UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Gustavo Alexssandro Tonini

UM ALGORITMO DE ALOCAÇÃO PARA BANCOS DE DADOS BIOLÓGICOS DISTRIBUÍDOS

Florianópolis

2014

Gustavo Alexssandro Tonini

UM ALGORITMO DE ALOCAÇÃO PARA BANCOS DE DADOS BIOLÓGICOS DISTRIBUÍDOS

Dissertação submetida ao Programa de Pós Graduação em Ciência da Computação para a obtenção do Grau de Mestre em Ciência da Computação.
Orientador: Prof. Dr. Frank Augusto Siqueira

Florianópolis

2014

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

Tonini, Gustavo Alexssandro
Um algoritmo de alocação para bancos de dados biológicos distribuídos / Gustavo Alexssandro Tonini ; orientador, Frank Augusto Siqueira - Florianópolis, SC, 2014.
128 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. bancos de dados distribuídos. 3. alocação de dados. 4. biotecnologia. 5. datawarehouses biológicos. I. Siqueira, Frank Augusto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Gustavo Alexssandro Tonini

UM ALGORITMO DE ALOCAÇÃO PARA BANCOS DE DADOS BIOLÓGICOS DISTRIBUÍDOS

Esta Dissertação foi julgada aprovada para a obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós Graduação em Ciência da Computação.

Florianópolis, 18 de setembro 2014.

Prof. Dr. Ronaldo dos Santos Mello
Coordenador do Curso

Prof. Dr. Frank Augusto Siqueira
Orientador

Banca Examinadora:

Prof. Dr. Frank Augusto Siqueira
Presidente

Prof. Dr. Renato Fileto

Prof. Dr. Lau Cheuk Lung

Prof. Dra. Jeroniza Marchaukoski

RESUMO

O presente trabalho propõe um algoritmo de alocação de dados distribuídos baseado na afinidade de dados e perfis de uso com foco em bancos de dados (BD) relacionais biológicos. A proposta visa instruir os administradores de banco de dados (DBAs) sobre como alocar os dados nos nós de um cluster visando obter o melhor desempenho possível nas consultas e demais requisições dos usuários. O esquema é verificado através de testes em laboratório. Os experimentos são realizados sobre o sistema *datawarehouse* (DW) Intermine (SMITH et al., 2012) utilizando o pgGrid, que adiciona funções de replicação e fragmentação no PostgreSQL e o HadoopDB (implementação do modelo Map-Reduce para bancos de dados relacionais). O algoritmo é comparado com outras propostas de alocação geradas por algoritmos desenvolvidos em pesquisas recentes.

Palavras-chave: biotecnologia, bancos de dados distribuídos, datawarehouses biológicos, otimização de consultas, alocação de dados, particionamento de dados

ABSTRACT

This work proposes a data allocation algorithm based on distributed data affinity and query profile with focus on biological relational databases. The proposal aims to help database administrators (DBAs) about how to allocate the data across nodes in a cluster in order to obtain the maximum performance improvements on query time and executing other user requests. The allocation schema is verified in laboratory tests. The Intermine *datawarehouse* (DW) system (SMITH et al., 2012) was chosen as subject of this evaluation. The experiments were executed on distributed database platforms such as pgGrid, which adds replication and fragmentation functions to PostgreSQL and HadoopDB (implementation of Map-Reduce model for relational databases). Finally, the algorithm is compared with other allocation methods developed in recent researches.

Keywords: biotechnology, distributed databases, biological datawarehouses, query optimization, data allocation, data partitioning

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SUMÁRIO

1 INTRODUÇÃO

Bases de dados e repositórios de informações biológicas crescem rapidamente (NCBI, 2013). Aplicações específicas e algoritmos de mineração analisam o enorme volume de dados obtido nestes repositórios buscando correlações que possam fornecer informações para o entendimento das condições evolutivas das espécies e processos de formação de doenças (WU et al., 2012).

