento dos comandos de definição do <i>grid</i>	p. 38
5.4.1	Processamento do comando <i>PLACE</i>	p. 39
5.5	Localização dos dados distribuídos	p. 39
5.5.1	Redução para fragmentação horizontal	p. 39
5.5.2	Redução para fragmentação vertical	p. 41
5.6	Processamento das consultas distribuídas	p. 41
5.7	Distribuição dos dados nas transações	p. 42
5.7.1	Índices	p. 42
6	Implementação do pgGrid	p. 44
6.1	Alteração dos catálogos do sistema	p. 44
6.2	Alteração da gramática do pgCluster	p. 45
6.3	Alterações nos comandos de manipulação de dados	p. 45
6.4	Processamento das consultas distribuídas	p. 46
6.5	Módulos alterados	p. 47

6.6	Comparativo dos resultados obtidos	p. 49
6.7	Repositório do projeto	p. 50
6.8	Projetos relacionados	p. 50
7	Caso de uso	p. 51
7.1	Inicialização e configuração dos sites	p. 52
7.2	Definição do esquema de dados	p. 53
7.3	Definição e alocação dos fragmentos	p. 53
7.4	Inserção dos dados	p. 54
7.5	Testes das consultas	p. 55
7.6	Testes de desempenho	p. 56
8	Conclusão e trabalhos futuros	p. 62
8.1	Trabalhos futuros	p. 62
	Referências Bibliográficas	p. 64
	Apêndice A – Patch para o pgCluster	p. 66
	Apêndice B – Artigo	p. 107

Lista de abreviaturas e siglas

2PC	Two phase commit (Efetivação em duas fases)
2PL	Two phase locking (Bloqueio em duas fases)
BD	Banco de dados
BDD	Banco de dados distribuído
DBA	Database administrator (Administrador de banco de dados)
DDL	Data definition language (linguagem de definição de dados)
DRDA	Distributed Relational Database Architecture (arquitetura de bancos de dados relacionais distribuídos)
ECG	Esquema conceitual global
ECL	Esquema conceitual local
GNU	GNU's not Unix
GPL	GNU public license (Licença pública GNU)
MVCC	Multi-version concurrent control (controle de concorrência baseado em várias versões)
PG	PostgreSQL
RAC	Real application clusters
SGBD	Sistema gerenciador de banco de dados
SGBDD	Sistema gerenciador de banco de dados distribuído
TCP	Transmission control protocol
TO	Timestamp ordering (Ordenação por tempo)

Lista de Figuras

3.1	Comunicação não-hierárquica (a). Comunicação hierárquica (b). Fonte: (TANENBAUM, 2008, p. 102)	p. 19
3.2	Fluxograma do 2PC. Fonte: (OZSU, 1999, p. 383)	p. 26
5.1	DER dos novos catálogos	p. 38
6.1	Exemplo de árvore de consulta otimizada	p. 47
6.2	Módulos alterados do PostgreSQL	p. 48
7.1	Modelo ER referente ao caso de uso do pgGrid	p. 51
7.2	Medição dos tempos das operações	p. 59
7.3	Quantidade de pacotes de rede transmitidos durante inserção	p. 60
7.4	Quantidade de pacotes de rede transmitidos durante a consulta distribuída	p. 60
7.5	Espaço em disco médio ocupado pelo banco de dados	p. 61
7.6	Carga de trabalho média dos servidores durante a consulta	p. 61

Resumo

À medida que as organizações crescem, também cresce a necessidade de armazenar grandes massas de dados e organizá-los de uma forma que favoreça sua recuperação. A proposta deste trabalho é oferecer uma extensão ao SGBD PostgreSQL que permita a fragmentação dos dados para que os mesmos sejam distribuídos da forma mais conveniente nos servidores de banco de dados que compõem o conjunto, além de gerenciar sua replicação. Para isso será necessário adicionar os catálogos que definem o esquema da fragmentação e modificar a ferramenta "pgcluster" para gerenciar a localização dos dados e otimizar as consultas. Além disso será proposta uma extensão à linguagem DDL para a definição dos parâmetros da distribuição dos dados e dos "sítios" que formam o sistema de banco de dados distribuído. Todas as ferramentas e metodologias utilizadas no trabalho e em sua respectiva execução fazem parte do ambiente de trabalho dos desenvolvedores do PostgreSQL e serão descritas de acordo com sua relevância.

Palavras-chave: banco de dados, fragmentação, sistemas distribuídos, redes

Abstract

As an organization grows, so it needs to store great amounts of data and to organize them in such a way that favors its recovery. The proposal of this project is to offer an extension to the PostgreSQL DBMS that allows the spalling/fragmentation of data, so that they are distributed in the most convenient form into the sites that compose the distributed database system, and to manage the replication of such data. For this it will be necessary to add the catalogues that define the fragmentation schema and modify the "pgcluster" tool, to manage data localization and to optimize queries. Moreover, an extension to the DDL will be proposed, for the definition of the data distribution parameters and the sites that compose the distributed database system. All the tools and methodologies used in this project and its respective execution are part of the work environment of the PostgreSQL developers and will be described in accordance with its relevance.

Keywords: database, fragmentation, distributed systems, computer networks

1 *Introdução*

Nos primórdios da computação, todo processamento era realizado de forma centralizada. Com o passar dos anos foi-se notando o potencial que as características dos sistemas distribuídos proporcionam. Sistemas de banco de dados distribuídos são sistemas distribuídos que armazenam, manipulam e organizam dados.

Muito estudo já foi realizado nesta área, principalmente no que diz respeito ao gerenciamento de transações e integridade dos dados em sistemas deste tipo. No entanto, existem poucas implementações sendo utilizadas em aplicações reais.

De acordo com Ozsu (1999), as principais promessas e características dos sistemas de banco de dados distribuídos são:

- Administração transparente dos dados fragmentados nas máquinas que compõem o sistema
- Confiabilidade nas transações distribuídas
- Melhoria no desempenho das consultas, através da paralelização das mesmas e outras estratégias de otimização
- Fácil expansão e boa escalabilidade do sistema

Baseando-se nestas características, nos padrões e algoritmos clássicos de gerenciamento este trabalho visa construir uma especificação formal da proposta do SBBD, e implementá-la como uma extensão do SGBD PostgreSQL (POSTGRESQL, 2007).

1.1 **Objetivo geral**

O principal objetivo do trabalho é produzir uma *extensão* para o PostgreSQL que adicione as funções de definição do esquema de BDD, fragmentação, replicação parcial e distribuição de dados nas máquinas que compõem o mesmo.

1.2 Objetivos específicos

Os objetivos específicos são os requisitos adicionais necessários para facilitar a tarefa e para clarificar a utilização do protótipo implementado. Os principais objetivos específicos são:

- Enumerar os algoritmos clássicos envolvidos no processo de distribuição e manutenção dos dados replicados
- Realizar um estudo detalhado do código-fonte atual do PostgreSQL e de sua metodologia de desenvolvimento
- Estudar os módulos do SGBD que serão afetados pelo *patch*
- Implementar as alterações necessárias no código-fonte do pgCluster
- Escrever testes de unidade e de regressão para o código-fonte produzido de acordo com a especificação metodológica dos desenvolvedores
- Coletar a opinião dos desenvolvedores sobre o material produzido
- Escrever o manual de usuário do *patch*

1.3 Justificativa

Muitas ferramentas de distribuição de dados estão disponíveis no mercado. Tais ferramentas realizam vários tipos de distribuição dos dados, tais como: horizontal, vertical e híbrida. No entanto, a maioria delas possuem sua área de atuação externa ao SGBD impossibilitando, desta forma, um controle transacional mais rígido e abrangente. O texto a seguir ressalta a importância da fragmentação dos dados para a melhoria do tempo de resposta nos sistemas de banco de dados:

A decomposição de relações em fragmentos permite que várias transações sejam executadas de forma concorrente. Além disso, a fragmentação de relações geralmente resulta na execução paralela de uma única consulta em vários fragmentos. Logo, isso aumenta o nível de concorrência e, portanto melhora o tempo de resposta do sistema (OZSU, 1999, p. 107).

Este trabalho justifica-se pela melhoria de desempenho que a solução propõe e pela falta de SGBDD completos no mercado.

1.4 Metodologia

A pesquisa teve como base o estudo detalhado dos conceitos-chave dos sistemas de banco de dados distribuídos, juntamente com dedicação de um período para discussão da implementação com alguns colaboradores do projeto PostgreSQL/pgCluster, a qual propiciou a determinação dos caminhos coesos para a conclusão do trabalho. A discussão e a colaboração são os alicerces do desenvolvimento de software livre, portanto isto ocorreu muito no período de produção.

O código da *extensão* teve como base o código original do PostgreSQL e do pgCluster (disponíveis no cvs) que são escritos em linguagem C. Isso é possível devido ao fato de ambos os projetos possuírem seu código-fonte aberto. Muitos outros projetos possuem o código-fonte do PostgreSQL como base, tais projetos podem ser encontrados em www.pgfoundry.org.

1.5 Organização do texto

O trabalho está dividido em duas partes principais: revisão bibliográfica (capítulos 2 a 4) e projeto/implementação das funções propostas (capítulos 5 a 8). Os primeiros capítulos descrevem todos os conceitos relacionados ao trabalho. Depois é apresentado um estudo das implementações existentes na área de BDD, e então dá-se início à segunda parte.

Os capítulos referentes à implementação das funções propostas fornecem as informações necessárias sobre os problemas resolvidos pela proposta e uma descrição detalhada das soluções.

2 *Sistemas distribuídos*

Este capítulo define a terminologia básica e os conceitos fundamentais dos sistemas distribuídos e suas arquiteturas que são a base para o desenvolvimento do protótipo proposto por este trabalho.

Um sistema distribuído é um sistema consistindo de um conjunto de máquinas autônomas conectadas por redes de comunicação e equipadas com softwares que produzem um ambiente de computação integrado e consistente. Sistemas distribuídos permitem que os usuários cooperem de forma mais efetiva e eficiente. (JIA, 2005 apud COULOURIS, 2005, p. 1)

O ambiente distribuído fornece bons resultados no que se refere a confiabilidade e escalabilidade, permitindo, desta forma, que o grupo de usuários interopere de maneira homogênea, sem distúrbios causados por diferenças no desempenho das máquinas clientes, como ocorre nos sistemas centralizados.

Define-se um “cluster” como o conjunto de computadores (também denominados sítios) que cooperam para executar um serviço ou alcançar um objetivo comum.

2.1 **Características**

Esta seção trata de algumas características relevantes dos sistemas distribuídos inerentes aos sistemas de bancos de dados.

2.1.1 **Replicação e confiabilidade**

Sistemas distribuídos que visam a tolerância a falhas devido ao fato de alguma informação não estar disponível em algum sítio implementam redundância dos dados. Assim, existe a necessidade de um controle mais apurado no que se refere à sincronia dos sítios e o tratamento dos conflitos gerados.

Para tolerar vários tipos de falhas, a maior parte das técnicas de tolerância utilizam a adição de componentes redundantes ao sistema. Replicação é a tecnologia que permite a tolerância a falhas em um sistema de banco de dados distribuído (JIA, 2005, p. 312).

O tratamento dos conflitos gerados pelo acesso simultâneo ao mesmo bloco de informações em sistemas de dados replicados é uma área que é muito estudada na computação. Atualmente é possível implementar mecanismos de chaveamento (*locking*) de forma eficiente e eficaz, mesmo se tratando de um ambiente distribuído complexo.

2.1.2 Escalabilidade

Um sistema distribuído executando em um pequeno número de máquinas pode ser facilmente estendido para um grande número de máquinas para aumentar o poder de processamento (JIA, 2005, p. 312).

Uma grande parte das tarefas submetidas a um sistema distribuído pode ser facilmente paralelizada, ou seja, dividida em partes onde cada uma delas é submetida a uma máquina que gera os resultados, os quais são concatenados formando o resultado final do processamento. Desta forma, à medida que o número de máquinas cresce, também cresce a capacidade de processar mais tarefas.

Um sistema distribuído é dito escalável quando o número de unidades de processamento está diretamente relacionado à capacidade de processamento do conjunto.

2.1.3 Disponibilidade

A disponibilidade de serviço em um sistema distribuído está diretamente relacionada aos serviços de replicação do mesmo.

Para um serviço não-replicado, se o servidor cai por algum motivo, o serviço se torna indisponível. Com um sistema de servidores replicados, alta disponibilidade pode ser alcançada porque o software/dados estão disponíveis em vários sítios. (JIA, 2005, p. 312)

2.2 Arquiteturas

Computação em grade é um modelo computacional distribuído que consegue alcançar grandes taxas de processamento e compartilhamento de recursos dividindo as tarefas entre as máquinas de uma rede local ou global, formando uma máquina virtual (FOSTER, 1998).

Muitas companhias apostam na computação em grade como uma tecnologia que revolucionará as técnicas de processamento modernas.

Cluster é definido como um conjunto de computadores, que utilizam a mesma infraestrutura de *hardware* e *software*, cooperando para alcançar um objetivo comum.

Há diversos tipos de cluster. Um tipo famoso é o cluster da classe *Beowulf*, constituído por diversos nós escravos gerenciados por um só computador (PFISTER, 1997).

Os termos cluster e grade (*grid*) significam quase a mesma coisa. Para alguns autores os termos são sinônimos, para outros a diferença entre *cluster* e *grid* está na forma de gerenciamento dos recursos.

Nos clusters sempre existe uma máquina central responsável por gerenciar as operações. Já no caso do *grid*, a computação e o gerenciamento são totalmente distribuídos. Então pode-se afirmar que a diferença está diretamente relacionada com a autonomia de cada componente do sistema.

3 *Bancos de dados distribuídos*

A principal característica de um sistema de banco de dados distribuído é a fragmentação das relações.

Podemos definir um banco de dados distribuído como uma coleção de vários bancos de dados relacionados logicamente, distribuídos em uma rede de computadores. [...]A distribuição dos dados deve ficar transparente para o usuário. (OZSU, 1999, p. 4).

Fragmentar uma relação significa separar pedaços (fragmentos) de uma relação nos sítios do sistema de acordo com predicados e seleções simples.

3.1 Modelos arquiteturais para BDD

Vários modelos devem ser considerados no estudo de sistemas de bancos de dados distribuídos. Há uma gama imensa de formas e modificações que podem ser realizadas na definição de um sistema distribuído. OZSU (1999) define três características arquiteturais que podem ser combinadas para formar um SBDD. Estas são descritas nas próximas três seções.

3.1.1 Autonomia

Este conceito define o poder de decisão que um SGBDD tem sobre o controle do sistema. A palavra controle, neste caso, define se o SGBDD pode, sozinho, tomar decisões pertinentes ao controle de operações executadas no sistema.

3.1.2 Distribuição

Enquanto a autonomia se refere ao controle, o quesito *distribuição* se refere aos dados (OZSU, 1999, p. 84).

O conceito define quão distribuídos estão os dados nos sítios do sistema. Os sistemas podem

ser totalmente distribuídos (quando os dados estão em todos os sítios do sistema), parcialmente distribuídos (quando cada sítio possui um pedaço dos dados), ou não-distribuído (quando somente um servidor possui os dados). No último caso, o sistema funciona como cliente/servidor.

3.1.3 Heterogeneidade

Este conceito se refere às diferenças que os gerenciadores de dados (SGBDD) possuem entre si. Alguns exemplos de diferenças são modelos de dados, linguagens de consulta e protocolo de gerenciamento de transações (OZSU, 1999, p. 84).

Tomando uma classificação mais genérica, diz-se que um sistema de banco de dados distribuído é *homogêneo* se todos os SGBDD dos seus sítios são idênticos e é dito *heterogêneo* o sistema cujos SGBDD dos sítios são diferentes entre si.

3.2 Arquiteturas

A partir das três características citadas na seção anterior é possível construir arquiteturas que satisfaçam as necessidades dos usuários do sistema.

Sistemas distribuídos podem ser construídos sobre várias topologias e arquiteturas de sistemas de comunicação. Esta sessão descreve as principais arquiteturas que podem ser construídas a partir da escolha de algumas características do ambiente.

3.2.1 Cliente/servidor

Antigamente (em alguns casos atualmente também) o processo que solicitava um determinado serviço era o mesmo que provia este serviço.

O modelo cliente/servidor tem uma idéia simples e elegante: distinguir a função de solicitar e fornecer um determinado serviço, dividindo estas funções em duas classes: cliente e servidor. (OZSU, 1999, p. 88).

O motivo da existência deste capítulo é simples: o software que foi estudado e alterado neste trabalho utiliza o modelo cliente/servidor para a comunicação entre os sítios do cluster e entre os utilizadores e o serviço de banco de dados.

No que se refere a SGBD, o conceito de software cliente e software servidor estende, de certa forma, a arquitetura centralizada. O software servidor, neste caso, executa algumas

funções adicionais além de fornecer o acesso ao banco de dados ao software cliente. Tais funções são (OZSU, 1999, p. 89):

- **Controle semântico dos dados:** integridade, correlação e verificação;
- **Otimização de consultas;**
- **Administração de transações:** correção de conflitos e controle de acesso;
- **Recuperação:** integridade, verificação e reconstrução dos dados.

3.2.2 *Peer to peer (P2P)*

Nos sistemas distribuídos, a organização interna dos dados em cada máquina pode ser diferente. Para que se possa implementar níveis de independência (que posteriormente resultarão em garantia de serviço e tolerância a falhas) faz-se necessário manter, além do esquema global (ECG), um esquema local (ECL) nos sítios para definir a organização interna.

Desta forma, cada máquina pode responder pelo sistema inteiro e, quando necessita de um serviço que não está disponível no esquema local, pode solicitar o mesmo a uma outra máquina indicada no esquema global, criando assim uma rede de comunicação *peer to peer*.

Tanenbaum (2008, p. 102) trata estas forma de gerenciamento como níveis de hierarquia. Alguns sistemas possuem coordenadores. Isto implica em adicionar níveis hierárquicos ao sistema. Nos sistemas *P2P* conhecidos como não-hierárquicos, todos os componentes são iguais, nenhum comanda e todas as decisões são tomadas coletivamente.

Sistemas *P2P* não-hierárquicos possuem boa escalabilidade e balanceamento de carga. Outras técnicas de otimização podem ser utilizadas para melhorar ainda mais estes dois fatores.

A figura 3.1 ilustra a diferença entre os sistemas *P2P* hierárquicos e os não-hierárquicos.

3.3 Classificação

Alguns termos específicos são comumente utilizados para classificar ou agrupar os sistemas de banco de dados distribuídos. Esta seção descreve alguns destes termos.

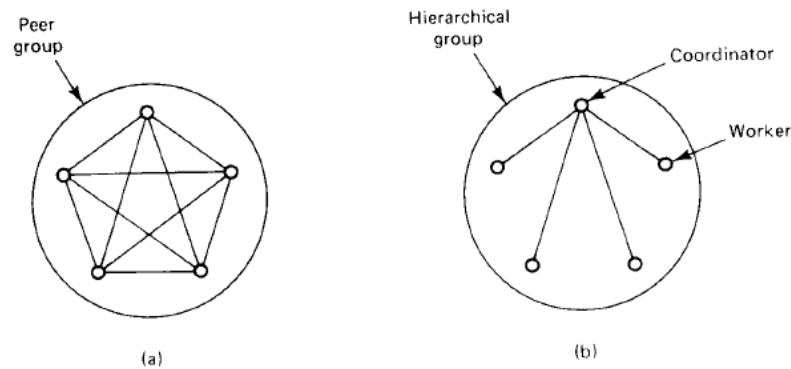


Figura 3.1: Comunicação não-hierárquica (a). Comunicação hierárquica (b). Fonte: (TANENBAUM, 2008, p. 102)

3.3.1 Mestre/escravo (*Master-slave*)

No âmbito de sistemas de banco de dados distribuídos o esquema mestre/escravo define o tipo de sistema que possui um mestre, ou seja, uma máquina onde as transações são processadas e onde os dados podem ser alterados. A máquina mestre distribui os dados para N máquinas escravas que permitem a leitura dos dados pelos clientes.

Transações não podem ser submetidas às máquinas “escravas”. As alterações nos dados somente são tratadas pelo mestre.

3.3.2 Multimaster

O termo multimaster ou “múltiplos mestres” se refere ao esquema de sistema de banco de dados distribuído que possui vários servidores e onde uma transação pode ser tratada por qualquer um deles, o qual mantém o conjunto sincronizado depois da alteração.

Esse tipo de sistema exige um controle complexo de conflitos, mas é muito versátil e eficiente no que diz respeito à escalabilidade.

3.3.3 Sincronismo e assincronismo

Dada uma transação de banco de dados T executada em um sistema de banco de dados composto pelos sítios S_1, S_2, \dots, S_n , o sistema é definido *síncrono* se, e somente se, T é efetivada somente depois da efetivação das alterações em cada sítio que possui itens de dados modificados por T .

Caso contrário o sistema é dito *assíncrono*.

3.3.4 Replicação total ou parcial

Os termos *total* e *parcial* tem significado óbvio. Diz-se que a replicação é total quando cada sítio de um sistema de banco de dados distribuído possui todos os itens de dados e esquemas pertencentes ao banco de dados em questão.

Caso contrário, afirma-se que os dados estão fragmentados entre os sítios, ou seja, o sistema possui replicação parcial.

A diferença entre replicação total e parcial tem importância fundamental neste trabalho, visto que o objetivo principal é adicionar as funções de replicação parcial em um SGBDD que trabalha com replicação total.

A seção a seguir define uma classificação para as fragmentações de dados.

3.4 Tipos de fragmentação

As fragmentações são classificadas, de acordo com a natureza dos fragmentos em três tipos: horizontal, vertical e híbrida (junção dos dois anteriores).

3.4.1 Fragmentação horizontal

Fragmentar uma relação horizontalmente significa particionar a mesma ao longo de suas tuplas. (OZSU, 1999, p. 112)

O particionamento é realizado através de predicados (condições) impostas que filtram os dados que compõem cada fragmento das relações.

O operador da álgebra relacional responsável pela fragmentação horizontal é o σ .

Um fragmento é um subconjunto das tuplas da relação em questão e pode ser definido com um comando de seleção de uma linguagem de consulta, como a SQL. Por exemplo: selecionar todos os atributos de todas as tuplas de uma relação “CLIENTE” que possuem o atributo “CODIGO” > 20.

3.4.2 Fragmentação vertical

A fragmentação vertical de uma relação R produz fragmentos R_1, R_2, \dots, R_n que contém um subconjunto dos atributos de R e deve conter a chave primária da mesma. (OZSU, 1999, p. 131)

O operador da álgebra relacional responsável pela fragmentação vertical é o π .

No caso dos fragmentos verticais, o fragmento pode ser definido através de um comando de projeção de uma linguagem de consulta. Por exemplo: selecionar os atributos “CODIGO” e “NOME” de todas as tuplas uma relação “CLIENTE”.

3.4.3 Fragmentação híbrida

Chamamos de fragmentação híbrida o sistema de fragmentação que executa a fragmentação horizontal sobre um fragmento vertical ou vice-versa.

No exemplo da relação “CLIENTE”, um fragmento híbrido pode ser a seleção dos atributos “CODIGO” e “NOME” nas tuplas que possuem o atributo “CODIGO” > 20 .

3.5 Alocação dos fragmentos

A tarefa da alocação de fragmentos em um sistema de banco de dados distribuído consiste em associar um conjunto de fragmentos (F_1, F_2, \dots, F_n) a um conjunto de sítios (S_1, S_2, \dots, S_n) do sistema.

Ozsu (1999) define o problema de “alocação ótima” como a produção de uma associação de alocação que respeite dois critério de otimização:

- Custo mínimo
- Desempenho

Muito estudo já foi dedicado ao assunto de produzir fragmentação ótima. No entanto, este assunto não será abordado neste trabalho. O objetivo aqui é implementar ferramentas que permitam ao administrador do sistema de banco de dados definir os sítios, fragmentos e a associação dos dois anteriores (alocação) e, a partir disso, aplicar as regras e fragmentar os dados de forma transparente aos usuários.

3.6 Arquiteturas para integração dos dados

Para que os dados sejam corretamente fragmentados, uma estratégia deve ser definida para integrar os dados no processo pós-fragmentação, que é o processo de recuperação (consulta) dos dados fragmentados.

Existem duas abordagens para integração de dados fragmentados (GIUSEPPE, 2004):

- **LAV (Local as View):** as fontes dos dados são definidas de acordo com o esquema global. Isso significa que o esquema global é definido primeiro e os fragmentos são distribuídos nos repositórios seguindo as regras definidas no mesmo;
- **GAV (Global as View):** o esquema global é definido a partir das informações das fontes dos dados. Isso significa que o esquema global é dependente da estrutura das fontes de dados, para isso é necessário que as fontes de dados exportem o formato de sua (s) estrutura (s);

3.7 Regras de uma fragmentação correta

Ozsu (1999) define três regras para a fragmentação, que juntas garantem que o banco de dados relacional não sofre modificações semânticas devido à fragmentação:

- **Completeness:** Se uma relação R é decomposta em n fragmentos. Cada item de dados pode ser encontrado em um ou mais desses fragmentos. Na prática, esta regra define que a fragmentação não pode fazer que os dados da relação inicial sejam perdidos.
- **Reconstruction:** Se uma relação R é decomposta em n fragmentos, deve ser possível definir um operador que reconstrua (forme) a relação inicial a partir dos n fragmentos.
- **Disjunction:** Se uma relação R é decomposta horizontalmente em n fragmentos e um item de dados está em um destes fragmentos, o mesmo item não pode estar em nenhum outro. Este critério define que os fragmentos são disjuntos, ou seja, sejam os fragmentos R_i e R_j dois fragmentos quaisquer de R , com $i \neq j$, então se uma tupla $t \in R_i$, então $t \notin R_j$

Se tais regras forem satisfeitas para todas as operações do sistema, pode-se garantir que a fragmentação está correta, ou seja, não altera o funcionamento do mesmo.

3.8 Controle distribuído de concorrência

Assim como nos sistemas centralizados, é necessário garantir que uma alteração em um conjunto de dados mantenha o esquema de banco de dados íntegro após a conclusão (*commit*) da transação relacionada. O módulo do SGBD que executa essa função é chamado de “gerenciador de transações”.

As técnicas e algoritmos de controle de concorrência são divididos, de acordo com sua taxonomia, em pessimistas e otimistas (OZSU, 1999, p. 307).

A diferença entre os algoritmos pessimistas/otimistas é que os pessimistas sincronizam os dados da transação concorrente mais cedo do que os otimistas.

Os algoritmos de controle de concorrência são divididos em dois grandes grupos de acordo com a forma com que escolhem a transação vencedora (no caso de concorrência entre duas ou mais transações, o SGBD deve escolher qual delas será efetivada e as demais serão abortadas) (OZSU, 1999, p. 306) :

- Baseados em bloqueios (*locks*): São os algoritmos que bloqueiam o conjunto de dados sempre que uma transação é executada sobre ele até que a mesma seja finalizada. Então qualquer transação que tentar executar alguma alteração em paralelo sobre estes dados é abortada.
- Baseados em tempo (*timestamp*): São os algoritmos que escolhem a transação vencedora pelo momento em que a mesma iniciou. Neste caso, as transações mais antigas são efetivadas e as mais novas são descartadas.

Existem muitos algoritmos para permitir o controle de concorrência. Dos mais notáveis, podem-se citar o 2PL (*two-phase locking*), TO (*Timestamp ordering*) e o MVCC (*Multi-version concurrent control*). Este trabalho detalha o algoritmo MVCC porque o mesmo é utilizado na implementação do protótipo.

3.8.1 O algoritmo MVCC

É algoritmo pessimista de controle de versões baseado em tempo (*timestamp*). Neste algoritmo, cada escrita de um item de dado cria uma nova cópia do mesmo.

Cada versão de um item de dado é marcada com o *timestamp* (identificador único temporal) da transação que executou a operação.

Uma operação de leitura disparada por uma transação R com *timestamp* t retorna o item de dado que possui o maior *timestamp* menor que t .

Uma operação de escrita E com *timestamp* t somente é aceita se não existir nenhuma outra transação que leu/escreveu o mesmo item de dado com *timestamp* maior que t . Caso contrário o gerenciador de transações rejeita a transação e retorna erro.

As várias cópias dos itens de dados permanecem armazenadas no sistema de banco de dados permanentemente. Por isso os sistemas que utilizam o MVCC necessitam de algum mecanismo de limpeza (*clean-up*) para liberar as cópias antigas dos dados.

3.9 Protocolos de confiabilidade distribuída

Os SGBDD precisam manter a sincronização dos dados quando ocorre a finalização das transações. Isso também é importante no caso de algum sítio necessitar de recuperação pela ocorrência de alguma falha qualquer.

Para facilitar a tarefa de sincronização, existem alguns protocolos que garantem a confiabilidade no processo de finalizar as transações, ou seja, se uma transação é efetivada ou abortada em algum sítio, os protocolos garantem que todos os demais sítios envolvidos também executarão o mesmo procedimento.

Estes protocolos geralmente utilizam estratégias de envio de mensagens em várias fases do processo de finalização para manter os sítios sincronizados. Dentre os principais protocolos é possível citar o 2PC e o 3PC. A maioria dos protocolos de confiabilidade é baseada em um dos dois anteriores. Por conveniência, esta implementação utiliza o protocolo 2PC para garantir a confiabilidade das transações.

3.9.1 O protocolo 2PC

Para garantir que as transações distribuídas obedeçam às regras impostas pelas propriedades ACID (acrônimo formado pelas iniciais de atomicidade, consistência, isolamento e durabilidade), é necessário utilizar um protocolo de efetivação de transações em várias fases.

O 2PC é um protocolo simples e elegante que assegura a efetivação atômica das transações distribuídas. (OZSU, 1999, p. 381).

O protocolo é composto de duas fases:

1. Quando o comando de efetivação (*commit*) é enviado para um sítio, denominado *coordenador da transação*, este envia para todos os sítios cujos dados foram alterados (denominados *participantes da transação*), uma mensagem de “preparar commit” e grava uma entrada de *log* para o evento (“início do commit”). Então ocorre uma verificação de conflitos em todos os participantes e cada um deles informa seu “voto” para o coordenador quanto a continuar o processo ou abortar a operação.
2. Se algum participante votou em abortar a transação, o coordenador envia um comando para abortar/ignorar o *commit* (“aborto global”) para todos os participantes e grava uma entrada no seu *log* para o evento.

Se todos os participantes votaram favoravelmente em continuar o processo, o coordenador envia uma mensagem de “commit global” para todos os participantes. Então cada um deles efetiva a transação, gravando os dados nos arquivos e *logs* e retorna o resultado da operação (sucesso ou falha) para o coordenador. No caso de falha, o coordenador reporta o acontecimento ao usuário e grava isto no *log*. Caso contrário, é gravado o evento “fim de transação” no *log* do coordenador.

A figura 3.2 mostra o fluxograma do funcionamento do protocolo 2PC do ponto de vista do coordenador e dos participantes.

3.10 Sistema gerenciador de bancos de dados distribuídos

[...] Um sistema gerenciador de bancos de dados distribuídos é o software que permite a administração do banco de dados distribuído a deixa a distribuição transparente para os usuários. (OZSU, 1999, p. 4).

O gerenciador de BDD deve desempenhar as mesmas funções que um SGBD centralizado sem que os usuários tenham conhecimento da fragmentação dos dados nos sítios e deve fornecer ao administrador de banco de dados distribuído formas de definir a maneira com que os dados são fragmentados e como devem ser alocados.

Os próximos capítulos tratam de definir uma proposta viável para transformar o PostgreSQL em um SGBDD que atenda às características citadas acima.

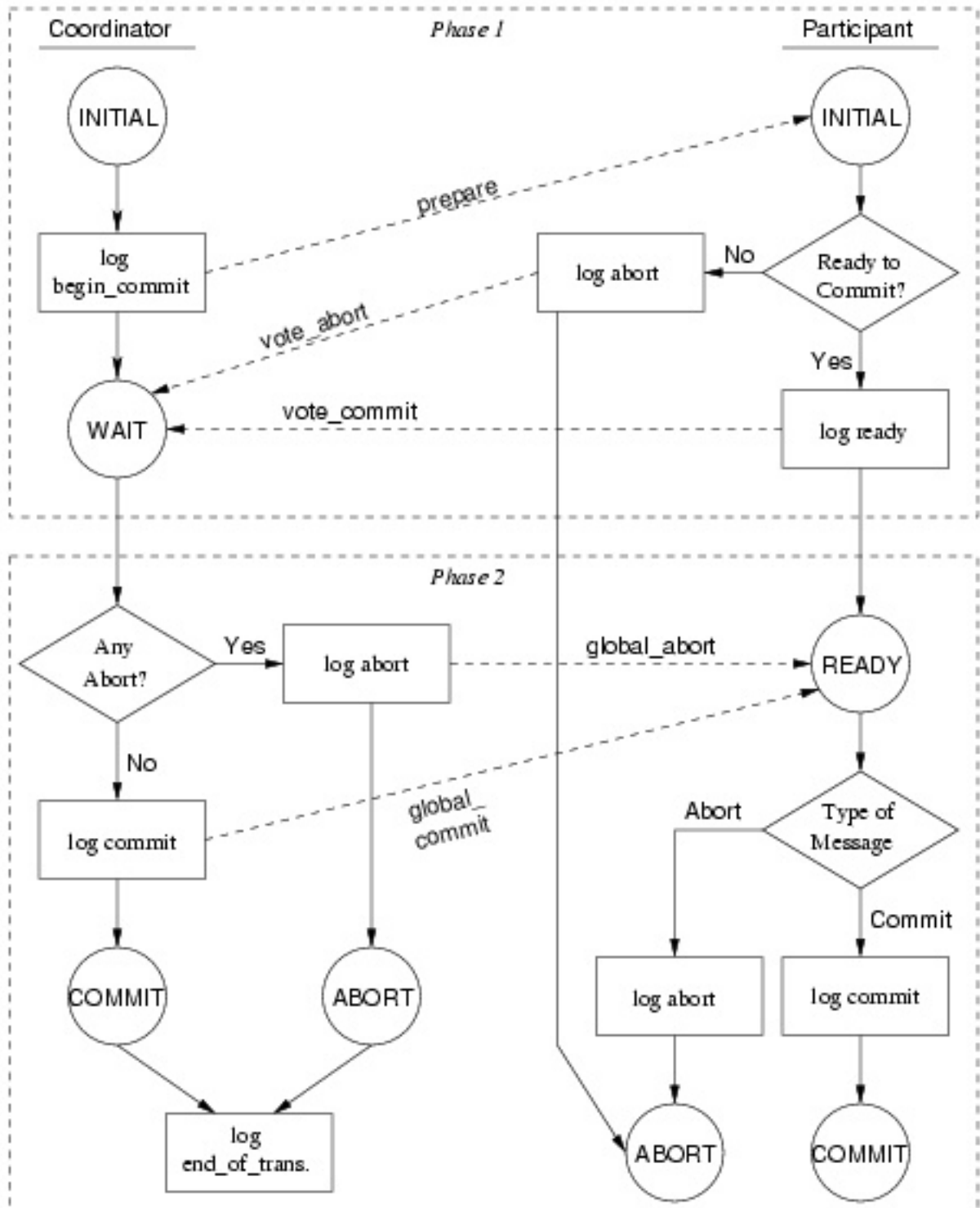


Figura 3.2: Fluxograma do 2PC. Fonte: (OZSU, 1999, p. 383)

4 Exemplos de SGBDs distribuídos

Este capítulo descreve e ilustra algumas implementações de *grid* em SGBDs disponíveis no mercado. Também é apresentada uma comparação entre as implementações descritas, considerando algumas características importantes do ambiente.

4.1 PostgreSQL e pgCluster

PostgreSQL é um sistema gerenciador de banco de dados de código-fonte aberto, multiplataforma, com mais de 15 anos de desenvolvimento. Atende os padrões SQL99 e seu gerenciador de transações implementa as propriedades ACID (POSTGRESQL, 2007).

Dentre uma gama enorme de SGBD relacionais e objeto-relacionais disponíveis no mercado, o PostgreSQL se sobressai devido à sua aceitação pela comunidade de software livre mundial. Além de todas as qualidades citadas acima, o PostgreSQL possui grande usabilidade em ambientes distribuídos homogêneos, possuindo grande parte das funções de replicação já implementadas e testadas.

Por este motivo, este trabalho utiliza o PostgreSQL (*postgres*, como é costumeiramente chamado) como base no desenvolvimento do ambiente distribuído proposto.

O *pgCluster* é um sistema de replicação síncrono baseado no PostgreSQL. O sistema possui duas funções principais:

1. **Balanceamento de carga:** Um módulo recebe os comandos de consulta e transações e distribui da forma mais conveniente levando em consideração a carga (capacidade de processamento comprometida) dos servidores.
2. **Alta disponibilidade:** É garantida através da replicação total dos dados, então se um servidor cai, os demais podem responder em seu lugar, desde que pelo menos um servidor de replicação continue no ar.

No caso do pgCluster, o esquema conceitual global é replicado de forma síncrona entre os sítios do cluster. Então é possível saber onde buscar determinada informação através de uma consulta local.

Conforme pode-se perceber, o maior problema do pgCluster é a falta das funcionalidades de fragmentação que exigem que o usuário replique totalmente as bases de dados para que o sistema funcione, degradando o desempenho e também desperdiçando recursos do sistema.

Nos próximos capítulos são propostas alterações que visam a solução destes problemas construindo, desta forma, uma estrutura de grade para o PostgreSQL.

4.2 DRDA - Uma arquitetura distribuída para BD relacionais

DRDA representa a proposta da IBM na área de bancos de dados distribuídos. Esta arquitetura define regras e protocolos para desenvolver programas que implementam acesso a bancos de dados distribuídos (BEDOYA, 2006, p. 84).

A linguagem SQL é utilizada como padrão na proposta DRDA. O protocolo de confiabilidade distribuída recomendado é o 2PC. Todo SGBD que está de acordo com a norma deve, obrigatoriamente, implementar o protocolo 2PC.

O padrão define as regras e ações que devem ser executadas desde a chegada de uma requisição em um SGBD até que a mesma seja executada e o resultado retornado para o solicitante. Para tal, o padrão utiliza a terminologia de AR (*application requester*) para o servidor de banco de dados que recebe a solicitação e AS (*application server*) para os demais servidores envolvidos no processamento da requisição.

O DRDA provê quatro níveis de serviço para uma requisição SQL dentro de um ambiente de banco de dados distribuído (BEDOYA, 2006, p. 85):

- **Nível 0 (requisição):** Inicia quando uma requisição, denominada UOW (*unit of work*), chega a um SGBD (denominado AR).
- **Nível 1:** uma ou mais requisições SQL são executadas em apenas um servidor.
- **Nível 2:** várias requisições SQL são executadas em vários servidores. O protocolo 2PC é necessário neste caso.
- **Nível 3:** uma única requisição SQL é executada em vários servidores. Ela contém referência para vários SGBD, como uma junção entre relações armazenadas em locais di-

ferentes.

Muitos fabricantes de SGBD adotam o padrão DRDA nos seus produtos. Por exemplo: Informix, Oracle e XDB. DRDA foi adotado como um padrão de mercado pelo OpenGroup.

4.3 Implementação do DRDA no SGBD DB2

A IBM desenvolve um SGBD desde a década de 70. Seu produto, o DB2, é considerado um dos mais completos sistemas gerenciadores de bases de dados atualmente no mercado.

DB2 implementa os níveis de serviço DRDA 1 e 2, sendo que o nível 2 está disponível somente para servidores que utilizam a arquitetura de rede SNA, desenvolvida pela IBM. A implementação do nível 3 será desenvolvida futuramente (BEDOYA, 2006, p. 86).

Para utilizar os objetos distribuídos, o código SQL deve conter as funções de conexão com cada SGBD definidas explicitamente, bem como a localização dos objetos. Portanto, não há transparência para o DBA, ou seja, o administrador/programador deve conhecer o esquema de distribuição para conseguir atingir seus objetivos.

A seguir está um exemplo de script SQL, que poderia executar no DB2 utilizando um banco de dados distribuído.

```
/*Exclui todos os dados da relação PRODUTO do nodos de
FLORIANOPOLIS e JOINVILLE*/
/*Define a utilização do sinalizador de erro SQLCA */
EXEC SQL INCLUDE SQLCA END-EXEC.

/*Conecta, explicitamente, o servidor FLORIANOPOLIS*/
EXEC SQL CONNECT TO FLORIANOPOLIS
END-EXEC.
IF SQLCODE NOT = 0 AND SQLCODE NOT = -842 THEN
DISPLAY "Erro conectando FLORIANOPOLIS"
END-IF.

/*Conecta, explicitamente, o servidor JOINVILLE*/
EXEC SQL CONNECT TO JOINVILLE
END-EXEC
```



```

IF SQLCODE NOT = 0 AND SQLCODE NOT = -842 THEN
DISPLAY "Erro conectando JOINVILLE"
END-IF.

/*Apaga os produtos de Joinville*/
EXEC SQL SET CONNECTION JOINVILLE END-EXEC
EXEC SQL DELETE FROM PRODUTO END-EXEC
IF ERR-FLG = 0 THEN
/*Se tudo deu certo em Joinville, apaga de Florianopolis*/
EXEC SQL SET CONNECTION FLORIANOPOLIS END-EXEC
EXEC SQL DELETE FROM PRODUTO END-EXEC
IF ERR-FLG = 0 THEN
IF ERR-FLG = 0 THEN
/*Commit protegido pelo protocolo 2PC*/
EXEC SQL COMMIT END-EXEC

IF ERR-FLG = 0 THEN
STOP RUN
END-IF
END-IF
END-IF

END-IF.

* Em caso de erros, ROLLBACK protegido pelo protocolo 2PC
EXEC SQL ROLLBACK
END-EXEC.
STOP RUN.

```

4.4 MySQL Cluster

MySQL é um famoso SGBD de código aberto mantido pela empresa americana *MySQL AB* (recentemente incorporada à *Sun Microsystems* que também foi incorporada à *Oracle Corporation*). Um dos melhores atributos do SGBD é a possibilidade de escolha entre os vários me-

canismos de armazenamento disponíveis. Entre eles *MyISAM*, *InnoDB*, *MERGE* e *MEMORY*. Cada mecanismo é indicado para um caso de uso específico. Na versão 5.0 do produto, o mecanismo *NDB* foi adicionado. Este mecanismo fornece funções de distribuição dos dados entre várias instâncias do *MYSQL* e extensões para administração de dados fragmentados utilizando arquitetura multimaster.

A fragmentação pode ser implementada em tabelas que possuem chave primária e a alocação é realizada automaticamente através de um hash sobre o valor da mesma (MYSQL, 2007).

O mecanismo *NDB* não suporta chaves estrangeiras e não permite que o administrador escolha o servidor onde as tuplas serão armazenadas a partir de padrões sobre os dados, o que limita muito a manutenção de um *SBDD* sobre uma rede *WAN*, pois os dados podem ficar longe dos seus respectivos consumidores.

Os comandos abaixo ilustram a criação de uma relação fragmentada em um ambiente de cluster com cinco servidores *MySQL*(MYSQL, 2007).

```
mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)
```

```
mysql> SHOW CREATE TABLE ctest \G
***** 1. row *****
      Table: ctest
Create Table: CREATE TABLE 'ctest' (
  'i' int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

```
shell> ndb_mgm
ndb_mgm> SHOW
Cluster Configuration
-----
[ndbd(NDB)]      1 node(s)
id=2      @127.0.0.1 (Version: 3.5.3, Nodegroup: 0, Master)

[ndb_mgmd(MGM)] 1 node(s)
id=1      @127.0.0.1 (Version: 3.5.3)
```

```
[mysqld(API)] 3 node(s)
id=3 @127.0.0.1 (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

Todas as tuplas da relação “ctest” são distribuídas de forma transparente no cluster descrito pelo comando “SHOW”.