Intermine é um sistema datawarehouse (DW) usado para construir BD biológicos visando formatar e simplificar a análise de dados vindos de diversas fontes (SMITH et al., 2012). Ele utiliza o sistema gerenciador de banco de dados (SGBD) PostgreSQL para armazenamento e processamento de consultas. No entanto, devido à grande quantidade de dados, a utilização de um servidor de banco de dados centralizado mostrou-se ineficiente em alguns casos que envolvem relações com um grande volume de dados.

Este trabalho propõe a utilização de um banco de dados distribuído (BDD) para resolver este problema. Fazendo algumas modificações no esquema de dados do Intermine, um esquema de referência para alocação dos dados nos nós do cluster é sugerido visando obter o melhor aproveitamento possível do paralelismo nesta aplicação específica.

As heurísticas utilizadas para gerar o esquema de alocação aplicado no Intermine são apresentadas na forma de um algoritmo de alocação que analisa qualquer BD e indica o esquema de fragmentação a ser utilizado visando melhorar seu desempenho. O algoritmo é comparado de forma qualitativa e quantitativa com diversas soluções e arquiteturas relacionadas.

1.1 PROBLEMA DE PESQUISA

O BD dos projetos que usam o Intermine - e da maioria dos projetos biológicos em geral - crescem rapidamente (NCBI, 2013). Cada instância do Intermine tem uma base separada, inviabilizando ou dificultando possíveis integrações.

Também pode-se citar o projeto "The Cancer Genome Atlas"(TCGA) (NIH, 2005) como exemplo para demostrar o crescimento das bases de dados biológicas nos tempos atuais. O projeto gerou até o momento mais de 1PB de dados brutos visando análise dos

fatores causadores do câncer.

Muitas vezes são usados sistemas de banco de dados relacionais (BDR) para armazenar e analisar tais dados (GUIZELINI, 2010). O problema é que, à medida que o volume de dados e acessos crescem, o desempenho de tais sistemas é degradado e o custo de manutenção cresce bastante, tornando o crescimento da plataforma inviável (ÖZSU; VALDURIEZ, 2011).

1.2 CONTEXTUALIZAÇÃO DO PROBLEMA

Está claro que a cura ou, ao menos, detecção prematura de diversas doenças genéticas como alguns tipos de câncer, Alzheimer e Parkinson está relacionada com o entendimento dos fatores causadores e manipulação do código genético e proteômico dos organismos (STURGEON et al., 2003).

A ciência da computação contribui com a pesquisa de algoritmos e tecnologias para análise de dados biológicos. O emprego de BDDs é especialmente aplicado neste tipo de pesquisa possibilitando o armazenamento e processamento escalável das informações.

O computador é o meio que permite que as pesquisas avancem o entendimento humano sobre os fatores causadores das doenças. Quanto melhores forem as tecnologias que suportam os estudos, melhores serão os resultados obtidos nas pesquisas (STURGEON et al., 2003).

Os avanços recentes nas tecnologias para extração de dados biológicos, como Microarranjos (PAUL; IBA, 2009), Pirosequenciamento (RONAGHI; UHLéN; NYRéN, 1998) e Espectrometria de Massa (SPARKMAN, 2006) permitem mapear a composição química de uma determinada substância com rapidez. É possível, por exemplo, extrair células cancerosas de um paciente e mapear trechos do genoma em minutos. Segundo Wu (2012), detecção de biomarcadores e classificação do câncer são assuntos correlatos, visto que só é possível saber o tipo do câncer se for encontrado um marcador do tipo da célula problemática.

As estatísticas do Genbank (NCBI, 2013) se comparadas com os dados das décadas anteriores mostram claramente o crescimento das bases biológicas e a necessidade de soluções computacionais mais eficientes para o processamento destes dados (WEISS, 2010).