4.5 Oracle Real Application Clusters (RAC)

O produto Oracle Database é um consolidado SGBD, pioneiro em muitos aspectos nesta área. No ano de 2001 foi lançada a versão 9i do pacote de aplicativos de gestão de bancos de dados. A partir desta versão, há um produto opcional, chamado Oracle RAC, que fornece funções de “clusterização” que possibilitam a construção de sistemas distribuídos com replicação multimestre e troca de dados.

Oracle RAC suporta a distribuição transparente de um banco de dados entre vários servidores, provendo tolerância a falhas e alta disponibilidade (GOPALAKRISHNAN, 2006). Na configuração padrão, os sites acessam os dados no mesmo dispositivo de armazenamento. No entanto, há a opção de “shared-nothing”, onde os dados são fragmentados em diversos dispositivos e o software controla as funções de compartilhamento, bloqueio e recuperação distribuídos.

A solução da Oracle é escalável e de fácil configuração. Existem ferramentas gráficas para a manipulação dos servidores que compõem o cluster. Há suporte para clusters com até 100 servidores.

4.5.1 Cache Fusion

Visando melhorar o desempenho das consultas e compartilhamento de dados no cluster, a Oracle desenvolveu um mecanismo de cache compartilhado para sua solução de cluster. *Cache Fusion* utiliza o princípio de que consultar um dado que esteja em cache em outro site do cluster é mais rápido do que consultar o dado no disco local (GOPALAKRISHNAN, 2006, p. 241).

4.6 Comparativo das tecnologias

Para comparar as tecnologias atualmente em uso foram consideradas características fundamentais abordadas nos itens anteriores, que quando somadas, formam um ambiente distribuído praticamente completo.

A tabela seguinte aponta tais características e indica sua presença em cada um dos gerenciadores apresentados previamente.

Característica	MySQL Cluster	Oracle RAC	IBM DB2	pgCluster
Consultas distribuídas	Sim	Sim	Sim	Não
Tecnologia multi-mestre	Sim	Sim	Sim	Sim
Replicação parcial	Não	Sim	Sim	Sim
Replicação síncrona	Sim	Sim	Sim	Sim
Replicação assíncrona	Não	Não	Não	Não
Fragmentação	Sim	Sim	Sim	Não
Definição de regras de fragmentação	Não	Sim	Não	Não
Integridade referencial entre os sítios	Não	Não	Não	Não
Protocolo de confiabilidade distribuída	2PC	2PC	2PC	2PC

As características mais focadas pelo presente trabalho estão destacadas na tabela e são relacionadas à definição flexível de regras de fragmentação pelo DBA e a manutenção da integridade referencial, através de chaves estrangeiras, entre os sítios do sistema distribuído. A próxima seção trata destas características.

Também é possível notar que tais características não estão totalmente disponíveis nas outras implementações abordadas.

5 *Projeto do pgGrid*

Esta seção apresenta o projeto de desenvolvimento do pgGrid, dando destaque às modificações que serão realizadas no PostgreSQL e no pgCluster.

5.1 Adição da extensão à gramática SQL

Atualmente a configuração da replicação no pgCluster exige a manipulação de arquivos XML. Desta forma, ficará muito difícil para o administrador definir as novas configurações complexas, como a definição dos fragmentos e a alocação dos mesmos. Para resolver este problema serão adicionados comandos específicos para a definição destes parâmetros que serão gravados nos catálogos do sistema.

Três grupos de instruções especiais serão adicionadas à gramática:

1. Definição, alteração e exclusão de sítio: Os comandos seguem o esquema a seguir e produzem a disponibilização de servidores no *grid* para alocar os fragmentos.

```
CREATE SERVER <nome> HOST <nome do host> PORT <porta>
RECOVERY PORT <porta recuperador>;
```

```
ALTER SERVER <nome> HOST <nome do host> PORT <porta>
RECOVERY PORT <porta recuperador>;
```

```
DROP SERVER <nome>;
```

onde *nome* é um apelido para o novo servidor, *nome do host* é o endereço de rede do servidor, *porta* é a porta que o servidor de replicação usará para replicar os dados e *porta recuperador* é a porta que o servidor de replicação usará para recuperar as transações no caso de falhas. Os parâmetros *porta* e *porta recuperador* possuem os valores 5432 e 7000, respectivamente, como padrão.

Exemplo de sintaxe:

```
CREATE SERVER gustavo HOST gustavotonini.com PORT 5432
RECOVERY PORT 7000;
```

```
DROP SERVER gustavo;
```

2. Definição de fragmento: estes comandos definem um fragmento de dados que pode ser alocado a um ou mais servidores:

```
CREATE FRAGMENT <nome> ON <relacao> [<lista de atributos>]
[<sentença where>];
```

```
ALTER FRAGMENT <nome> ON <relacao> [<lista de atributos>]
[<sentença where>];
```

```
DROP FRAGMENT <nome>;
```

onde *nome* é um apelido único para o fragmento, *relacao* é uma relação presente no esquema e *lista de atributos* é uma lista com atributos da relação *relacao* e *sentença where* é instrução de seleção normal do padrão SQL que define um subconjunto de dados da relação.

É importante ressaltar que, neste caso, todos os atributos e relações envolvidos devem estar presentes no esquema de banco de dados.

Exemplo de sintaxe:

```
CREATE FRAGMENT cliente_pessoa_fisica ON CLIENTE
where tipo_pessoa = 'F';
```

```
CREATE FRAGMENT cliente_filial1 ON CLIENTE
where cod_filial = 1;
```

```
CREATE FRAGMENT dados_basicos_clientes ON
CLIENTE (codigo, nome);
```

```
CREATE FRAGMENT dados_basicos_clientes_filial1 ON
```

```

CLIENTE (codigo, nome)
where cod_filial = 1;

```

```

DROP FRAGMENT cliente_pessoa_fisica;

```

3. Alocação de fragmentos: estes comandos produzem uma alocação de um fragmento para um servidor:

```

PLACE <nome fragmento> ON <nome servidor>;

```

```

REMOVE <nome fragmento> FROM <nome servidor>;

```

onde *nome fragmento* é o apelido do fragmento e *nome servidor* é o apelido do servidor.

É importante ressaltar que quando uma relação/atributo é criado, o mesmo é alocado automaticamente para o servidor onde foi criado, podendo o administrador, posteriormente, transferi-lo para outro servidor.

Exemplo de sintaxe:

```

PLACE cliente_pessoa_fisica ON gustavo;

```

```

REMOVE cliente_pessoa_fisica FROM gustavo;

```

Como o PostgreSQL utiliza o *yacc/bison*¹ para gerar o analisador léxico do motor, basta definir os novos comandos na gramática que o código será gerado automaticamente.

5.2 Modificação nos catálogos e *bootstrap*

Atualmente o pgCluster mantém as configurações da replicação em arquivos no formato XML. No entanto, como serão adicionadas configurações mais complexas, como definição de fragmentos e alocação, será necessário criar catálogos para armazenar tais informações.

Os catálogos do pgCluster ficam armazenados em todos os servidores que compõem o sistema.

¹Famoso gerador de analisadores léxico e sintático para compiladores

5.2.1 O catálogo de sites (*pg_grid_site*)

É o catálogo em que são armazenadas as referências para os servidores que compõem o *grid*. A estrutura de atributos do mesmo está na tabela seguinte:

Nome do atributo	Tipo de dado	Descrição
siteid	Oid	Identificador do sítio
sitename	Char	Nome do sítio
hostname	Char	Endereço de rede do sítio
port	int2	Porta para replicação
recoveryport	int2	Porta para recuperação de falhas

5.2.2 Os catálogo de fragmentos (*pg_grid_fragment*)

São os catálogos em que são armazenadas as informações referentes à definição dos fragmentos do *grid*. Duas relações foram necessárias para definir a estrutura.

A primeira relação (*pg_grid_fragment*) contém os atributos básicos e os predicados para fragmentação horizontal. A estrutura segue a tabela a seguir:

Nome do atributo	Tipo de dado	Descrição
fragmentid	Oid	Identificador do fragmento
relid	Oid	Identificador da relação fragmentada
fragmentname	Char	Nome do fragmento
whereclause	text	Cláusula de seleção para a fragmentação horizontal

A segunda relação (*pg_grid_fragment_attribute*) contém a lista de colunas da relação para a fragmentação vertical. A estrutura segue a tabela a seguir:

Nome do atributo	Tipo de dado	Descrição
fragmentid	Oid	Identificador do fragmento
attid	Oid	Identificador do atributo fragmentado

5.2.3 O catálogo de alocação dos fragmentos (*pg_grid_allocation*)

Para manter a associação dos fragmentos definidos pelo usuário com os sítios que armazenarão os mesmos, faz-se necessário criar um catálogo adicional (do tipo associativo). O esquema deste catálogo está definido a seguir:

Nome do atributo	Tipo de dado	Descrição
fragmentid	Oid	Identificador do fragmento
siteid	Oid	Identificador do sítio que armazena o fragmento

A figura 5.1 mostra o esquema entidade-relacionamento dos novos catálogos.

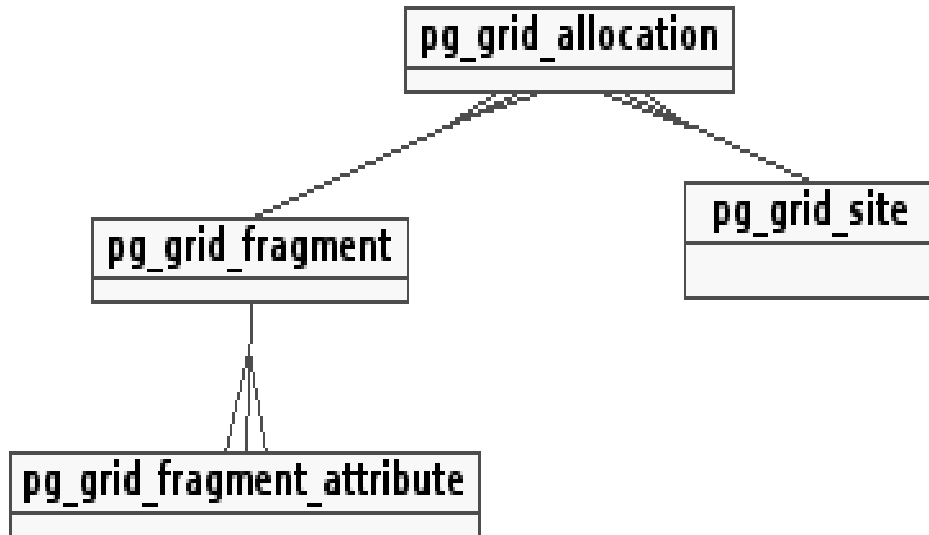


Figura 5.1: DER dos novos catálogos

5.3 Restrições semânticas

Além das verificações básicas (duplicidade de nomes e se as colunas pertencem à relação) na criação e distribuição dos fragmentos, o analisador semântico deve realizar algumas verificações adicionais antes de aprovar a execução dos comandos.

- Não permitir a definição de dois fragmentos idênticos;
- Não permitir definir um sítio para a mesma máquina na mesma porta;
- Não permitir a exclusão da chave primária de relações fragmentadas, pois isto afetaria a capacidade de reconstrução das mesmas.

5.4 Processamento dos comandos de definição do *grid*

Este trabalho propõe uma abordagem do tipo LAV (ver seção 3.6) para a definição do sistema de fragmentação de dados, visto que o esquema global é definido de antemão e só depois são definidas as regras de fragmentação do mesmo.

Tal estratégia é interessante porque permite ao administrador do sistema ter uma visão local dos dados e estruturas antes de implementar a fragmentação, e porque esta abordagem é menos impactante na estrutura do pgCluster, pois o mesmo também utiliza esta estratégia.

O processamento dos comandos DDL que definem o esquema de banco de dados não sofrerá muitas modificações pela extensão. Quando um comando de modificação do esquema é executado em um sítio, o mesmo deve ser replicado para os demais sítios do cluster, incluindo os bloqueios (*locks*) das relações modificadas. Desta forma, o catálogo ficará disponível para consulta em todos os sítios. Isso facilita muito a localização dos dados, necessária em várias situações.

5.4.1 Processamento do comando *PLACE*

O comando “*place*”, além de replicar a alteração nos catálogos, faz com que os dados do fragmento alocado sejam copiados para o sítio escolhido utilizando a biblioteca “*copy*” do *backend* do PostgreSQL.

Antes de iniciar a cópia dos dados, o coordenador deve obter um “*write lock*” (bloqueio de escrita) na relação do fragmento e no catálogo de alocação.

5.5 Localização dos dados distribuídos

Ozsu (1999) define algumas reduções que permitem a implementação de um algoritmo determinístico para verificar a existência de um determinado conjunto de dados em um determinado sítio sem a necessidade de consultá-lo. Isso é realizado através de simples consultas aos catálogos de definição e o procedimento minimiza a utilização da rede no sistema.

A localização dos dados definidos por fragmentação híbrida no protótipo proposto será realizada através da implementação de uma “união” das reduções propostas para fragmentação horizontal e vertical.

5.5.1 Redução para fragmentação horizontal

O comando *CREATE FRAGMENT* contém uma sentença *where* que define a parte horizontal da fragmentação híbrida. Tal sentença é opcional e é composta por vários predicados unidos por operadores lógicos, ou seja,

$$where = P1 \cup P2 \cup \dots \cup Pn$$

Onde P1, P2, ..., Pn são sentenças lógicas. Por exemplo, dada a relação CLIENTE(codigo, nome, endereco, idade), os predicados podem ser:

$$P1 = \sigma_{codigo < 1000}(CLIENTE)$$

$$P2 = \sigma_{idade < 50}(CLIENTE)$$

$$P3 = \sigma_{nome < 'm'}(CLIENTE)$$

Os predicados podem ser unidos por operadores lógicos, então as uniões são substituídas por operadores como o *and* ou *or*. Por exemplo:

$$codigo < 1000 \text{ AND } idade < 50 \text{ AND } nome < 'm'$$

A localização dos dados dos fragmentos horizontais sempre é executada sobre outra sentença “where”, seja dentro de um comando de seleção ou dentro das regras de um comando de atualização/exclusão. Esta sentença também é composta de uma união de predicados de forma idêntica aos fragmentos.

A redução para fragmentação horizontal consiste em encontrar predicados conflitantes entre as duas sentenças de tal forma que impliquem em uma interseção vazia entre os dois conjuntos resultantes. Por exemplo, se a sentença de fragmentação é:

```
where cod_cliente > 5
```

e a sentença do comando que solicita os dados é:

```
where cod_cliente < 5
```

o conjunto resultado se torna obviamente vazio. Quando tais disjunções não são encontradas, não é possível garantir que a intersecção seja vazia, então é necessário replicar os comandos para verificar a existência dos dados.

Baseando-se na verificação desta regra através dos catálogos (replicados) é possível criar um algoritmo que defina se uma sentença de seleção é compatível com uma sentença de fragmentação.

5.5.2 Redução para fragmentação vertical

A redução vertical para um fragmento F pode ser verificada facilmente através de uma consulta ao catálogo *pg_grid_fragment_attribute*.

Se algum atributo necessário para a execução do comando em questão estiver contido no conjunto de atributos de um fragmento do sítio, então este sítio deve ser incluído na execução do comando.

5.6 Processamento das consultas distribuídas

O processamento distribuído de consultas envolve três etapas:

1. Decomposição de consultas e localização dos dados;
2. Executar a consulta nos sítios selecionados;
3. Concatenar os resultados.

A etapa 1 é a mais complexa porque envolve uma busca por intersecções entre o conjunto de dados sendo pesquisado com o conjunto de dados do grupo de fragmentos presentes nos sítios do *grid*.

Dada uma seleção/projeção S , o comando que executa S deve ser enviado para o sítio I se, e somente se, a intersecção entre o conjunto de dados resultante da união dos fragmentos de I , chamado de $F(I)$, com o conjunto S não for vazio, ou seja:

$$S \cap F(I) \neq \emptyset$$

Tal processo está detalhado no item 5.5.

Depois que esta relação é verificada, o comando de seleção deve ser enviado para os sítios utilizando a biblioteca cliente do PostgreSQL.

Quando todos os sítios retornarem seu conjunto de dados resultado da operação, o sítio em que o comando foi submetido deve iniciar o processo de concatenação do resultado que visa montar o conjunto resultado final que será apresentado ao usuário.

Para executar esta tarefa, será adicionado ao processador de seleções um módulo que processa cada tupla retornada pelos sítios, eliminando as duplicatas (considerando a tupla com

maior *timestamp*). A verificação de duplicidade pode ser realizada facilmente, pois quando uma tupla é inserida no PostgreSQL, é associada a ela um identificador único, chamado Oid (*Object identifier*, ou identificador de objeto).

As tuplas duplicadas ocorrem quando o mesmo fragmento está alocado para mais de um sítio, ou quando a tupla pertence a dois ou mais fragmentos alocados em sítios diferentes.

5.7 Distribuição dos dados nas transações

Toda vez que um comando de atualização (*insert*, *update*, *delete*) é submetido ao motor do SGBD, o gerenciador da replicação (servidor de replicação) deve verificar a necessidade de submeter o mesmo a um ou mais sítios do *grid*. Para isso será utilizado o modelo para decomposição de consultas e localização dos dados descrito no item 5.5.

Todos os comandos transacionais (*início*, *rollback*, *commit*, *locks*) deverão ser replicados. O módulo deve usar o protocolo 2PC do PostgreSQL para efetivar as transações.

5.7.1 Índices

Para melhorar o desempenho do acesso às tuplas, o SGBD cria estruturas de dados e métodos de acesso específicos para realizar esta operação. Adicionando alguma redundância, é possível melhorar muito o tempo de acesso aos objetos do sistema.

O procedimento envolve atualizar os dados da estrutura de acesso sempre que ocorre alguma alteração nos itens de dados reais. Essa estratégia implica em alterações também no ambiente distribuído pois, as mesmas operações que ocorrem no ambiente centralizado devem ser “simuladas” no *grid*.

Um índice é composto de uma lista de atributos cujos dados serão ordenados de forma a facilitar o acesso às tuplas que contém tais dados.

O comando de criação de um índice somente deve ser enviado para os sítios que possuem fragmentos que contém todos os atributos definidos na criação do mesmo.

O comando de atualização de índices também deverá ser distribuído entre os sítios que participaram da transação que implicou na atualização do mesmo. Assim, quando a efetivação de uma transação implica na atualização de dados de um ou mais índices, somente os sítios que possuem o conjunto de dados pertencente aos mesmos terão seus dados atualizados.

Esta regra será implementada como uma extensão dos comandos de adição/atualização/exclusão

de tuplas (*insert, update, delete*).

6 *Implementação do pgGrid*

A implementação do *pgGrid* foi dividida nas seguintes etapas:

- Alteração dos catálogos do sistema
- Alteração da gramática do pgCluster
- Alterações nos comandos de manipulação de dados
- Processamento das consultas distribuídas

Este capítulo descreve e documenta as ações executadas em cada etapa da implementação do protótipo, e analisa os resultados.

6.1 **Alteração dos catálogos do sistema**

Adicionar novos catálogos no PostgreSQL é uma atividade simples. Basta criar cabeçalhos C descrevendo cada atributo das relações que formam o catálogo, depositá-los num diretório especial do código-fonte (*src/include/catalog/*) e registrá-los no arquivo que controla a compilação do SGBD (*Makefile*).

Depois de compilado, o PostgreSQL contém um utilitário para a criação do sistema de banco de dados: *initdb*. Ele é responsável pela criação da estrutura do BD e dos catálogos.

Nesta atividade, quatro novos catálogos foram adicionados ao PostgreSQL:

- *pg_grid_site*
- *pg_grid_fragment*
- *pg_grid_fragment_attribute*
- *pg_grid_allocation*

Os dados armazenados nestes catálogos serão gerados pelos comandos adicionados à linguagem DML suportada pelo PostgreSQL. As próximas seções tratam de como estes comandos são analisados/processados.

6.2 Alteração da gramática do pgCluster

As seções anteriores descrevem as alterações necessárias na gramática SQL do PostgreSQL para facilitar a definição do esquema de banco de dados distribuído, como adição de sites, definição de fragmentos e alocação dos mesmos.

Os comandos *CREATE SITE*, *CREATE FRAGMENT* e *PLACE FRAGMENT* foram adicionados à gramática utilizada pelo analisador sintático. Cada um destes comandos foi definido através das regras gramaticais utilizadas pelo gerador *bison* e foram adicionados tratadores para popular os catálogos do pgGrid com os dados presentes nos comandos.

Cada tratador recebe os dados informados pelo usuário no comando (via módulo *parser*) e atualiza seu catálogo correspondente. Este tratamento é realizado no módulo *commands* do PostgreSQL.

Com as funções de administração do *grid* adicionadas ao processador de comandos e populando os catálogos, resta alterar os comandos de seleção e atualização de dados para seguir as regras definidas pelo administrador do *grid*. As próximas seções descrevem como este procedimento foi implementado.

6.3 Alterações nos comandos de manipulação de dados

Implementando de forma fiel o padrão SQL, o PostgreSQL implementa três comandos de manipulação de dados: INSERT, UPDATE e DELETE.

Os comandos acima citados executam funções de baixo nível e internas ao SGBD. Tais funções, implementadas pelo módulo denominado *executor*, manipulam as tuplas nos arquivos de dados. Existem bibliotecas específicas para a manipulação das tuplas, o módulo que executa as operações INSERT, UPDATE e DELETE simplesmente utiliza tais bibliotecas para executar suas operações.

Tais bibliotecas foram alteradas, adicionando-se uma espécie de filtro que desvia o fluxo de execução das mesmas para o código do pgGrid. Neste trecho é realizada a verificação das tuplas de acordo com as regras impostas pelos catálogos de alocação.

Quando os dados contidos nas tuplas indicam que a operação deve ser enviada para outros sites, o código utiliza as funções implementadas no pgCluster para enviar a operação através da rede.

Antes da alteração, o pgCluster fazia replicação total. Por isso, o protocolo de comunicação entre os servidores foi modificado para transferir, além do comando, o endereço de rede do servidor específico que deve receber os dados.

O impacto da fragmentação no mecanismo de replicação do pgCluster foi relativamente pequeno, pois ocorre apenas uma espécie de "filtragem" de tuplas e atributos que devem ser enviados aos sítios que compõem o sistema.

As próximas seções descrevem os testes que foram realizados na ferramenta e nos algoritmos que implementam a consulta dos dados fragmentados.

6.4 Processamento das consultas distribuídas

O processador de consultas é o maior módulo do PostgreSQL e está contido no módulo *executor*. O processador funciona em duas etapas: transformação/otimização e execução. A primeira etapa transforma a árvore de consulta para obter um melhor desempenho e a segunda etapa solicita a leitura das tuplas para o mecanismo de armazenamento e processa as operações definidas na primeira etapa.

Assim, a alteração imposta pelo pgGrid foi realizada no mecanismo de armazenamento e leitura das tuplas. Ao invés de ler tuplas no ambiente local, o mecanismo foi alterado para tornar a consulta distribuída.

Depois que a consulta é otimizada pelo planejador de consultas, uma árvore é retornada para o executor. Esta árvore, denominada *árvore de consulta* contém as operações necessárias para processar e retornar os dados desejados pelo usuário.

Os nós inferiores da árvore contém as operações que acessam diretamente as relações e os superiores contém as operações realizadas com os dados já filtrados (geralmente projeções e operações de conjunto). A figura 6.1 mostra um exemplo de árvore de consulta, onde as relações *J*, *G* e *E* são a base.

O executor do PostgreSQL foi alterado para distribuir as operações que acessam as relações, através das bibliotecas do pgCluster. Então, quando os nós inferiores da árvore de consulta são executados, o pgGrid solicita os dados para os sites que contém algum fragmento da relação (consultando os catálogos) e cria tabelas temporárias para armazenar os dados retornados. Por

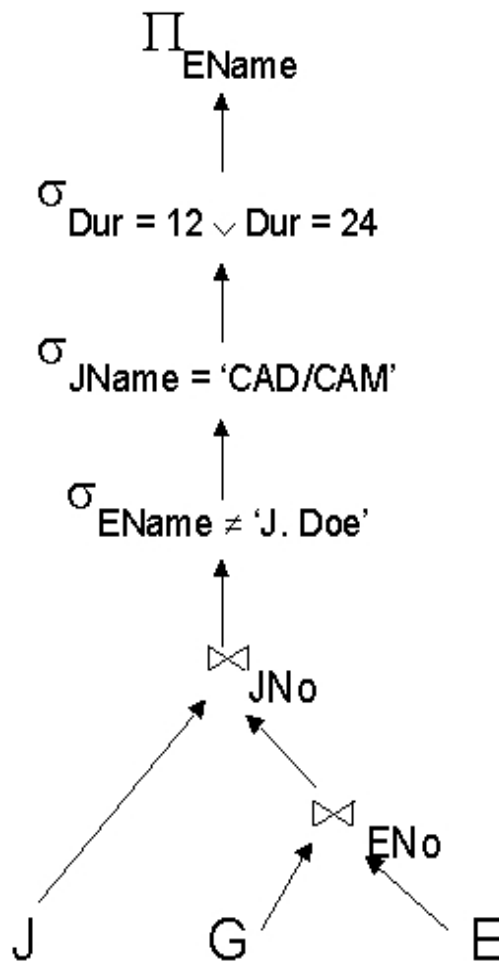


Figura 6.1: Exemplo de árvore de consulta otimizada

fim, executa as operações de junção/união para reconstruir a relação previamente fragmentada e informa ao executor que o resultado da operação encontra-se na tabela temporária criada.

Na versão proposta para este trabalho, não foi planejada a implementação de nenhum algoritmo de otimização de consultas distribuídas. Desta forma, a alteração no processador de consultas se resumiu em distribuir a consulta no *grid* e retornar uma espécie de junção dos dados retornados pelos sítios.

6.5 Módulos alterados

Diversos módulos do PostgreSQL foram alterados para que as funcionalidades propostas neste trabalho pudessem ser implementadas. Módulos como bootstrap (criador dos catálogos),

processador de consultas e o executor. A figura 6.2 destaca os módulos alterados.

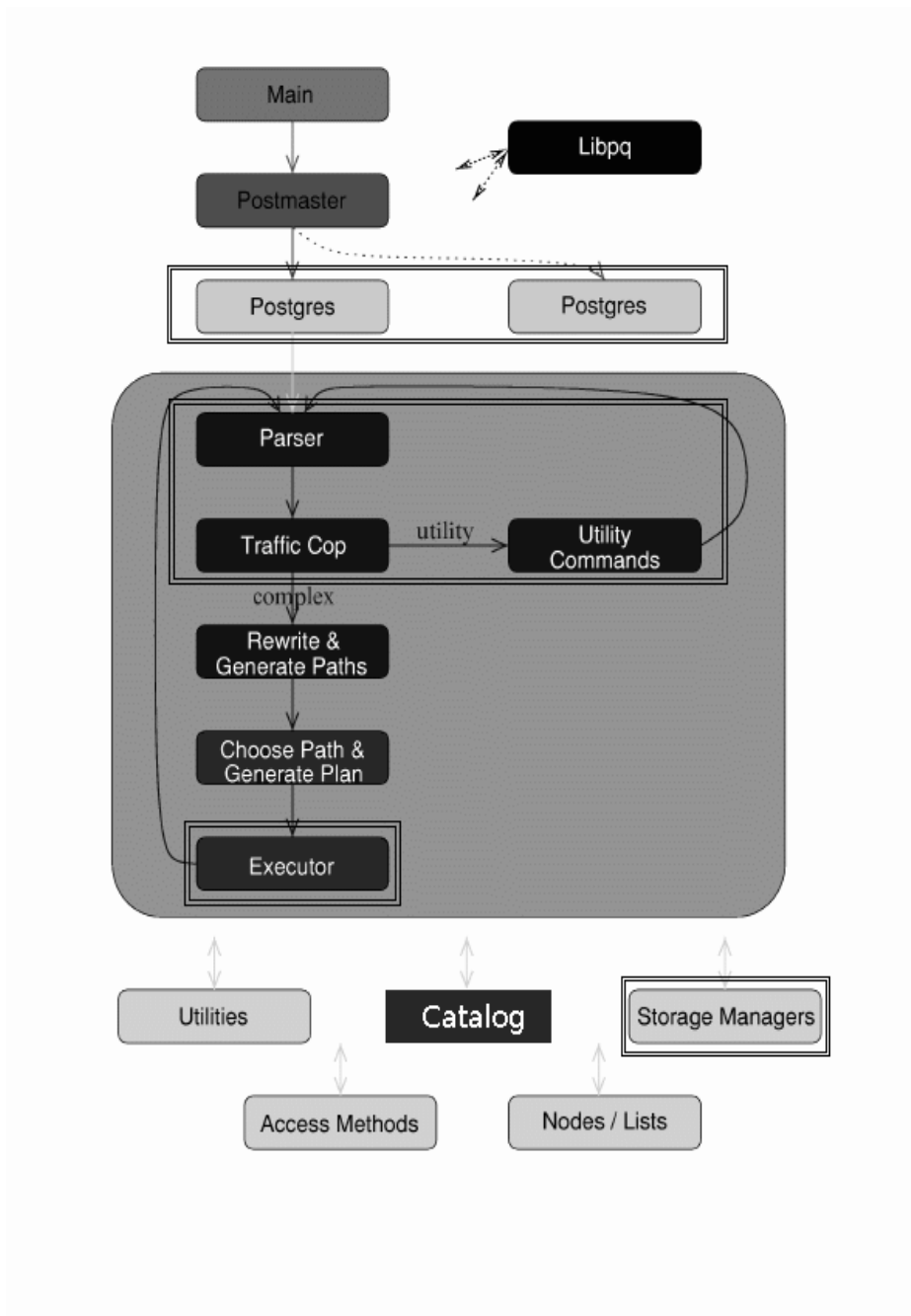


Figura 6.2: Módulos alterados do PostgreSQL

Os módulos *Postgres* e *Executor* foram alterados para recepcionar os comandos de atualização de dados e replicar os mesmos para os sites que possuem fragmentos envolvendo os dados sendo atualizados.

Os comandos para definição dos servidores, fragmentos e alocação exigiram alterações nos módulos *Parser*, *Traffic Cop*. (analisador e escalonador de comandos para os módulos) e *Utility Commands* (bibliotecas utilitárias para o *parser*), principalmente na gramática e no analisador

semântico do SGBD.

Finalmente, o módulo *Storage Managers* foi alterado para suportar as consultas distribuídas. Este módulo contém os mecanismos de gravação e retorno das tuplas. Antes do pgGrid, este módulos gravava e retornava apenas as tuplas armazenadas localmente. Então, foi implementado um filtro que verifica os catálogos do pgGrid e executa as consultas em todos os sites do sistema, de forma transparente para os demais módulos. Esta alteração foi importante para garantir que as funções de consulta e armazenamento distribuídos funcionem de forma independente dos outros módulos. Isso garante, por exemplo, que a verificação da integridade referencial seja mantida entre dados localizados em sites diferentes.

6.6 Comparativo dos resultados obtidos

Na seção 4.6 foi realizada uma comparação entre as principais tecnologias de BDD existentes no mercado. Para ilustrar os resultados obtidos com a implementação do pgGrid, a tabela é reapresentada abaixo com a adição do pgGrid na lista de SGBDD.

Característica	MySQL Cluster	Oracle RAC	IBM DB2	pgCluster	pgGrid
Consultas distribuídas	Sim	Sim	Sim	Não	Sim
Tecnologia multi-mestre	Sim	Sim	Sim	Sim	Sim
Replicação parcial	Não	Sim	Sim	Sim	Sim
Replicação síncrona	Sim	Sim	Sim	Sim	Sim
Replicação assíncrona	Não	Não	Não	Não	Não
Fragmentação	Sim	Sim	Sim	Não	Sim
Def. de regras de frag.	Não	Sim	Não	Não	Sim
Integridade ref. dist.	Não	Não	Não	Não	Sim
Protocolo de conf. dist.	2PC	2PC	2PC	2PC	2PC

Analisando a tabela, pode-se notar que as deficiências do pgCluster perante os demais gerenciadores são tratadas pelo pgGrid. Além disso, o fato do pgGrid permitir a definição da localização dos dados e manter a integridade referencial distribuída são características inovadoras que certamente representarão vantagens na adoção da tecnologia.

6.7 Repositório do projeto

PgGrid é um software de código-fonte aberto, disponível pela licença GPL e está disponível para download no site: <http://pgfoundry.org/projects/pggrid/>

6.8 Projetos relacionados

Existem diversos projetos relacionados à replicação de dados no PostgreSQL. Abaixo estão listados os projetos mais famosos:

- pgPool (*master-slave*);
- Slony (*master-slave* e replicação baseada em gatilhos);
- Bucardo (replicação assíncrona);
- SequoiaDB (replicação heterogênea).

7 *Caso de uso*

Para demonstrar as funcionalidades do pgGrid, um esquema de BDD foi montado. O esquema armazena dados sobre produtos produzidos no estado de Santa Catarina.

Cinco *sites* pertecem ao sistema, localizados nas principais cidades do estado:

- Florianópolis (FLN)
- Joinville (JVL)
- Blumenau (BLU)
- Criciúma (CRI)
- Chapecó (XAP)

As regras de armazenamento são simples: cada cidade armazena seu cadastro e os produtos produzidos em sua região. A capital (FLN), além dos dados de seus produtos, armazena o cadastro de todas as cidades do estado.

Os produtos possuem como atributos: código, nome e cidade de origem. As cidades possuem código, nome e a distância (em quilômetros) até a capital.

O modelo está representado pelo diagrama ER na figura 7.1.

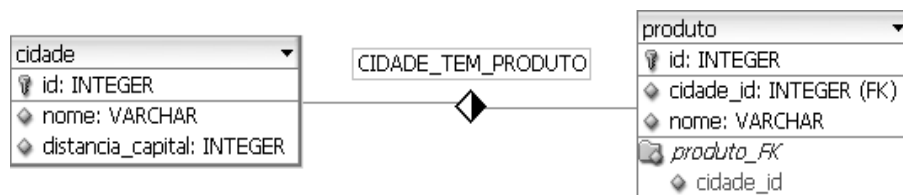


Figura 7.1: Modelo ER referente ao caso de uso do pgGrid

Para implementar o modelo sobre o pgGrid, as seguintes etapas devem ser executadas:

1. Inicialização e configuração dos sites;
2. Definição do esquema de dados;
3. Definição e alocação dos fragmentos;
4. Inserção dos dados;
5. Testes das consultas.

7.1 Inicialização e configuração dos sites

Todos os sites foram armazenados na mesma máquina, o que diferencia um do outro é a porta TCP que o servidor vai responder.

Para inicializar os sites, os seguintes comandos foram executados no diretório de instalação do pgGrid:

```
bin/initdb fln
bin/initdb jvl
bin/initdb blu
bin/initdb cri
bin/initdb xap
```

O BDD foi criado com o seguinte comando:

```
CREATE DATABASE SC;
```

Depois, conectando no BD da capital, os sítios foram criados desta forma:

```
sc=# CREATE SERVER FLN HOST "127.0.0.1" PORT 5432 RECOVERY PORT 8001;
CREATE PGGRID SERVER
sc=# CREATE SERVER JVL HOST "127.0.0.1" PORT 5433 RECOVERY PORT 8002;
CREATE PGGRID SERVER
sc=# CREATE SERVER BLU HOST "127.0.0.1" PORT 5434 RECOVERY PORT 8003;
CREATE PGGRID SERVER
sc=# CREATE SERVER CRI HOST "127.0.0.1" PORT 5435 RECOVERY PORT 8004;
CREATE PGGRID SERVER
sc=# CREATE SERVER XAP HOST "127.0.0.1" PORT 5436 RECOVERY PORT 8005;
CREATE PGGRID SERVER
```

7.2 Definição do esquema de dados

O esquema de dados consiste na duas relações: *cidade* e *produto*. As mesmas foram criadas com o seguinte código DML:

```

/*Criação da relação cidade*/
sc=# CREATE TABLE CIDADE (ID INT, NOME VARCHAR, DISTANCIA_CAPITAL INT);
CREATE TABLE
sc=# ALTER TABLE CIDADE ADD CONSTRAINT PK_CIDADE PRIMARY KEY (ID);
NOTICE: ALTER TABLE / ADD PRIMARY KEY will create implicit index
"pk_cidade" for table "cidade"
ALTER TABLE

/*Criação da relação produto*/
sc=# CREATE TABLE PRODUTO (ID INT, NOME VARCHAR, ID_CIDADE_ORIGEM INT);
CREATE TABLE
sc=# ALTER TABLE PRODUTO ADD CONSTRAINT PK_PRODUTO PRIMARY KEY (ID);
NOTICE: ALTER TABLE / ADD PRIMARY KEY will create implicit index
"pk_produto" for table "produto"
ALTER TABLE
sc=# ALTER TABLE PRODUTO ADD CONSTRAINT FK_PRODUTO_CIDADE FOREIGN
sc=# KEY (ID_CIDADE_ORIGEM) REFERENCES CIDADE (ID);
ALTER TABLE

```

7.3 Definição e alocação dos fragmentos

Os fragmentos são baseados no identificador da cidade que armazena os dados. Portanto, é necessário definir, de antemão, qual número vai representar cada cidade. A tabela abaixo mostra a alocação do identificador de cada cidade.

Identificador	Cidade
1	Florianópolis
2	Joinville
3	Blumenau
4	Criciúma
5	Chapecó

Com os identificadores definidos, definir os fragmentos da relação *produto* foi uma tarefa fácil, visto que esta relação é baseada apenas no identificador da cidade de origem. O trecho de código abaixo mostra como a relação *produto* foi fragmentada.

```
CREATE FRAGMENT PRODUTO_FLN ON PRODUTO WHERE ID_CIDADE_ORIGEM=1;
PLACE PRODUTO_FLN ON FLN;
CREATE FRAGMENT PRODUTO_JVL ON PRODUTO WHERE ID_CIDADE_ORIGEM=2;
PLACE PRODUTO_JVL ON JVL;
CREATE FRAGMENT PRODUTO_BLU ON PRODUTO WHERE ID_CIDADE_ORIGEM=3;
PLACE PRODUTO_BLU ON BLU;
CREATE FRAGMENT PRODUTO_CRI ON PRODUTO WHERE ID_CIDADE_ORIGEM=4;
PLACE PRODUTO_CRI ON CRI;
CREATE FRAGMENT PRODUTO_XAP ON PRODUTO WHERE ID_CIDADE_ORIGEM=5;
PLACE PRODUTO_XAP ON XAP;
```

A relação *cidade* possui a mesma regra para fragmentação, exceto pelo fato da capital armazenar todas as cidades. O código a seguir demonstra esta separação.

```
CREATE FRAGMENT CIDADE_FLN ON CIDADE;
PLACE CIDADE_FLN ON FLN;
CREATE FRAGMENT CIDADE_JVL ON CIDADE WHERE ID=2;
PLACE CIDADE_JVL ON JVL;
CREATE FRAGMENT CIDADE_BLU ON CIDADE WHERE ID=3;
PLACE CIDADE_BLU ON BLU;
CREATE FRAGMENT CIDADE_CRI ON CIDADE WHERE ID=4;
PLACE CIDADE_CRI ON CRI;
CREATE FRAGMENT CIDADE_XAP ON CIDADE WHERE ID=5;
PLACE CIDADE_XAP ON XAP;
```

7.4 Inserção dos dados

A inserção dos dados confirmou o funcionamento da fragmentação imposta pelo pgGrid no pgCluster. Depois de cada comando “INSERT”, foi possível verificar no log do servidor de replicação que os dados foram inseridos nos *sites* de acordo com as regras dos fragmentos.

Também é possível verificar os dados contidos em cada servidor desfazendo a configuração do cluster e iniciando cada servidor no modo centralizado. Tal estratégia é muito importante

para melhorar a disponibilidade dos dados no cluster, pois quando um servidor fica indisponível, os demais devem continuar respondendo as consultas possíveis.

Cada site armazenou o cadastro da cidade que representa e os produtos da mesma. O script a seguir contém os comandos executados para preencher as relações *cidade* e *produto*.

```
INSERT INTO CIDADE (ID, NOME, DISTANCIA_CAPITAL) VALUES (1, 'FLORIANOPOLIS', 0);
INSERT INTO CIDADE (ID, NOME, DISTANCIA_CAPITAL) VALUES (2, 'JOINVILLE', 170);
INSERT INTO CIDADE (ID, NOME, DISTANCIA_CAPITAL) VALUES (3, 'BLUMENAU', 150);
INSERT INTO CIDADE (ID, NOME, DISTANCIA_CAPITAL) VALUES (4, 'CRICIUMA', 190);
INSERT INTO CIDADE (ID, NOME, DISTANCIA_CAPITAL) VALUES (5, 'CHAPECO', 550);
```

```
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (1, 'TAINHA', 1);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (2, 'OSTRAS', 1);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (3, 'REFRIGERADOR', 2);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (4, 'MOTOR ELETRICO', 2);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (5, 'TECIDO', 3);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (6, 'SOFTWARE', 3);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (7, 'CERAMICA', 4);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (8, 'CARVAO MINERAL', 4);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (9, 'MILHO', 5);
INSERT INTO PRODUTO (ID, NOME, ID_CIDADE_ORIGEM) VALUES (10, 'SUINO', 5);
```

7.5 Testes das consultas

Como requisito básico para atestar a capacidade de fragmentação e reconstrução das relações fragmentadas, a consulta distribuída dos dados armazenados no cluster foi executada com sucesso. A seguir está a consulta utilizada e o resultado da mesma:

```
sc=#SELECT CIDADE.NOME as nome_cidade, PRODUTO.NOME as nome_produto
sc=#FROM PRODUTO, CIDADE WHERE PRODUTO.ID_CIDADE_ORIGEM = CIDADE.ID;
```

nome_cidade	nome_produto
FLORIANOPOLIS	TAINHA
FLORIANOPOLIS	OSTRAS
JOINVILLE	REFRIGERADOR

JOINVILLE		MOTOR ELETRICO
BLUMENAU		TECIDO
BLUMENAU		SOFTWARE
CRICIUMA		CERAMICA
CRICIUMA		CARVAO MINERAL
CHAPECO		MILHO
CHAPECO		SUINO

A consulta foi executada no site de Florianópolis. Como pode ser notado, a consulta faz a junção das duas relações do modelo. Sendo assim, foi necessário ao pgGrid obter os dados de produtos distribuídos nos sites e depois executar a operação de junção com a relação de cidades armazenada localmente.

7.6 Testes de desempenho

Os testes executados no caso de uso abordaram principalmente as garantias da integridade e reconstrução dos dados distribuídos no sistema. Esta seção examina o desempenho das operações dentro do sistema distribuído utilizado no caso de uso, fazendo uma comparação com o desempenho do pgCluster com a mesma configuração e massa de dados.

Buscando resultados mais precisos, foi gerada uma massa de dados com 500000 produtos que foi distribuída proporcionalmente nos sites do sistema, de acordo com as regras de fragmentação definidas no capítulo 7.

Os parâmetros utilizados nos testes estão listados a seguir e correspondem ao tempo das operações, carga dos servidores, rede e dos dispositivos de armazenamento.

- **Tempo para inserção/atualização/exclusão na relação *CIDADE*:** o tempo desde a submissão do comando *INSERT/UPDATE/DELETE* até o retorno do servidor;
- **Tempo para inserção/atualização/exclusão na relação *PRODUTO*:** o tempo desde a submissão do comando *INSERT/UPDATE/DELETE* até o retorno do servidor;
- **Tempo para consulta na relação *CIDADE*:** Tempo desde a submissão do comando *SELECT* para a relação *CIDADE* até o retorno dos dados pelo servidor;
- **Tempo para consulta na relação *PRODUTO*:** Tempo desde a submissão do comando *SELECT* para a relação *PRODUTO* até o retorno dos dados pelo servidor;

- **Tempo para busca de um produto específico:** Tempo desde a submissão do comando *SELECT * from PRODUTO WHERE nome = 'PROD500000'* até o retorno dos dados pelo servidor;
- **Tempo para consulta distribuída:** Tempo desde a submissão da consulta envolvendo a junção das duas relações do esquema até o retorno dos dados pelo servidor.
- **Carga de trabalho média dos servidores durante a consulta:** Média aritmética sobre a maior carga de trabalho dos servidores durante a execução da consulta distribuída;
- **Quantidade de pacotes de rede transmitidos durante inserção:** Quantidade de pacotes que foram enviados/recebidos pelos servidores do cluster durante a execução de uma inserção na relação *PRODUTO*.
- **Quantidade de pacotes de rede transmitidos durante a consulta distribuída:** Quantidade de pacotes que foram enviados/recebidos pelos servidores do cluster durante a execução da consulta envolvendo as duas relações do modelo.
- **Espaço em disco médio ocupado pelo banco de dados:** Média aritmética do espaço em disco ocupado pelos arquivos de dados em cada servidor do sistema.

Para efeitos de padronização, todas as consultas no sistema gerenciado pelo pgGrid foram submetidas ao servidor de Florianópolis.

Os dados foram distribuídos em três servidores com a seguinte configuração de hardware:

1. Servidor FLN

- Processador: Intel Pentium IV 3.06 GHz
- Memória: 3GB DDR2 800MHz
- Controlador de disco: SATA II
- Interface de rede: Gigabit Ethernet

2. Servidor JVL/BLU

- Processador: Intel Pentium IV 3.06 GHz
- Memória: 1GB DDR2 800MHz
- Controlador de disco: SATA II
- Interface de rede: Fast Ethernet

3. Servidor CRI/XAP

- Processador: Intel Celeron D 800 MHz
- Memória: 512MB DDR2 800MHz
- Controlador de disco: IDE
- Interface de rede: *Fast Ethernet*

Os servidores foram conectados através de um roteador utilizando rede TCP/IP e *Fast Ethernet* na camada de enlace, com taxa de transferência máxima de 100 Mbps.

A tabela 7.1 contém os resultados dos testes de desempenho utilizando o mesmo conjunto de servidores para o pgGrid/pgCluster.

Parâmetro	pgCluster	pgGrid
Tempo para inserção/atualização/exclusão na relação CIDADE	1.8s	0.9s
Tempo para inserção/atualização/exclusão na relação PRODUTO	2.7s	1.1s
Tempo para consulta na relação CIDADE	1s	1s
Tempo para consulta na relação PRODUTO	5.3s	16.5s
Tempo para busca de um produto específico (<i>PROD500000</i>)	4s	2.3s
Tempo para consultas distribuídas	5.7s	19.3s
Carga de trabalho média dos servidores durante a consulta	20%	38%
Quantidade de pacotes de rede transmitidos durante inserção	16	4
Quantidade de pacotes de rede transmitidos durante a consulta	0	400020
Espaço em disco médio ocupado pelo banco de dados	30MB	7.57MB

Tabela 7.1: Resultados dos testes de desempenho executados sobre o cluster

Pode-se notar, através dos dados obtidos, que a fragmentação nos dados melhorou as operações de atualização, pois não é necessário atualizar todos os sites do sistema quando um item de dado é modificado. No entanto, a maioria das operações de consulta envolvendo mais de um site tiveram seu desempenho degradado devido à necessidade de juntar os dados distribuídos entre os servidores. No caso do pgCluster, com os dados totalmente replicados, apenas uma consulta local resolve o problema.

A grande melhoria notada pela fragmentação se refere ao espaço em disco médio utilizado nos servidores. Com a remoção das réplicas desnecessárias, os recursos utilizados em cada servidor podem ser configurados de uma forma mais otimizada para atender às necessidades dos seus usuários.

É importante ressaltar que o desempenho do sistema depende muito da arquitetura do mesmo. Neste caso, grandes melhorias foram percebidas com o uso da fragmentação, mas

algumas consultas executadas no pgGrid obtiveram desempenho muito pior do que no pgCluster. Portanto, é necessário analisar cuidadosamente o que deve ser replicado e o que deve ser fragmentado e avaliar as consequências desta decisão. A principal motivação deste trabalho é permitir que os arquitetos de BD decidam como seus sistemas devem se comportar e otimizar ao máximo a utilização dos recursos utilizados pelo mesmo.

As figuras 7.2, 7.3, 7.4, 7.5, 7.6 demonstram, graficamente, os resultados obtidos nos testes de desempenho.

Medição dos tempos das operações

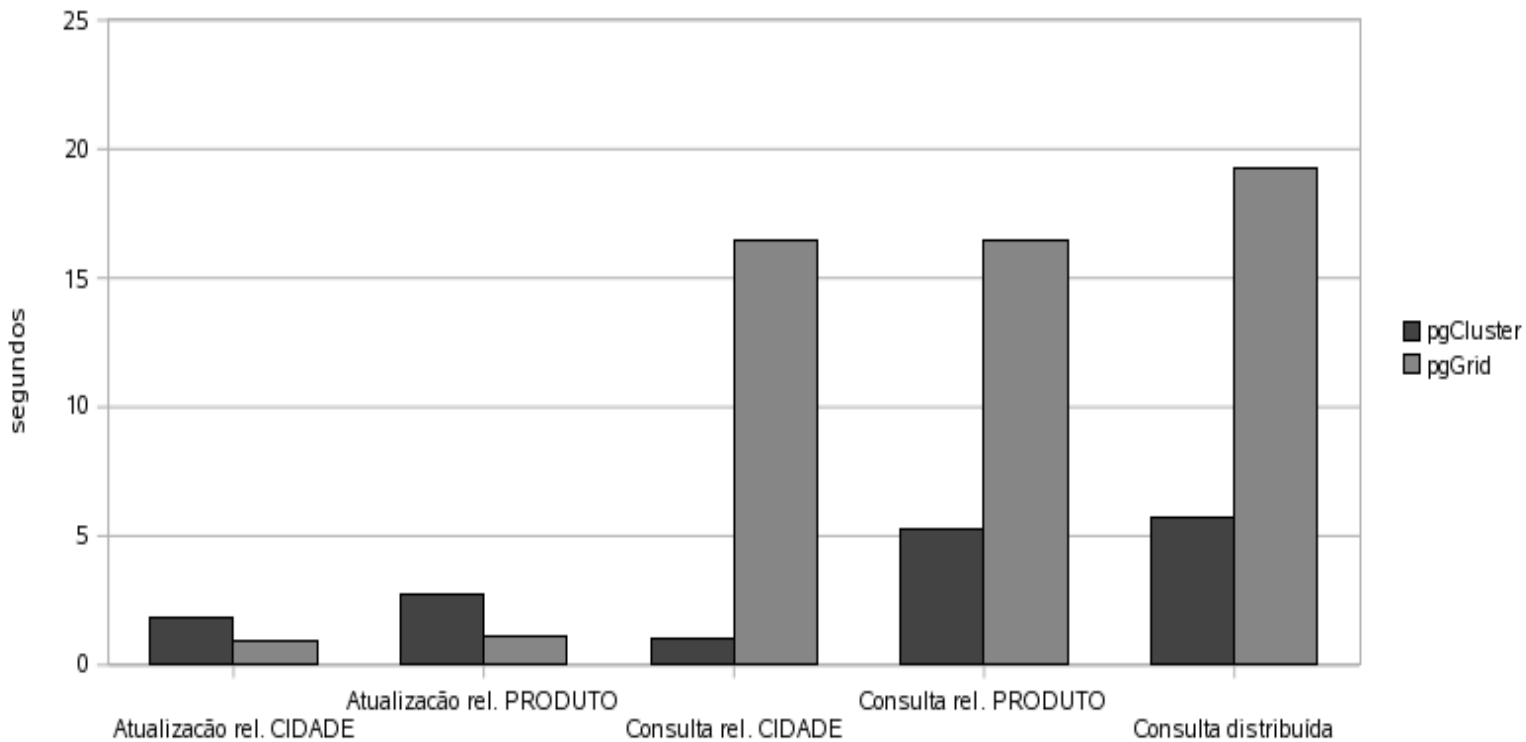


Figura 7.2: Medição dos tempos das operações

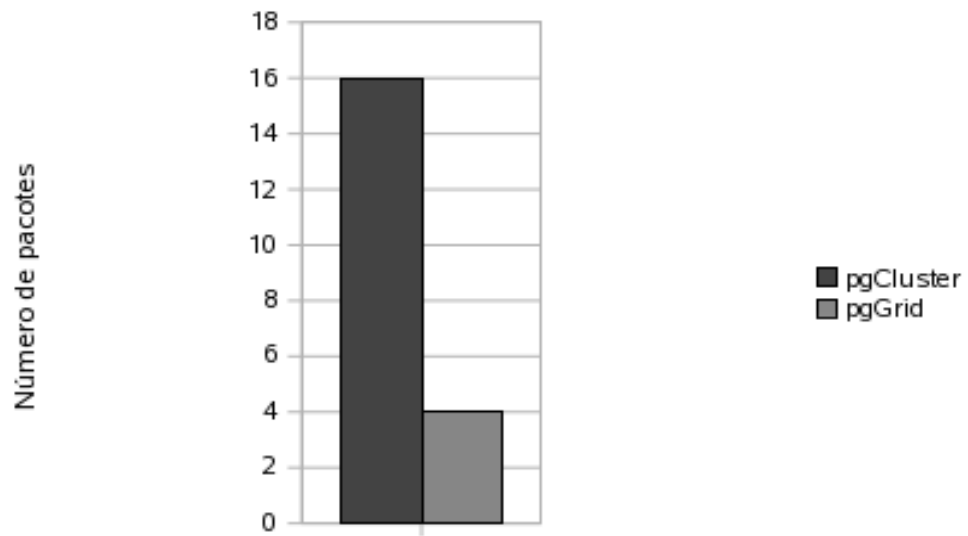


Figura 7.3: Quantidade de pacotes de rede transmitidos durante inserção

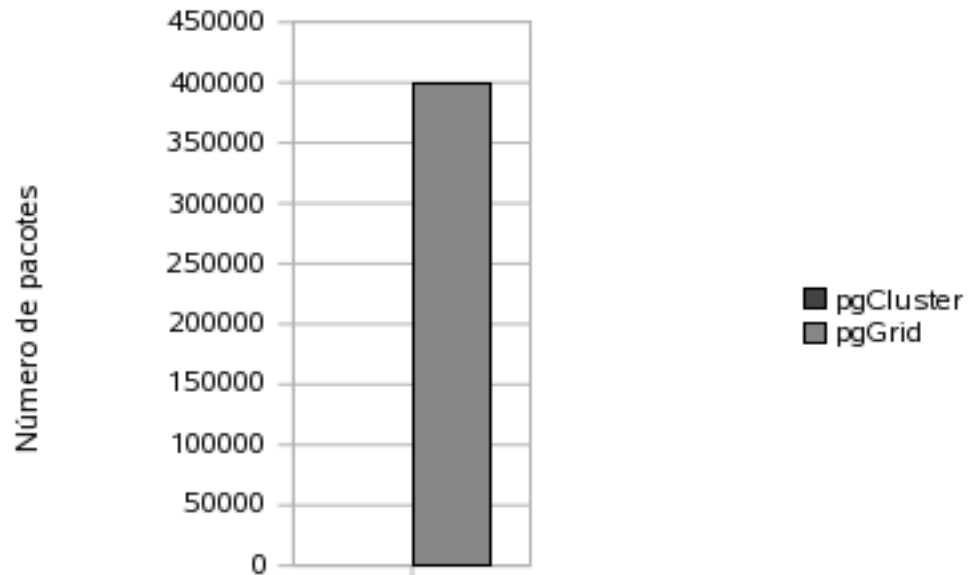


Figura 7.4: Quantidade de pacotes de rede transmitidos durante a consulta distribuída

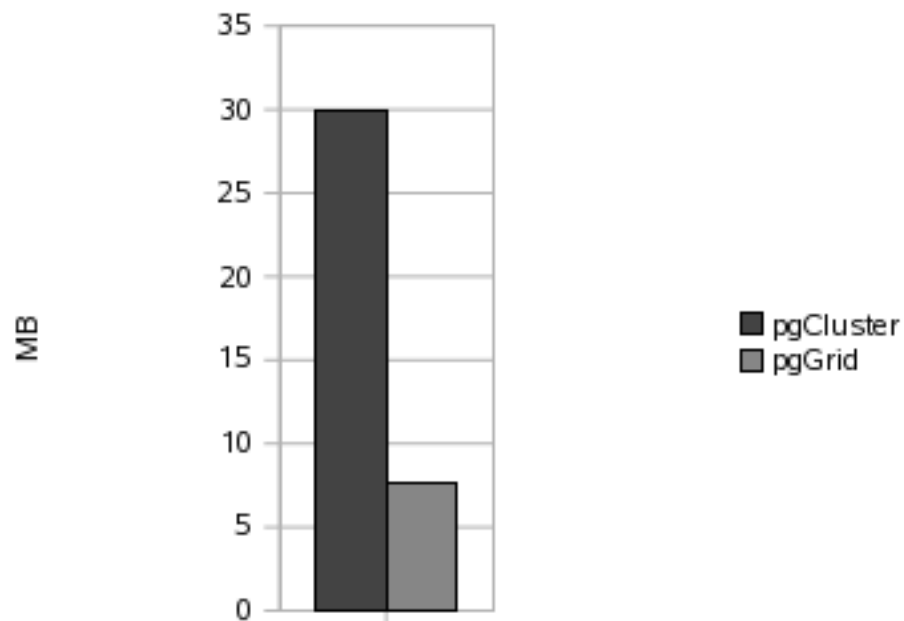


Figura 7.5: Espaço em disco médio ocupado pelo banco de dados

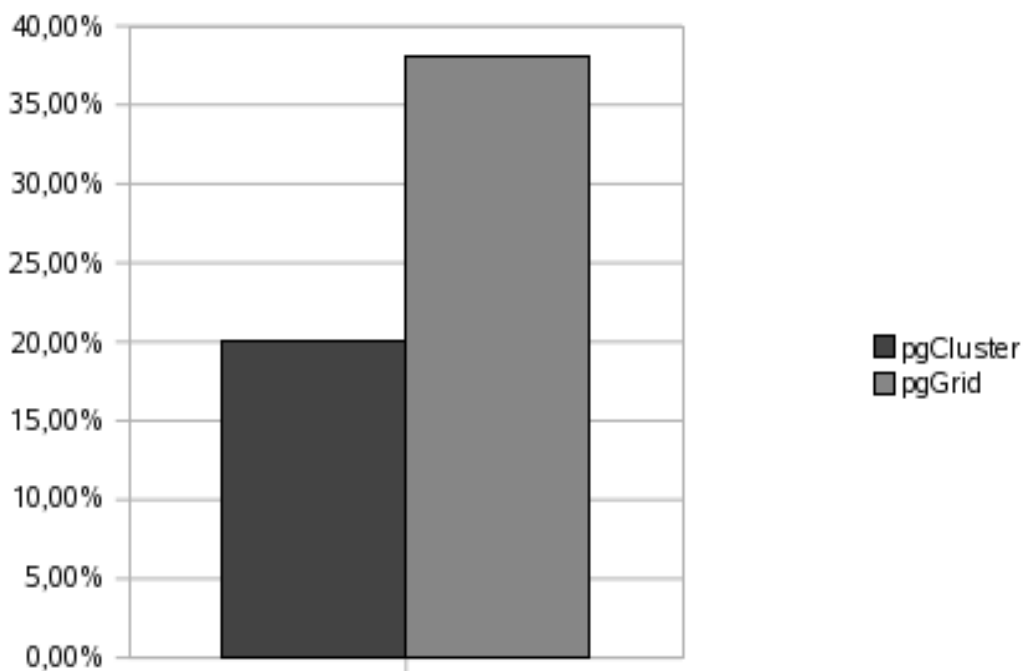


Figura 7.6: Carga de trabalho média dos servidores durante a consulta

8 *Conclusão e trabalhos futuros*

Com a crescente necessidade de compartilhamento de informações, está surgindo uma demanda muito grande no armazenamento e centralização das mesmas. No entanto, com volumes muito grandes de dados, a centralização torna o processo de recuperação e carga dos dados ineficiente.

Para resolver tal problema, os sistemas distribuídos entram em jogo fornecendo soluções escaláveis e paralelizáveis. Este trabalho possui o intuito de propor uma solução deste tipo no contexto de banco de dados relacionais e objetos-relacionais.

No decorrer do trabalho, a escolha do PostgreSQL como base para a implementação se tornou muito vantajosa devido ao nível de estabilidade e amadurecimento de suas funcionalidades. Assim como pela qualidade e legibilidade do seu código-fonte.

A proposta apresentada no início do trabalho mostrou-se viável no decorrer do desenvolvimento do mesmo. Tal viabilidade foi ilustrada pelo caso de uso e pelos testes de desempenho. Existem muitas otimizações que devem ser implementadas e o software deve passar por testes mais rigorosos antes de ser colocado em produção em casos reais.

Como próximo passo planeja-se implementar as otimizações de consultas que o ambiente distribuído possibilita, tais como a paralelização de subconsultas. Também pretende-se aperfeiçoar os servidores de modo a melhorar o protocolo de comunicação entre os mesmos e remover a necessidade de servidores de replicação.

A versão atual do pgGrid (0.1) possui o código-fonte aberto, pode ser utilizada livremente nos termos da licença *GPL* e está disponível no *site* <<http://pggrid.projects.postgresql.org/>>.

8.1 **Trabalhos futuros**

Este trabalho definiu as funções básicas para a ferramenta pgGrid. As próximas etapas visando a melhoria e otimização das suas funções são:

- Otimização de consultas: aplicar otimizações baseadas em reduções (ver seção 5.5) nos comandos *UPDATE* e *DELETE*, pois os mesmos estão sendo enviados para todos os servidores que possuem fragmentos da relação sendo atualizada;
- Remoção da necessidade de servidores de replicação (camada adicional do pgCluster para balanceamento de carga), pois a mesma está afetando o tempo de resposta das operações;
- Adição, na sintaxe SQL, do padrão “tabela@site” para consultar dados de um site específico;
- Criação de pacotes binários para diversas plataformas: compilar e disponibilizar o pg-Grid no formato binário (atualmente somente o código-fonte está disponível). Os pacotes devem ser disponibilizados no *site* da ferramenta;
- Implementação da replicação assíncrona para melhorar a disponibilidade do sistema no caso de falha dos seus componentes. Ver seção 3.3.3.

Referências Bibliográficas

- BEDOYA, Hernando et al. *Advanced Functions and Administration on DB2 Universal Database for iSeries* Disponível em <<http://www.redbooks.ibm.com/abstracts/sg244249.html>>. Acesso em jan. 2009. ISBN 0738422320, Rochester: 2001
- BUYYA, Rajkumar. *High Performance Cluster Computing: Architectures and Systems*. Vol. 1. New Jersey: Prentice Hall, 1999. ISBN 0130137847, 881 p.
- CHANG, Fay et al. Bigtable: A Distributed Storage System for Structured Data In: 7th USENIX SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, 2006, Seattle: OSDI 2006.
- COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim *Distributed systems: concept and design*. 4 ed. Boston: Addison Wesley, 2005, ISBN 0321263545
- FOSTER, Ian; KELESSELMAN, Carl. *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco: Morgan Kaufmann, 1998. ISBN 1558604758, 701 p.
- GIUSEPPE DE, G., L. Domenico, et al. Tackling inconsistencies in data integration through source preferences. In: 2004 INTERNATIONAL WORKSHOP ON INFORMATION QUALITY IN INFORMATION SYSTEMS, 2004, Paris. *Proceedings...* Paris, 2004.
- GOPALAKRISHNAN, K *Oracle Database 10g Real Application Clusters Handbook*. Columbus: McGraw-Hill Osborne Media (Oracle Press), 2006, ISBN 9780071465090
- JIA, Weijia; ZHOU, Wanlei *Distributed network systems: from concepts to implementations*. New York: Springer, 2005
- MYSQL AB. *MySQL 5.0 Reference Manual*. Disponível em <<http://dev.mysql.com/doc/refman/5.0/>>. Acesso em out. 2007
- OZSU, M. Tamer; VALDURIEZ, Patrick *Principles of distributed database systems*. New Jersey: Prentice Hall, 1999. ISBN 0136597076, 666 p.
- PFISTER, Gregory *In Search of Clusters*. 4 ed. New Jersey: Prentice Hall, 1997. ISBN 0138997098, 608 p.
- POSTGRESQL DEVELOPMENT GROUP *About PostgreSQL*. Disponível em <<http://www.postgresql.org>>. Acesso em set. 2007
- TANENBAUM, Andrew S. *Computer Networks*. 4 ed. New Jersey: Prentice Hall, 2002. ISBN 0130661023, 912 p.
- TANENBAUM, Andrew S; STEEN, Maarten van *Distributed Systems: Principles and Paradigms*. 2 ed. New Jersey: Pearson Education, 2008. ISBN 0136135536, 704 p.

WIKIPEDIA. *Wikipedia. verbete: Cluster*. Disponível em:
<<http://pt.wikipedia.org/wiki/Cluster>>. Acesso em: 17 jan 2009.

WIKIPEDIA. *Wikipedia. verbete: Computação em grelha*. Disponível em:
<http://pt.wikipedia.org/wiki/Computacao_em_grelha>. Acesso em: 17 jan 2009.

APÊNDICE A – Patch para o pgCluster

Index: pgGrid0.1/config.log

```

=====
--- pgGrid0.1/config.log      (revisão 53)
+++ pgGrid0.1/config.log      (cópia de trabalho)
@@ -4,7 +4,7 @@
    It was created by PostgreSQL configure 8.3.0, which was
    generated by GNU Autoconf 2.59.  Invocation command line was

- $ ./configure
+ $ ./configure --prefix=/usr/local/pggrid/

## ----- ##
## Platform. ##
@@ -12,9 +12,9 @@

hostname = linux-v0z9
uname -m = x86_64
-uname -r = 2.6.25.18-0.2-default
+uname -r = 2.6.25.20-0.1-default
uname -s = Linux
-uname -v = #1 SMP 2008-10-21 16:30:26 +0200
+uname -v = #1 SMP 2008-12-12 20:30:38 +0100

/usr/bin/uname -p = unknown
/bin/uname -X      = unknown
@@ -310,7 +310,7 @@
configure:5491: result: yes

```

```

configure:5502: checking for library containing setproctitle
configure:5532: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lm >&5
-/tmp/cckBINGA.o: In function 'main':
+/tmp/cculspcL.o: In function 'main':
confctest.c:(.text+0x7): undefined reference to 'setproctitle'
collect2: ld returned 1 exit status
configure:5538: $? = 1
@@ -347,7 +347,7 @@
|   return 0;
| }
configure:5587: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lutil -lm >&5
-/tmp/ccc6ka1P.o: In function 'main':
+/tmp/ccaVQ8Ck.o: In function 'main':
confctest.c:(.text+0x7): undefined reference to 'setproctitle'
collect2: ld returned 1 exit status
configure:5593: $? = 1
@@ -386,7 +386,7 @@
configure:5621: result: no
configure:5628: checking for library containing dlopen
configure:5658: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lm >&5
-/tmp/ccqyY7B6.o: In function 'main':
+/tmp/ccIiZ77n.o: In function 'main':
confctest.c:(.text+0x7): undefined reference to 'dlopen'
collect2: ld returned 1 exit status
configure:5664: $? = 1
@@ -441,7 +441,7 @@
configure:5873: result: none required
configure:5880: checking for library containing shl_load
configure:5910: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith

```

```

-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE    confctest.c -ldl -lm >&5
-/tmp/cccidRqX.o: In function 'main':
+/tmp/ccG960I9.o: In function 'main':
  confctest.c:(.text+0x7): undefined reference to 'shl_load'
  collect2: ld returned 1 exit status
  configure:5916: $? = 1
@@ -525,7 +525,7 @@
  configure:6256: result: none required
  configure:6263: checking for library containing crypt
  configure:6293: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE    confctest.c -ldl -lm >&5
-/tmp/cc05wkpm.o: In function 'main':
+/tmp/ccud0oiS.o: In function 'main':
  confctest.c:(.text+0x7): undefined reference to 'crypt'
  collect2: ld returned 1 exit status
  configure:6299: $? = 1
@@ -3497,9 +3497,9 @@
  configure:15635: result: yes
  configure:15547: checking for getpeereid
  configure:15604: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE    confctest.c -lz -lreadline -lcrypt -ldl -lm >&5
-/tmp/cc04Bka6.o: In function 'main':
+/tmp/cc6RIdLo.o: In function 'main':
  confctest.c:(.text+0x9): undefined reference to 'getpeereid'
-/tmp/cc04Bka6.o:(.data+0x0): undefined reference to 'getpeereid'
+/tmp/cc6RIdLo.o:(.data+0x0): undefined reference to 'getpeereid'
  collect2: ld returned 1 exit status
  configure:15610: $? = 1
  configure: failed program was:
@@ -3649,9 +3649,9 @@
  configure:15635: result: yes
  configure:15547: checking for pstat

```

```

configure:15604: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lz -lreadline -lcrypt -ldl -lm >&5
-/tmp/ccemM2kn.o: In function 'main':
+/tmp/ccGa3re3.o: In function 'main':
confctest.c:(.text+0x9): undefined reference to 'pstat'
-/tmp/ccemM2kn.o:(.data+0x0): undefined reference to 'pstat'
+/tmp/ccGa3re3.o:(.data+0x0): undefined reference to 'pstat'
collect2: ld returned 1 exit status
configure:15610: $? = 1
configure: failed program was:
@@ -3785,9 +3785,9 @@
configure:15635: result: yes
configure:15547: checking for setproctitle
configure:15604: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lz -lreadline -lcrypt -ldl -lm >&5
-/tmp/cc86SpsL.o: In function 'main':
+/tmp/cce97Req.o: In function 'main':
confctest.c:(.text+0x9): undefined reference to 'setproctitle'
-/tmp/cc86SpsL.o:(.data+0x0): undefined reference to 'setproctitle'
+/tmp/cce97Req.o:(.data+0x0): undefined reference to 'setproctitle'
collect2: ld returned 1 exit status
configure:15610: $? = 1
configure: failed program was:
@@ -4672,9 +4672,9 @@
configure:16831: result: yes
configure:16743: checking for strlcat
configure:16800: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lz -lcrypt -ldl -lm >&5
-/tmp/ccRPzDKi.o: In function 'main':
+/tmp/ccgXd2q2.o: In function 'main':
confctest.c:(.text+0x9): undefined reference to 'strlcat'
-/tmp/ccRPzDKi.o:(.data+0x0): undefined reference to 'strlcat'

```



```

+/tmp/ccgXd2q2.o:(.data+0x0): undefined reference to 'strlcat'
collect2: ld returned 1 exit status
configure:16806: $? = 1
configure: failed program was:
@@ -4830,9 +4830,9 @@
configure:16831: result: no
configure:16743: checking for strlcpy
configure:16800: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lz -lcrypt -ldl -lm >&5
-/tmp/ccK7s60y.o: In function 'main':
+/tmp/ccizZICe.o: In function 'main':
confctest.c:(.text+0x9): undefined reference to 'strlcpy'
-/tmp/ccK7s60y.o:(.data+0x0): undefined reference to 'strlcpy'
+/tmp/ccizZICe.o:(.data+0x0): undefined reference to 'strlcpy'
collect2: ld returned 1 exit status
configure:16806: $? = 1
configure: failed program was:
@@ -5113,7 +5113,7 @@
configure:17898: result: yes
configure:17914: checking for optreset
configure:17935: gcc -o confctest -O2 -Wall -Wmissing-prototypes -Wpointer-arith
-Winline -Wdeclaration-after-statement -Wendif-labels -fno-strict-aliasing
-D_GNU_SOURCE confctest.c -lz -lreadline -lcrypt -ldl -lm >&5
-/tmp/ccEi2Adh.o: In function 'main':
+/tmp/cc0LI9Xg.o: In function 'main':
confctest.c:(.text+0x2): undefined reference to 'optreset'
collect2: ld returned 1 exit status
configure:17941: $? = 1
@@ -6106,7 +6106,9 @@
config.status:761: creating GNUmakefile
config.status:761: creating src/Makefile.global
config.status:864: creating src/include/pg_config.h
+config.status:1256: src/include/pg_config.h is unchanged
config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h

```

```

+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
  config.status:1320: linking ./src/backend/port/tas/dummy.s to
src/backend/port/tas.s
  config.status:1320: linking ./src/backend/port/dynloader/linux.c to
src/backend/port/dynloader.c
  config.status:1320: linking ./src/backend/port/sysv_sema.c to
src/backend/port/pg_sema.c
@@ -6414,7 +6416,7 @@
  build_cpu='x86_64'
  build_os='linux-gnu'
  build_vendor='unknown'
-configure_args=''
+configure_args='--prefix=/usr/local/pggrid/'
  datadir='${prefix}/share'
  default_port='5432'
  docdir='${prefix}/doc'
@@ -6446,7 +6448,7 @@
  perl_embed_ldflags=''
  perl_privlibexp=''
  perl_useshrplib=''
-prefix='/usr/local/pgsql'
+prefix='/usr/local/pggrid/'
  program_transform_name='s,x,x,'
  python_additional_libs=''
  python_configdir=''
@@ -6614,3 +6616,111 @@
  #define USE_SYSV_SHARED_MEMORY 1

  configure: exit 0
+
+## ----- ##
+## Running config.status. ##
+## ----- ##
+
+This file was extended by PostgreSQL config.status 8.3.0, which was

```

```

+generated by GNU Autoconf 2.59.  Invocation command line was
+
+ CONFIG_FILES      =
+ CONFIG_HEADERS    =
+ CONFIG_LINKS      =
+ CONFIG_COMMANDS   =
+ $ ./config.status src/interfaces/ecpg/include/ecpg_config.h
+
+on linux-v0z9
+
+config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h
+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
+
+## ----- ##
+## Running config.status. ##
+## ----- ##
+
+This file was extended by PostgreSQL config.status 8.3.0, which was
+generated by GNU Autoconf 2.59.  Invocation command line was
+
+ CONFIG_FILES      =
+ CONFIG_HEADERS    =
+ CONFIG_LINKS      =
+ CONFIG_COMMANDS   =
+ $ ./config.status src/interfaces/ecpg/include/ecpg_config.h
+
+on linux-v0z9
+
+config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h
+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
+
+## ----- ##
+## Running config.status. ##
+## ----- ##
+

```

```

+This file was extended by PostgreSQL config.status 8.3.0, which was
+generated by GNU Autoconf 2.59.  Invocation command line was
+
+ CONFIG_FILES      =
+ CONFIG_HEADERS    =
+ CONFIG_LINKS      =
+ CONFIG_COMMANDS   =
+ $ ./config.status src/interfaces/ecpg/include/ecpg_config.h
+
+on linux-v0z9
+
+config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h
+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
+
+## ----- ##
+## Running config.status. ##
+## ----- ##
+
+This file was extended by PostgreSQL config.status 8.3.0, which was
+generated by GNU Autoconf 2.59.  Invocation command line was
+
+ CONFIG_FILES      =
+ CONFIG_HEADERS    =
+ CONFIG_LINKS      =
+ CONFIG_COMMANDS   =
+ $ ./config.status src/interfaces/ecpg/include/ecpg_config.h
+
+on linux-v0z9
+
+config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h
+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
+
+## ----- ##
+## Running config.status. ##
+## ----- ##

```

```

+
+This file was extended by PostgreSQL config.status 8.3.0, which was
+generated by GNU Autoconf 2.59.  Invocation command line was
+
+
+ CONFIG_FILES      =
+ CONFIG_HEADERS    =
+ CONFIG_LINKS      =
+ CONFIG_COMMANDS   =
+ $ ./config.status src/interfaces/ecpg/include/ecpg_config.h
+
+on linux-v0z9
+
+config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h
+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
+
+## ----- ##
+## Running config.status. ##
+## ----- ##
+
+This file was extended by PostgreSQL config.status 8.3.0, which was
+generated by GNU Autoconf 2.59.  Invocation command line was
+
+
+ CONFIG_FILES      =
+ CONFIG_HEADERS    =
+ CONFIG_LINKS      =
+ CONFIG_COMMANDS   =
+ $ ./config.status src/interfaces/ecpg/include/ecpg_config.h
+
+on linux-v0z9
+
+config.status:864: creating src/interfaces/ecpg/include/ecpg_config.h
+config.status:1256: src/interfaces/ecpg/include/ecpg_config.h is unchanged
Index: pgGrid0.1/config.status
=====
--- pgGrid0.1/config.status      (revisão 53)

```

```

+++ pgGrid0.1/config.status      (c3pia de trabalho)
@@ -307,7 +307,7 @@
    ac_cs_version="\
PostgreSQL config.status 8.3.0
    configured by ./configure, generated by GNU Autoconf 2.59,
-   with options \"\"
+   with options \"'--prefix=/usr/local/pggrid/'\"

Copyright (C) 2003 Free Software Foundation, Inc.
This config.status script is free software; the Free Software Foundation
@@ -385,8 +385,8 @@
fi

if $ac_cs_recheck; then
- echo "running /bin/sh ./configure " $ac_configure_extra_args " --no-create
--no-recursion" >&6
- exec /bin/sh ./configure $ac_configure_extra_args --no-create
--no-recursion
+ echo "running /bin/sh ./configure " '--prefix=/usr/local/pggrid/'
$ac_configure_extra_args " --no-create --no-recursion" >&6
+ exec /bin/sh ./configure '--prefix=/usr/local/pggrid/'
$ac_configure_extra_args --no-create --no-recursion
fi

for ac_config_target in $ac_config_targets
@@ -466,7 +466,7 @@
s,@PACKAGE_STRING@,PostgreSQL 8.3.0,;t t
s,@PACKAGE_BUGREPORT@,pgsql-bugs@postgresql.org,;t t
s,@exec_prefix@,{prefix},;t t
-s,@prefix@,/usr/local/pgsql,;t t
+s,@prefix@,/usr/local/pggrid/,;t t
s,@program_transform_name@,s,x,x,,;t t
s,@bindir@,{exec_prefix}/bin,;t t
s,@sbindir@,{exec_prefix}/sbin,;t t
@@ -488,7 +488,7 @@

```

```

s,@ECHO_N@,-n,;t t
s,@ECHO_T@,,;t t
s,@LIBS@,-lz -lreadline -lcrypt -ldl -lm ,;t t
-s,@configure_args@,,;t t
+s,@configure_args@,'--prefix=/usr/local/pggrid/';;t t
s,@build@,x86_64-unknown-linux-gnu,;t t
s,@build_cpu@,x86_64,;t t
s,@build_vendor@,unknown,;t t

```

Index: pgGrid0.1/src/include/pggrid/dquery.h

=====

```

--- pgGrid0.1/src/include/pggrid/dquery.h      (revisão 0)
+++ pgGrid0.1/src/include/pggrid/dquery.h      (revisão 224)
@@ -0,0 +1,8 @@

```

```

+#include "postgres.h"
+#include "access/heapam.h"
+#include "pggrid.h"
+#include "nodes/execnodes.h"
+
+int ExecCountForeignTuples(Index relid);
+
+Oid ExecClusterQuery(Index relid);

```

\ No newline at end of file

Index: pgGrid0.1/src/include/pggrid/pggrid.h

=====

```

--- pgGrid0.1/src/include/pggrid/pggrid.h      (revisão 0)
+++ pgGrid0.1/src/include/pggrid/pggrid.h      (revisão 224)
@@ -0,0 +1,2 @@

```

```

+
+#define SiteList List

```

Index: pgGrid0.1/src/include/pggrid/allocation.h

=====

```

--- pgGrid0.1/src/include/pggrid/allocation.h  (revisão 0)
+++ pgGrid0.1/src/include/pggrid/allocation.h  (revisão 224)
@@ -0,0 +1,32 @@

```

```

+
```

```

#include "postgres.h"
#include "access/heapam.h"
#include "pggrid.h"
#include "nodes/execnodes.h"
#include "catalog/pg_grid_site.h"
+
#define PGGRID 5
+
+/*
+   Search catalogs and applies operation to the fragments/sites that rules
matches
+*/
+bool
+ReplicateOperation(ExprContext *econtext, HeapTuple tuple, Relation relation,
int operation);
+
+/*
+   Last command needs to be replicated in current host (receiver)
+*/
+bool
+ReplicateLastCommandLocally();
+
+/*
+   Check if relation is fragmented between sites
+*/
+bool
+IsRelationFragmented(Oid relid);
+
+/*
+   Verifies if the specied site is current backend
+*/
+extern
+bool IsLocalSite(HeapTuple sitetup, Form_pg_grid_site site);
\ No newline at end of file
Index: pgGrid0.1/src/include/replicate.h

```



```

=====
--- pgGrid0.1/src/include/replicate.h      (revisão 53)
+++ pgGrid0.1/src/include/replicate.h      (cópia de trabalho)
@@ -16,6 +16,7 @@
#include "tcop/dest.h"
#include "storage/proc.h"
#include "lib/stringinfo.h"
+#include "utils/rel.h"
#include "replicate_com.h"

#define STAND_ALONE_TAG                    "When_Stand_Alone"
@@ -139,6 +140,12 @@
extern int PGR_Not_Replicate_Rec_Num;
extern bool autocommit;
extern bool PGR_Is_Replicated_Query;
+
+//#ifdef PGGRID
+extern Relation PGR_target_temp_rel;
+extern char** PGR_target_temp_rel_values;
+extern unsigned int PGR_target_temp_rel_att_counter;
+//#endif
extern PGR_Check_Lock_Type PGR_Check_Lock;
extern int PGR_Sock_To_Replication_Server;
extern bool PGR_Need_Notice;
@@ -170,12 +177,25 @@
extern int PGR_get_replicate_server_socket ( ReplicateServerInfo * sp , int
socket_type );
extern ReplicateServerInfo * PGR_get_replicate_server_info(void);
extern ReplicateServerInfo * PGR_check_replicate_server_info(void);
-extern char * PGR_Send_Replicate_Command(char * query_string, int query_len,
char cmdSts ,char cmdType);
+extern char *
+//#ifdef PGGRID
+PGR_Send_Replicate_Command(char * query_string, int query_len, char cmdSts
,char cmdType, char *sitehostname, uint16_t siteport);

```

```

+/*#else
+PGR_Send_Replicate_Command(char * query_string, int query_len, char cmdSts
, char cmdType);
+#endif*/
extern bool PGR_Is_Replicated_Command(char * query);
extern int Xlog_Check_Replicate(int operation);
extern int PGR_Replicate_Function_Call(void);
extern void PGR_delete_shm(void);
-extern int PGR_replication(char * query_string, CommandDest dest, Node
*parsetree, const char * commandTag);
+extern int
+//#ifdef PGGRID
+PGR_replication(char * query_string, CommandDest dest, Node *parsetree, const
char * commandTag , char *sitehostname, uint16_t siteport);
+/*#else
+PGR_replication(char * query_string, CommandDest dest, Node *parsetree, const
char * commandTag);
+endif*/
+
+extern int PGR_getcountresult();
+
extern bool PGR_Is_System_Command(char * query);
extern int PGR_Call_System_Command(char * command);
extern int PGR_GetTimeOfDay(struct timeval *tp, struct timezone *tpz);
Index: pgGrid0.1/src/include/nodes/parsenodes.h
=====
--- pgGrid0.1/src/include/nodes/parsenodes.h      (revisão 53)
+++ pgGrid0.1/src/include/nodes/parsenodes.h      (cópia de trabalho)
@@ -1277,6 +1277,12 @@
     int                *recoveryport;    /* server recovery port */
 } CreateServerStmt;

+typedef struct DropServerStmt
+{
+    NodeTag            type;

```

```

+   char          *servername;          /* server name */
+} DropServerStmt;
+
/*
 * Pg Grid fragment manipulation statements
 */
@@ -1288,6 +1294,12 @@
        SelectStmt      *select_clause;      /* select */
    } CreateFragmentStmt;

+typedef struct DropFragmentStmt
+{
+   NodeTag          type;
+   char          *fragmentname;          /* fragment name */
+} DropFragmentStmt;
+
/*
 * Pg Grid fragment placing statements
 */
Index: pgGrid0.1/src/include/nodes/nodes.h
=====
--- pgGrid0.1/src/include/nodes/nodes.h      (revisão 53)
+++ pgGrid0.1/src/include/nodes/nodes.h      (cópia de trabalho)
@@ -67,6 +67,7 @@
        T_Hash,
        T_SetOp,
        T_Limit,
+   T_ClusterScan,

/*
 * TAGS FOR PLAN STATE NODES (execnodes.h)
@@ -80,6 +81,7 @@
        T_BitmapOrState,
        T_ScanState,
        T_SeqScanState,

```

```

+   T_ClusterScanState,
      T_IndexScanState,
      T_BitmapIndexScanState,
      T_BitmapHeapScanState,
@@ -350,6 +352,8 @@
      T_CreateServerStmt,
      T_CreateFragmentStmt,
      T_PlaceStmt,
+   T_DropFragmentStmt,
+   T_DropServerStmt,

/*
   * TAGS FOR RANDOM OTHER STUFF
Index: pgGrid0.1/src/include/nodes/execnodes.h
=====
--- pgGrid0.1/src/include/nodes/execnodes.h      (revisão 53)
+++ pgGrid0.1/src/include/nodes/execnodes.h      (cópia de trabalho)
@@ -989,6 +989,7 @@
   * no additional fields.
   */
typedef ScanState SeqScanState;
+typedef ScanState ClusterScanState;

/*
   * These structs store information about index quals that don't have simple
Index: pgGrid0.1/src/include/nodes/plannodes.h
=====
--- pgGrid0.1/src/include/nodes/plannodes.h      (revisão 53)
+++ pgGrid0.1/src/include/nodes/plannodes.h      (cópia de trabalho)
@@ -223,6 +223,7 @@
   * -----
   */
typedef Scan SeqScan;
+typedef Scan ClusterScan;

```

```

/* -----
 *      index scan node
Index: pgGrid0.1/src/include/parser/parser.h
=====
--- pgGrid0.1/src/include/parser/parser.h      (revisão 53)
+++ pgGrid0.1/src/include/parser/parser.h      (cópia de trabalho)
@@ -16,4 +16,7 @@

extern List *raw_parser(const char *str);

+List      *parsetree;          /* result of parsing is left here */
+char *parseCommand;
+
+ #endif /* PARSER_H */
Index: pgGrid0.1/src/include/replicate_com.h
=====
--- pgGrid0.1/src/include/replicate_com.h      (revisão 53)
+++ pgGrid0.1/src/include/replicate_com.h      (cópia de trabalho)
@@ -236,6 +236,10 @@
 #define PGR_QUERY_CONFIRM_ANSWER_FUNC_NO      (9)
 #define PGR_GET_OID_FUNC_NO                   (10)
 #define PGR_SET_OID_FUNC_NO                   (11)
+// #ifdef PGGRID
+#define PGR_RETURN_TUPLES_NO                   (12)
+#define PGR_RETURN_TUPLES_MSG "12"
+// #endif

 #define PGR_CMD_ARG_NUM                       (10)
 #define PGR_LOCK_CONFLICT_NOTICE_CMD         "PGR_LOCK_CONFLICT_NOTICE_CMD"
@@ -272,6 +276,14 @@
 #define PING_DB "template1"
 #define PING_QUERY "SELECT 1"

+//pggrid operation types
+/*Normal pgcluster operation*/

```

```

#define PGGRID_OP_NORMAL 0
+/*Distributed query*/
#define PGGRID_OP_DQUERY 1
+/*Distributed count query*/
#define PGGRID_OP_DCOUNT 2
+
#ifndef HAVE_UNION_SEMUN
union semun {
    int val;
@@ -318,6 +330,14 @@
    char md5Salt[4];
    char cryptSalt[2];
    char dummySalt[2];
+
+ //fragment parameters
+ //ifdef PGGRID
+ char sitehostname[HOSTNAME_MAX_LENGTH];
+ uint16_t siteport;
+ uint16_t siterecport;
+ char pggrid_op_type; //see PGGRID_OP_*
+ //endif
} ReplicateHeader;

typedef struct RecoveryPacketType
Index: pgGrid0.1/src/include/executor/nodeClusterScan.h
=====
--- pgGrid0.1/src/include/executor/nodeClusterScan.h    (revisão 0)
+++ pgGrid0.1/src/include/executor/nodeClusterScan.h    (revisão 224)
@@ -0,0 +1,6 @@
+#include "nodes/execnodes.h"
+
+extern int ExecCountSlotsClusterScan(ClusterScan *node);
+extern ClusterScanState *ExecInitClusterScan(ClusterScan *node, EState *estate,
int eflags);
+extern TupleTableSlot *ExecClusterScan(ClusterScanState *node);

```

```

+extern void ExecEndClusterScan(ClusterScanState *node);
\ No newline at end of file
Index: pgGrid0.1/src/include/utils/syscache.h
=====
--- pgGrid0.1/src/include/utils/syscache.h      (revisão 53)
+++ pgGrid0.1/src/include/utils/syscache.h      (cópia de trabalho)
@@ -1,110 +1,113 @@
-/*-----
- *
- * syscache.h
- *      System catalog cache definitions.
- *
- * See also lsyscache.h, which provides convenience routines for
- * common cache-lookup operations.
- *
- * Portions Copyright (c) 1996-2008, PostgreSQL Global Development Group
- * Portions Copyright (c) 1994, Regents of the University of California
- *
- * $PostgreSQL: postgresql/src/include/utils/syscache.h,v 1.71 2008/01/01 19:45:59
momjian Exp $
- *-----
- */
-#ifndef SYSCACHE_H
-#define SYSCACHE_H
-
-#include "utils/catcache.h"
-
-/*
- *      Declarations for util/syscache.c.
- *
- *      SysCache identifiers.
- *
- *      The order of these must match the order
- *      they are entered into the structure cacheinfo[] in syscache.c.

```

```
- *      Keep them in alphabetical order.
- */
-
-#define AGGFNOID          0
-#define AMNAME           1
-#define AMOID            2
-#define AMOPOPID        3
-#define AMOPSTRATEGY     4
-#define AMPROCNUM       5
-#define ATTNAME          6
-#define ATTNUM           7
-#define AUTHMEMMEMROLE   8
-#define AUTHMEMROLEMEM   9
-#define AUTHNAME        10
-#define AUTHOID          11
-#define CASTSOURCETARGET 12
-#define CLAAMNAMENSP    13
-#define CLAOID           14
-#define CONDEFAULT      15
-#define CONNAMENSP      16
-#define CONSTROID       17
-#define CONVOID          18
-#define DATABASEOID     19
-#define ENUMOID         20
-#define ENUMTYPOIDNAME  21
-#define INDEXRELID      22
-#define LANGNAME        23
-#define LANGOID         24
-#define NAMESPACE_NAME  25
-#define NAMESPACEOID    26
-#define OPERNAMENSP     27
-#define OPEROID         28
-#define OPFAMILYAMNENSP 29
-#define OPFAMILYOID     30
-#define PROCNAMEARGSNENSP 31
```



```

-#define PROCOID                32
-#define RELNAMENSP             33
-#define RELOID                 34
-#define RULERELNAME            35
-#define STATRELATT             36
-#define TSCONFIGMAP            37
-#define TSCONFIGNAMENSP        38
-#define TSCONFIGOID            39
-#define TSDICTNAMENSP          40
-#define TSDICTOID              41
-#define TSPARSERNAMENSP        42
-#define TSPARSEROID            43
-#define TSTEMPLATENAMENSP      44
-#define TSTEMPLATEOID          45
-#define TYPENAMENSP            46
-#define TYPEOID                47
-#define SITENAME                48
-#define FRAGMENTNAME            49
-
-extern void InitCatalogCache(void);
-extern void InitCatalogCachePhase2(void);
-
-extern HeapTuple SearchSysCache(int cacheId,
-                                Datum key1, Datum key2, Datum key3, Datum key4);
-extern void ReleaseSysCache(HeapTuple tuple);
-
-/* convenience routines */
-extern HeapTuple SearchSysCacheCopy(int cacheId,
-                                    Datum key1, Datum key2, Datum key3, Datum key4);
-extern bool SearchSysCacheExists(int cacheId,
-                                  Datum key1, Datum key2, Datum key3, Datum key4);
-extern Oid GetSysCacheOid(int cacheId,
-                           Datum key1, Datum key2, Datum key3, Datum key4);
-
-extern HeapTuple SearchSysCacheAttName(Oid relid, const char *attname);

```

```

-extern HeapTuple SearchSysCacheCopyAttName(Oid relid, const char *attname);
-extern bool SearchSysCacheExistsAttName(Oid relid, const char *attname);
-
-extern Datum SysCacheGetAttr(int cacheId, HeapTuple tup,
-                               AttrNumber attributeNumber, bool *isNull);
-
-/* list-search interface. Users of this must import catcache.h too */
-extern struct catclist *SearchSysCacheList(int cacheId, int nkeys,
-                                             Datum key1, Datum key2, Datum key3, Datum key4);
-
-#define ReleaseSysCacheList(x)    ReleaseCatCacheList(x)
-
-#endif    /* SYSCACHE_H */
+/*-----
+ *
+ * syscache.h
+ *      System catalog cache definitions.
+ *
+ * See also lsyscache.h, which provides convenience routines for
+ * common cache-lookup operations.
+ *
+ * Portions Copyright (c) 1996-2008, PostgreSQL Global Development Group
+ * Portions Copyright (c) 1994, Regents of the University of California
+ *
+ * $PostgreSQL: postgresql/src/include/utils/syscache.h,v 1.71 2008/01/01 19:45:59
+ * momjian Exp $
+ *-----
+ */
+#ifndef SYSCACHE_H
+#define SYSCACHE_H
+
+#include "utils/catcache.h"
+
+/*

```

```

+ *      Declarations for util/syscache.c.
+ *
+ *      SysCache identifiers.
+ *
+ *      The order of these must match the order
+ *      they are entered into the structure cacheinfo[] in syscache.c.
+ *      Keep them in alphabetical order.
+ */
+
+#define AGGFNOID          0
+#define AMNAME           1
+#define AMOID            2
+#define AMOPOPID         3
+#define AMOPSTRATEGY     4
+#define AMPROCNUM        5
+#define ATTNAME          6
+#define ATTNUM           7
+#define AUTHMEMMEMROLE   8
+#define AUTHMEMROLEMEM   9
+#define AUTHNAME         10
+#define AUTHOID          11
+#define CASTSOURCETARGET 12
+#define CLAAMNAMENSP     13
+#define CLAOID           14
+#define CONDEFAULT       15
+#define CONNAMENSP       16
+#define CONSTROID        17
+#define CONVOID           18
+#define DATABASEOID      19
+#define ENUMOID          20
+#define ENUMTYPOIDNAME   21
+#define INDEXRELID       22
+#define LANGNAME         23
+#define LANGOID          24
+#define NAMESPACE_NAME   25

```

```

+#define NAMESPACEOID          26
+#define OPERNAMENSP           27
+#define OPEROID               28
+#define OPFAMILYAMNAMENSP     29
+#define OPFAMILYOID           30
+#define PROCNAMEARGSNP        31
+#define PROCID                32
+#define RELNAMENSP            33
+#define RELOID                34
+#define RULERELNAME           35
+#define STATRELATT            36
+#define TSCONFIGMAP           37
+#define TSCONFIGNAMENSP       38
+#define TSCONFIGOID           39
+#define TSDICTNAMENSP         40
+#define TSDICTOID             41
+#define TSPARSERNAMENSP       42
+#define TSPARSEROID           43
+#define TSTEMPLATENAMENSP     44
+#define TSTEMPLATEOID         45
+#define TYPENAMENSP           46
+#define TYPEOID               47
+#define SITENAME              48
+#define FRAGMENTNAME           49
+#define FRAGMENTRELID         50
+#define ALLOCFRAGMENTID       51
+#define SITEID                52
+
+extern void InitCatalogCache(void);
+extern void InitCatalogCachePhase2(void);
+
+extern HeapTuple SearchSysCache(int cacheId,
+
+                               Datum key1, Datum key2, Datum key3, Datum key4);
+extern void ReleaseSysCache(HeapTuple tuple);
+

```

```

+/* convenience routines */
+extern HeapTuple SearchSysCacheCopy(int cacheId,
+          Datum key1, Datum key2, Datum key3, Datum key4);
+extern bool SearchSysCacheExists(int cacheId,
+          Datum key1, Datum key2, Datum key3, Datum key4);
+extern Oid GetSysCacheOid(int cacheId,
+          Datum key1, Datum key2, Datum key3, Datum key4);
+
+extern HeapTuple SearchSysCacheAttName(Oid relid, const char *attname);
+extern HeapTuple SearchSysCacheCopyAttName(Oid relid, const char *attname);
+extern bool SearchSysCacheExistsAttName(Oid relid, const char *attname);
+
+extern Datum SysCacheGetAttr(int cacheId, HeapTuple tup,
+          AttrNumber attributeNumber, bool *isNull);
+
+/* list-search interface. Users of this must import catcache.h too */
+extern struct catclist *SearchSysCacheList(int cacheId, int nkeys,
+          Datum key1, Datum key2, Datum key3, Datum key4);
+
+#define ReleaseSysCacheList(x)    ReleaseCatCacheList(x)
+
+#endif    /* SYSCACHE_H */
Index: pgGrid0.1/src/include/commands/fragment.h
=====
--- pgGrid0.1/src/include/commands/fragment.h      (revisão 53)
+++ pgGrid0.1/src/include/commands/fragment.h      (cópia de trabalho)
@@ -13,6 +13,8 @@

extern void CreateFragment(CreateFragmentStmt *stmt, const char *queryString);

+extern void DropFragment(DropFragmentStmt *stmt, const char *queryString);
+
extern void InsertFragmentAttribute(Oid fragmentid, Oid relid, const char
*attname);

```

```

#endif    /* FRAGMENT_H */
Index: pgGrid0.1/src/include/commands/server.h
=====
--- pgGrid0.1/src/include/commands/server.h    (revisão 53)
+++ pgGrid0.1/src/include/commands/server.h    (cópia de trabalho)
@@ -13,4 +13,6 @@

extern void CreateServer(CreateServerStmt *stmt);

+extern void DropServer(DropServerStmt *stmt);
+
#endif    /* SERVER_H */
Index: pgGrid0.1/src/include/catalog/indexing.h
=====
--- pgGrid0.1/src/include/catalog/indexing.h    (revisão 53)
+++ pgGrid0.1/src/include/catalog/indexing.h    (cópia de trabalho)
@@ -258,6 +258,15 @@
DECLARE_UNIQUE_INDEX(pg_grid_fragment_name_index, 3778, on pg_grid_fragment
using btree(fragmentname name_ops));
#define FragmentNameIndexId    3778

+DECLARE_INDEX(pg_grid_fragment_relid_index, 3779, on pg_grid_fragment using
btree(relid oid_ops));
+#define FragmentRelIndexId    3779
+
+DECLARE_INDEX(pg_grid_allocation_fragid_index, 3780, on pg_grid_allocation
using btree(fragmentid oid_ops));
+#define AllocFragmentIndexId    3780
+
+DECLARE_UNIQUE_INDEX(pg_grid_site_id_index, 3781, on pg_grid_site using
btree(siteid oid_ops));
+#define SiteIdIndexId    3781
+
/* last step of initialization script: build the indexes declared above */
BUILD_INDICES

```

Index: pgGrid0.1/src/backend/pggrid/allocation.c

```

=====
--- pgGrid0.1/src/backend/pggrid/allocation.c      (revisão 0)
+++ pgGrid0.1/src/backend/pggrid/allocation.c      (revisão 224)
@@ -0,0 +1,200 @@
+#include "pggrid/allocation.h"
+
+#include "utils/syscache.h"
+#include "utils/catcache.h"
+#include "utils/rel.h"
+#include "utils/palloc.h"
+#include "utils/memutils.h"
+#include "utils/builtins.h"
+
+#include "catalog/pg_grid_fragment.h"
+#include "catalog/pg_grid_allocation.h"
+
+#include "executor/executor.h"
+
+#include "nodes/nodes.h"
+#include "nodes/parsenodes.h"
+
+#include "parser/parser.h"
+#include "parser/analyze.h"
+
+#include "optimizer/clauses.h"
+
+#include "tcop/dest.h"
+#include "tcop/utility.h"
+
+#include "replicate.h"
+
+#include "postgres.h"
+

```

```

#include "postmaster/postmaster.h"
+
#include <string.h>
+
bool localReplicateLastCommand;
+
static
+List *GetFragmentQuals(HeapTuple fragmenttup, Form_pg_grid_fragment fragment);
+
+/*
+   Send each operation to the envolved site
+*/
static
+bool SendOperation(HeapTuple sitetup, Form_pg_grid_site site, int operation,
HeapTuple tuple);
+
static
+List *GetFragmentQuals(HeapTuple fragmenttup, Form_pg_grid_fragment fragment){
+   Datum wheredata=SysCacheGetAttr(FRAGMENTNAME, fragmenttup,
Anum_pg_grid_fragment_where, false);
+   char *where=DatumGetCString(DirectFunctionCall1(textout, wheredata));
+   if (strcmp(where, "")){
+       return make_and_implicit(stringToNode(where));
+   }
+   return NIL;
+}
+
+bool
+ReplicateOperation(ExprContext *econtext, HeapTuple tuple, Relation relation,
int operation){
+   int i, j;
+   EState *estate;
+   bool rv=true;
+   int fragment_count;
+   bool replicated=false, fragments=false;

```



```

+   CatCList *fragmentList;
+   if (PGR_Is_Replicated_Query){
+       //this is a replicated query. Just execute locally
+       localReplicateLastCommand=true;
+       return true;
+   }
+   fragmentList = SearchSysCacheList(FRAGMENTRELID, 1, relation->rd_id, 0, 0,
0);
+   localReplicateLastCommand=false;
+   fragment_count=fragmentList->n_members;
+
+   for (i = 0; i < fragmentList->n_members; i++)
+   {
+       HeapTuple   fragmenttup = &fragmentList->members[i]->tuple;
+       Form_pg_grid_fragment fragment = (Form_pg_grid_fragment)
GETSTRUCT(fragmenttup);
+
+       //for each fragment, test if the fragment rule matches.
+       //and if true, send to the sites assigned for
+       List *quals= GetFragmentQuals(fragmenttup, fragment);
+
+
+       //horizontal fragmentation is assembled here
+       bool match=true;
+       if (quals){
+           estate = CreateExecutorState();
+           econtext = GetPerTupleExprContext(estate);
+           TupleTableSlot *slot =
MakeSingleTupleTableSlot(RelationGetDescr(relation));
+           ExecStoreTuple(tuple, slot, InvalidBuffer, false);
+           econtext->ecxt_scantuple = slot;
+
+           List *predicate = (List *)
+               ExecPrepareExpr((Expr *) quals, estate);
+

```

```

+         match=ExecQual(predicate, econtext, false);
+
+         ExecDropSingleTupleTableSlot(slot);
+         ResetExprContext(econtext);
+         FreeExecutorState(estate);
+         //free(quals);
+     }
+     elog(NOTICE, "Fragment name");
+     elog(NOTICE, (fragment->fragmentname).data);
+
+     if (match)
+         elog(NOTICE, "Fragment criteria matches");
+     else
+         elog(NOTICE, "Fragment criteria does not match");
+     if (match){
+         //distribute fragment
+         CatCList *siteList = SearchSysCacheList(ALLOCFRAGMENTID, 1,
fragment->fragmentid, 0, 0, 0);
+         if (siteList){
+             for (j = 0; j < siteList->n_members; j++)
+             {
+                 HeapTuple    alloctup = &(amp;siteList->members[j]->tuple);
+                 Form_pg_grid_allocation allocation = (Form_pg_grid_allocation)
GETSTRUCT(alloctup);
+
+                 HeapTuple sitetup = SearchSysCache(SITEID,
allocation->siteid, 0, 0, 0);
+                 Form_pg_grid_site site = (Form_pg_grid_site)
GETSTRUCT(sitetup);
+                 if (IsLocalSite(sitetup, site)){
+                     //elog(NOTICE, "Just a local site");
+                     localReplicateLastCommand=true;
+                 }
+                 else{
+                     //elog(NOTICE, "Sending operation to site");

```

```

+             if (!SendOperation(sitetup, site, operation, tuple)){
+                 rv=false;
+                 ReleaseSysCache(sitetup);
+                 break;
+             }
+         }
+         ReleaseSysCache(sitetup);
+         replicated=true;
+     }
+     ReleaseSysCacheList(siteList);
+ }
+ }
+ }
+ ReleaseSysCacheList(fragmentList);
+
+ if (fragment_count == 0){
+     //table has no fragments. Store locally
+     localReplicateLastCommand=true;
+ }else if (!replicated){
+     elog(ERROR, "Record does not match to any fragment. Cancelled");
+ }
+
+ return rv;
+}
+
+static
+bool SendOperation(HeapTuple sitetup, Form_pg_grid_site site, int operation,
HeapTuple tuple){
+    int status=0;
+    Datum sitehostdata;
+    char *commandTag;
+    ListCell *cell;
+    Node *parsenode=NULL;
+    foreach(cell, parsetree)
+    {

```

```

+     parsenode=(Node *)lfirst(cell);
+     break;
+ }
+     commandTag = CreateCommandTag(parsenode);
+
+     sitehostdata=SysCacheGetAttr(SITENAME, sitetup, Anum_pg_grid_site_host,
false);
+     char *sitehostname=DatumGetCString(DirectFunctionCall1(textout,
sitehostdata));
+     int siteport=DatumGetInt32(SysCacheGetAttr(SITENAME, sitetup,
Anum_pg_grid_site_port, false));
+
+     status = PGR_replication(parseCommand, DestDebug, parsenode, commandTag,
sitehostname, siteport);
+     if (status == STATUS_REPLICATED || status == STATUS_CONTINUE)
+     {
+         return true;
+     }
+     else
+         return false;
+}
+
+bool
+IsLocalSite(HeapTuple sitetup, Form_pg_grid_site site){
+
+     Datum sitehostdata=SysCacheGetAttr(SITENAME, sitetup,
Anum_pg_grid_site_host, false);
+     char *sitehostname=DatumGetCString(DirectFunctionCall1(textout,
sitehostdata));
+     int siteport=DatumGetInt32(SysCacheGetAttr(SITENAME, sitetup,
Anum_pg_grid_site_port, false));
+
+     return PGRis_same_host(PGRSelfHostName, ntohs(PostPortNumber), sitehostname,
ntohs(siteport));
+}

```

```

+
+bool
+ReplicateLastCommandLocally(){
+    return localReplicateLastCommand;
+}
+
+
+bool
+IsRelationFragmented(Oid relid){
+    CatCList *fragmentList = SearchSysCacheList(FRAGMENTRELID, 1, relid, 0, 0,
0);
+    int fragment_count=fragmentList->n_members;
+    ReleaseSysCacheList(fragmentList);
+    return (fragment_count>0);
+}
+
Index: pgGrid0.1/src/backend/pggrid/dquery.c
=====
--- pgGrid0.1/src/backend/pggrid/dquery.c      (revisão 0)
+++ pgGrid0.1/src/backend/pggrid/dquery.c      (revisão 224)
@@ -0,0 +1,235 @@
+#include "pggrid/dquery.h"
+#include "pggrid/allocation.h"
+#include "utils/syscache.h"
+#include "utils/catcache.h"
+#include "utils/rel.h"
+#include "utils/palloc.h"
+#include "utils/memutils.h"
+#include "utils/builtins.h"
+
+#include "catalog/pg_grid_fragment.h"
+#include "catalog/pg_grid_site.h"
+#include "catalog/pg_grid_allocation.h"
+
+#include "executor/executor.h"

```

```
+
#include "nodes/nodes.h"
#include "nodes/parsenodes.h"
+
#include "parser/parser.h"
#include "parser/analyze.h"
+
#include "optimizer/clauses.h"
+
#include "tcop/dest.h"
#include "tcop/utility.h"
+
#include "replicate.h"
+
#include "postgres.h"
+
#include "postmaster/postmaster.h"
+
#include <string.h>
+
+static
+int SendCountOperation(HeapTuple sitetup, Form_pg_grid_site site, int
operation, Oid relationId);
+static
+bool RetrieveRelationTuplesFromSite(HeapTuple sitetup, Form_pg_grid_site site,
Index relid, Relation targetTempRelation);
+
+static
+Relation CreateTemporaryRelation(Index relid);
+
+
+int ExecCountForeignTuples(Index relid){
+   int i, j;
+   int rv=0, count;
+   int fragment_count;
```



```

+             rv=-1;
+             ReleaseSysCache(sitetup);
+             break;
+         }
+     }
+     ReleaseSysCache(sitetup);
+ }
+ ReleaseSysCacheList(siteList);
+ }
+ }
+ ReleaseSysCacheList(fragmentList);
+
+ return rv;
+
+}
+
+int
+SendCountOperation(HeapTuple sitetup, Form_pg_grid_site site, int operation,
Oid relationId){
+ int status=0;
+ Datum sitehostdata;
+ char parseCommand[256];
+
+ Relation onerel;
+
+ onerel = try_relation_open(relationId, NoLock);
+ if (!onerel)
+     return -1;
+
+ snprintf(parseCommand, sizeof(parseCommand),
+          "count %s", RelationGetRelationName(onerel));
+
+ relation_close(onerel, NoLock);
+

```



```

+
+   sitehostdata=SysCacheGetAttr(SITENAME, sitetup, Anum_pg_grid_site_host,
+ false);
+   char *sitehostname=DatumGetCString(DirectFunctionCall1(textout,
+ sitehostdata));
+   int siteport=DatumGetInt32(SysCacheGetAttr(SITENAME, sitetup,
+ Anum_pg_grid_site_port, false));
+
+   //elog(NOTICE, "PGGRID: Sending count operation");
+   status = PGR_replication(parseCommand, DestDebug, NULL, "COUNT", sitehostname,
+ siteport);
+   if (status == STATUS_REPLICATED || status == STATUS_CONTINUE)
+   {
+       return PGR_getcountresult();
+   }
+   return -1;
+}
+
+
+Oid
+ExecClusterQuery(Index relid){
+
+   CatCList *fragmentList;
+   int i, j;
+   Relation rv;
+   Oid ret;
+   //insert data in relation
+
+   rv=CreateTemporaryRelation(relid);
+   fragmentList = SearchSysCacheList(FRAGMENTRELID, 1, relid, 0, 0, 0);
+
+   //verifies each fragment regarding target relation
+   for (i = 0; i < fragmentList->n_members; i++)
+   {
+
+

```

```

+     //elog(NOTICE, "PGGRID: Processando fragmento");
+     HeapTuple    fragmenttup = &fragmentList->members[i]->tuple;
+     Form_pg_grid_fragment fragment = (Form_pg_grid_fragment)
GETSTRUCT(fragmenttup);
+
+     //verifies each site that stores target relation
+     CatCList *siteList = SearchSysCacheList(ALLOCFRAGMENTID, 1,
fragment->fragmentid, 0, 0, 0);
+     if (siteList){
+         for (j = 0; j < siteList->n_members; j++)
+         {
+             HeapTuple    alloctup = &(siteList->members[j]->tuple);
+             Form_pg_grid_allocation allocation = (Form_pg_grid_allocation)
GETSTRUCT(alloctup);
+
+             HeapTuple sitetup = SearchSysCache(SITEID, allocation->siteid,
0, 0, 0);
+             Form_pg_grid_site site = (Form_pg_grid_site)
GETSTRUCT(sitetup);
+
+             if (!IsLocalSite(sitetup, site)){
+
+                 if (!RetrieveRelationTuplesFromSite(sitetup, site, relid,
rv)){
+                     elog(ERROR, "PGGRID: Cannot get tuples from site %s",
site->sitename);
+                 }
+             }
+             ReleaseSysCache(sitetup);
+         }
+         ReleaseSysCacheList(siteList);
+     }
+ }
+ ReleaseSysCacheList(fragmentList);
+

```

```

+ //close temp rel
+ //elog(NOTICE, "PGGRID: closing temp rel");
+ ret=rv->rd_id;
+ heap_close(rv, NoLock);
+ //elog(NOTICE, "PGGRID: dquery executed");
+ return ret;
+}
+
+
+Relation
+CreateTemporaryRelation(Index relid){
+ //get tablespace Oid
+ Oid tablespaceOid;
+ Relation relation=heap_open(relid, NoLock);
+ if (relation && relation->rd_rel)
+ tablespaceOid=relation->rd_rel->reltablespace;
+ else
+ elog(ERROR, "PGGRID: failed to open relation");
+ heap_close(relation, NoLock);
+
+ Oid tempRelOid=make_new_heap(relid, "pg_temp_dquery", tablespaceOid);
+ return heap_open(tempRelOid, AccessExclusiveLock);
+}
+
+
+static
+bool RetrieveRelationTuplesFromSite(HeapTuple sitetup, Form_pg_grid_site site,
Index relid, Relation targetTempRelation){
+ int status=0;
+ Datum sitehostdata;
+ char parseCommand[256];
+
+ //get target relation name
+ Relation onerel;
+ onerel = try_relation_open(relid, NoLock);
+ if (!onerel)

```

```

+     return -1;
+
+     snprintf(parseCommand, sizeof(parseCommand),
+              "get %s", RelationGetRelationName(onerel));
+
+     relation_close(onerel, NoLock);
+
+
+     sitehostdata=SysCacheGetAttr(SITENAME, sitetup, Anum_pg_grid_site_host,
+ false);
+     char *sitehostname=DatumGetCString(DirectFunctionCall1(textout,
+ sitehostdata));
+     int siteport=DatumGetInt32(SysCacheGetAttr(SITENAME, sitetup,
+ Anum_pg_grid_site_port, false));
+
+     //elog(NOTICE, "PGGRID: Sending get operation");
+
+     //initialize temporary table variables
+     PGR_target_temp_rel=targetTempRelation;
+     PGR_target_temp_rel_values=(char **)
+ malloc(RelationGetNumberOfAttributes(PGR_target_temp_rel)*sizeof(char *));
+     PGR_target_temp_rel_att_counter=0;
+
+     status = PGR_replication(parseCommand, DestDebug, NULL, "LOAD DATA",
+ sitehostname, siteport);
+
+     //freeze temporary relation variables
+     PGR_target_temp_rel=NULL;
+     free(PGR_target_temp_rel_values);
+
+     if (status == STATUS_REPLICATED || status == STATUS_CONTINUE)
+     {
+         return true;
+     }
+     return false;

```

+}

+

APÊNDICE B - Artigo

PgGrid: uma implementação de fragmentação de dados para o pgCluster

Gustavo A. Tonini, Frank Siqueira

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Florianópolis – SC – Brasil

`gustavotonini@gmail.com, frank@inf.ufsc.br`

Abstract. *As an organization grows, it needs to store great amounts of data and to organize them in such a way that favors its recovery increases. The proposal of this project is to offer an extension to the PostgreSQL DBMS that allows the fragmentation of data, so that they are distributed in the most convenient form among sites that compose the distributed database system, and to manage the replication of such data. For this it was necessary to modify the "pgcluster" tool, to manage data location and to optimize queries. Moreover, an extension to the DDL was proposed, for the definition of the data distribution parameters and the sites that compose the distributed database system.*

Resumo. *À medida que as organizações crescem, também cresce a necessidade de armazenar grandes massas de dados e organizá-los de uma forma que favoreça sua recuperação. A proposta deste trabalho é oferecer uma extensão ao SGBD PostgreSQL que permita a fragmentação dos dados para que os mesmos sejam distribuídos da forma mais conveniente nos servidores de banco de dados que compõem o conjunto, além de gerenciar sua replicação. Para isso foi necessário modificar a ferramenta "pgcluster" para gerenciar a localização dos dados no sistema distribuído e otimizar as consultas. Além disso, foi implementada uma extensão à linguagem DDL para a definição dos parâmetros da distribuição dos dados e dos "sítios" que formam o sistema de banco de dados distribuído.*

1. Introdução

Nos primórdios da computação, todo processamento era realizado de forma centralizada. Com o passar dos anos foi-se notando o potencial que as características dos sistemas distribuídos proporcionam. Sistemas de banco de dados distribuídos são sistemas distribuídos que armazenam, manipulam e organizam dados.

Muitos estudos já foram realizados nesta área, principalmente no que diz respeito ao gerenciamento de transações e integridade dos dados em sistemas deste tipo. No entanto, existem poucas implementações disponíveis para uso.

De acordo com Ozsu (1999), as principais promessas e características dos sistemas de banco de dados distribuídos são:

- Administração transparente dos dados fragmentados nas máquinas que compõem o sistema;
- Confiabilidade nas transações distribuídas;
- Melhoria no desempenho das consultas, através da paralelização das mesmas e outras estratégias de otimização; e
- Fácil expansão e boa escalabilidade do sistema.

O presente artigo apresenta a implementação de uma extensão para o SGBD PostgreSQL que tem como intuito adicionar suporte à fragmentação e replicação de dados. Adicionalmente, a linguagem DDL do SGBD foi estendida de forma a suportar em sua sintaxe comandos que permitam definir a forma como os dados serão fragmentados e/ou replicados dentre os servidores integrantes do esquema de dados distribuído.

O artigo está organizado da seguinte maneira: a seção 2 descreve o PostgreSQL e o suporte para replicação existente; a seção 3 descreve as alterações efetuadas no SGBD de modo a suportar replicação e fragmentação; a seção 4 compara a implementação realizada com outros SGBDs distribuídos; a seção 5 demonstra, através de um caso de uso, a utilização do SGBD distribuído; por fim, a seção 6 apresenta as conclusões dos autores e perspectivas de evolução do trabalho.

2. PostgreSQL e pgCluster

PostgreSQL é um sistema gerenciador de banco de dados de código-fonte aberto poderoso, multiplataforma, com mais de 15 anos de desenvolvimento. Atende os padrões SQL99 e seu gerenciador de transações implementa as propriedades ACID [PostgreSQL 2009].

O *pgCluster* é um sistema de replicação síncrono baseado no PostgreSQL. O sistema possui duas funções principais:

- **Balanceamento de carga:** Um módulo recebe os comandos de consulta e transações e os distribui da forma mais conveniente, levando em consideração a carga (capacidade de processamento comprometida) dos servidores.
- **Alta disponibilidade:** É garantida através da replicação total dos dados; assim, se um servidor falha, os demais podem responder em seu lugar, desde que pelo menos um servidor de replicação continue no ar.

Conforme se pode facilmente perceber, o maior problema do pgCluster é a falta das funcionalidades de fragmentação, que exigem que o usuário replique totalmente as bases de dados para que o sistema funcione, degradando o desempenho e também desperdiçando recursos do sistema.

Dentre uma gama enorme de SGBD relacionais e objeto-relacionais disponíveis no mercado, o PostgreSQL se sobressai devido à sua aceitação pela comunidade de software livre mundial. Além das qualidades citadas acima, o PostgreSQL possui grande

usabilidade em ambientes distribuídos homogêneos, possuindo grande parte das funções de replicação já implementadas e testadas.

3. Alterações no pgCluster

Fragmentar os dados é uma necessidade básica na maioria dos ambientes distribuídos. Com este propósito, o pgGrid adiciona funções de fragmentação para o pgCluster e fornece comandos, como uma extensão à SQL, para definição do ambiente de cluster.

A configuração da replicação no pgCluster exige a manipulação de arquivos XML. Desta forma, torna-se complexo para o administrador definir as novas configurações do pgGrid, como a definição dos fragmentos e a alocação dos mesmos. Para resolver este problema foram adicionados comandos específicos para a definição destes parâmetros que serão gravados nos catálogos do sistema.

Três grupos de instruções especiais foram adicionados à gramática:

- Adição e exclusão de servidores no sistema (CREATE SERVER);
- Definição de fragmentos (CREATE FRAGMENT);
- Alocação de fragmentos nos sites (PLACE);

A Figura 1 apresenta alguns exemplos de uso dos comandos adicionados.

```
CREATE SERVER sitel HOST inf.ufsc.br PORT 5432 RECOVERY PORT 7000;
CREATE FRAGMENT cliente_pessoa_fisica ON CLIENTE where tipo_pessoa = 'F';
CREATE FRAGMENT cliente_filial1 ON CLIENTE where cod_filial = 1;
PLACE cliente_pessoa_fisica ON sitel;
```

Figura 1. Exemplo de uso dos comandos de definição do cluster

As informações definidas nos comandos supracitados ficam armazenadas nos catálogos do sistema. Quatro catálogos foram adicionados com este propósito:

- pg_grid_site;
- pg_grid_fragment;
- pg_grid_fragment_attribute;
- pg_grid_allocation;

Depois que os parâmetros foram definidos, o pgGrid mantém os servidores sincronizados de forma a garantir as regras de fragmentação explicitadas pelo administrador.

Sites e fragmentos podem ser definidos e alocados a qualquer momento, sem a necessidade de interrupção do funcionamento do sistema. Tal característica provê a escalabilidade necessária para aplicações reais.

3.1. Processamento dos comandos de manipulação de dados

Toda vez que um comando INSERT, UPDATE ou DELETE é submetido ao servidor, o mesmo é analisado e enviado para um módulo denominado *executor*. O executor foi

alterado para consultar os catálogos do pgGrid e verificar a necessidade de replicação da solicitação.

Quando o sistema identifica que um comando destes deve alterar dados de um ou mais sítios, é iniciada uma transação distribuída onde o comando em questão é replicado para todos os sites envolvidos. A execução do comando somente é dada como concluída quando todos os sites confirmarem o sucesso da operação para o site que recebeu a solicitação.

3.2. Processamento das consultas distribuídas

Toda vez que um comando SELECT é submetido ao servidor, o mesmo é analisado, passa por otimizações para melhorar seu desempenho e depois é enviado para o mesmo módulo *executor* dos demais comandos. O executor das consultas foi alterado para consultar os catálogos do pgGrid e verificar a necessidade de solicitação de dados para outros sites do cluster.

Quando o sistema identifica que uma consulta necessita de dados externos (através de reduções [Ozsu 1999]), uma solicitação é enviada para cada site que contém dados usados pela consulta. Cada site processa as operações possíveis no seu conjunto de dados (provendo paralelização) e envia o resultado para o servidor que recebeu a solicitação, este é responsável por “juntar” os dados (através de operações de união e junção) e apresentar o resultado final ao usuário.

4. Outras implementações de grid

Como parte do trabalho, foi realizada a avaliação de várias implementações de banco de dados distribuídos disponíveis no mercado.

Os gerenciadores avaliados foram: MySQL Cluster, Oracle RAC e IBM DB2. A avaliação foi realizada segundo alguns critérios relevantes para a manutenção de um sistema de banco de dados distribuído, de uma forma comparativa com o pgGrid.

A Tabela 1 lista os critérios escolhidos e a avaliação de cada produto quanto à compatibilidade com o mesmo.

Característica	MySQL	Oracle	IBM DB2	pgGrid
Consultas distribuídas	Sim	Sim	Sim	Sim
Tecnologia multi-mestre	Sim	Sim	Sim	Sim
Replicação parcial	Não	Sim	Sim	Sim
Replicação síncrona	Sim	Sim	Sim	Sim
Replicação assíncrona	Não	Sim	Não	Não
Fragmentação	Sim	Sim	Sim	Sim
Integridade referencial entre os sites	Não	Não	Não	Sim
Protocolo de confiabilidade distribuída	2PC	2PC	2PC	2PC

Tabela 1. Comparativo entre as tecnologias de BDD

5. Caso de uso

Para demonstrar as funcionalidades do pgGrid, um esquema de BDD foi montado. O esquema armazena dados sobre produtos produzidos no estado de Santa Catarina, conforme ilustrado no diagrama entidade-relacionamento da Figura 2.

Cinco sites pertencem ao sistema, localizados nas principais cidades do estado: Florianópolis (FLN), Joinville (JVL), Blumenau (BLU), Criciúma (CRI) e Chapecó (XAP). As regras de armazenamento são simples: cada cidade armazena seu cadastro e os produtos produzidos em sua região. A capital (FLN), além dos dados de seus produtos, armazena o cadastro de todas as cidades do estado. Os produtos possuem como atributos: código, nome e cidade de origem. As cidades possuem código, nome e a distância (em quilômetros) até a capital.

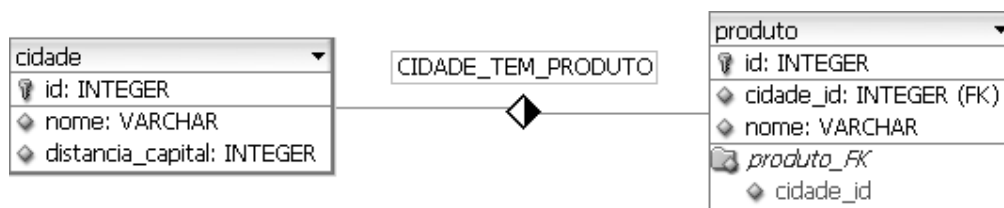


Figura 2. Diagrama entidade-relacionamento correspondente ao caso de uso

Depois que os cinco servidores foram configurados e colocados no ar, o modelo de dados e sua fragmentação foram definidos através dos comandos apresentados na Figura 3.

```

/*relação cidade*/
sc=# CREATE TABLE CIDADE (ID INT, NOME VARCHAR, DISTANCIA_CAPITAL INT);
sc=# ALTER TABLE CIDADE ADD CONSTRAINT PK_CIDADE PRIMARY KEY (ID);
/*relação produto*/
sc=# CREATE TABLE PRODUTO (ID INT, NOME VARCHAR, ID_CIDADE_ORIGEM INT);
sc=# ALTER TABLE PRODUTO ADD CONSTRAINT PK_PRODUTO PRIMARY KEY (ID);
sc=# ALTER TABLE PRODUTO ADD CONSTRAINT FK_PRODUTO_CIDADE FOREIGN
sc=# KEY (ID_CIDADE_ORIGEM) REFERENCES CIDADE (ID);
/*definição e alocação dos fragmentos dos produtos*/
sc=# CREATE FRAGMENT PRODUTO_FLN ON PRODUTO WHERE ID_CIDADE_ORIGEM=1;
sc=# PLACE PRODUTO_FLN ON FLN;
sc=# CREATE FRAGMENT PRODUTO_JVL ON PRODUTO WHERE ID_CIDADE_ORIGEM=2;
sc=# PLACE PRODUTO_JVL ON JVL;
sc=# CREATE FRAGMENT PRODUTO_BLU ON PRODUTO WHERE ID_CIDADE_ORIGEM=3;
sc=# PLACE PRODUTO_BLU ON BLU;
sc=# CREATE FRAGMENT PRODUTO_CRI ON PRODUTO WHERE ID_CIDADE_ORIGEM=4;
sc=# PLACE PRODUTO_CRI ON CRI;
sc=# CREATE FRAGMENT PRODUTO_XAP ON PRODUTO WHERE ID_CIDADE_ORIGEM=5;
sc=# PLACE PRODUTO_XAP ON XAP;
/*definição e alocação dos fragmentos das cidades*/
CREATE FRAGMENT CIDADE_FLN ON CIDADE;
PLACE CIDADE_FLN ON FLN;
CREATE FRAGMENT CIDADE_JVL ON CIDADE WHERE ID=2;
PLACE CIDADE_JVL ON JVL;
CREATE FRAGMENT CIDADE_BLU ON CIDADE WHERE ID=3;
PLACE CIDADE_BLU ON BLU;
CREATE FRAGMENT CIDADE_CRI ON CIDADE WHERE ID=4;
  
```

```

PLACE CIDADE_CRI ON CRI;
CREATE FRAGMENT CIDADE_XAP ON CIDADE WHERE ID=5;
PLACE CIDADE_XAP ON XAP;

```

Figura 3. Comandos utilizados na definição da fragmentação dos dados no caso de uso do pgGrid

A fragmentação foi definida através do identificador numérico de cada cidade. Este código representa a regra de alocação dos dados nos sites.

Foram realizados testes inserindo as cidades e vários produtos no sistema. Depois todos os sites foram reiniciados no modo centralizado (sem sincronia com o cluster). Cada site continha somente os dados da cidade que representava, atestando o funcionamento do pgGrid. O site que representava a capital listou todas as cidades do estado, conforme configurado.

6. Conclusões

Com a crescente necessidade de compartilhamento de informações, está surgindo uma demanda muito grande no armazenamento centralizado das mesmas. No entanto, com volumes muito grandes de dados, a centralização torna o processo de recuperação e carga dos dados ineficiente.

Para resolver tal problema, os sistemas distribuídos entram em jogo fornecendo soluções escaláveis e paralelizáveis. Este trabalho possui o intuito de propor uma solução deste tipo no contexto de banco de dados relacionais e objeto-relacionais.

No decorrer do trabalho, a escolha do PostgreSQL como base para a implementação se tornou muito vantajosa devido ao nível de estabilidade e amadurecimento de suas funcionalidades, assim como pela qualidade e legibilidade do seu código-fonte.

A proposta apresentada no início do trabalho mostrou-se viável no decorrer do desenvolvimento do mesmo. Tal viabilidade foi ilustrada pelo caso de uso. Existem muitas otimizações que devem ser implementadas e o software deve passar por testes mais rigorosos antes de ser colocado em produção em casos reais.

Como próximo passo planeja-se implementar as otimizações de consultas que o ambiente distribuído possibilita, tais como a paralelização de subconsultas. Também se pretende aperfeiçoar os servidores de modo a melhorar o protocolo de comunicação entre os mesmos e remover a necessidade de servidores de replicação.

Repositório do projeto

PgGrid é um software de código-fonte aberto, disponível pela licença GPL e está disponível para download no site <http://pgfoundry.org/projects/pggrid/>.

Referências

Ozsu, M. Tamer (1999) “Principles of distributed database systems”. New Jersey, United States, Prentice Hall, p. 7-10.

PostgreSQL Development Group (2009) “About PostgreSQL”, <http://www.postgresql.org/>, Janeiro.

Bedoya, Hernando et al. "Advanced Functions and Administration on DB2 Universal Database for iSeries", <http://www.redbooks.ibm.com/abstracts/sg244249.html>, Janeiro.