Essa possibilidade exige também um avanço na velocidade que os dados extraídos são analisados. Com os processadores alcançando o limite em desempenho, a alternativa é apelar aos sistemas distribuídos como forma de aumentar a velocidade de processamento através da pa-

ralelização das operações. Acreditando nisto, surgiram alguns projetos como o Berkeley Open Infrastucture For Network Computing (BOINC) (ANDERSON, 2004).

1.3 HIPÓTESE

É possível resolver o problema da degradação de desempenho dos BDs relacionais que armazenam informações biológicas, principalmente as operações *Online Analytical Processing* (OLAP) (CODD; CODD; SALLEY, 1993), fragmentando a base centralizada em um BDD. Criando um esquema de alocação padrão para a ferramenta Intermine, seria possível aplicar a solução para todos os seus subprojetos e, possivelmente, especializar tal solução para outros tipos de aplicações.

A segunda hipótese é que é possível encontrar um esquema de alocação que fragmente a base de forma equilibrada quanto ao volume de dados entre os nós, ou seja, que mantenha um volume de dados aproximado ou igual em cada nó do sistema, visando paralelizar as operações ao máximo e minimizar os recursos utilizados por cada nó.

1.3.1 Justificativa da hipótese

Sistemas distribuídos melhoram o desempenho de programas que podem ser executados de forma simultânea em vários computadores (COULOURIS et al., 2011). No entanto, a melhora depende de quanto o processo de negócio e os algoritmos utilizam os recursos compartilhados pela grade.

Para que a proposta seja válida, faz-se necessário desmembrar os algoritmos em "pedaços" independentes e implementar métodos de sincronismo entre os mesmos. Só assim poderá haver ganho (escalabilidade) com o uso dos sistemas distribuídos.

De acordo com Ozsu (2011), as principais promessas e características dos sistemas de BDDs são:

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

Sturgeon et al. (2003) descreve como os sistemas distribuídos podem auxiliar nas tarefas de análise de dados biológicos. Ele também comenta que existe uma grande lacuna de entendimento justamente porque as tecnologias da época não permitiam a análise eficiente do enorme volume de dados disponível.

1.4 OBJETIVOS

Nesta seção são descritos os objetivos do trabalho.

1.4.1 Objetivos Gerais

Os principais objetivos do trabalho são:

- Propor um método para identificação de esquemas de alocação de dados escaláveis e com bom desempenho para BDs distribuídos;
- Automatizar o método proposto através da implementação de um algoritmo de alocação e avaliar o seu funcionamento no DW Intermine;

1.4.2 Objetivos específicos

- Avaliar o desempenho do esquema usando as consultas mais problemáticas do Intermine;
- Comparar o esquema com outras propostas como Bellatreche, Cuzzocrea e Benkrid (2010), HadoopDB (ABOUZEID et al., 2009) e Daudpota (1998);
- Executar experimentos com diversos cenários de alocação de dados e recursos computacionais para verificar o impacto da aplicação do algoritmo no BD do Intermine.

1.5 MÉTODO DE PESQUISA

A pesquisa é, essencialmente, experimental por amostragem. São propostas formas de melhorar o desempenho da base de dados do sistema Intermine usando BDDs. As melhorias serão demonstradas por

experimentos e por ensaios estatísticos. Os parâmetros como tempo de consulta serão estimados segundo a distribuição t de Student (GOSSET et al., 1990) com amostras obtidas em experimentos.

As estratégias que geraram ganhos e o conhecimento adquirido no caso do Intermine serão generalizados na forma de um algoritmo de alocação, possibilitando a utilização em qualquer domínio de aplicação. A adaptação do algoritmo para cada aplicação é feita através dos valores dos parâmetros de entrada.

O produto final (algoritmo de alocação) é comparado com outros algoritmos de alocação descritos na Seção 4.1 e são realizados experimentos que comprovam a eficiência do algoritmo em diversos cenários montados a partir do ambiente real de produção do Intermine.

Durante a execução dos experimentos, foram realizadas melhorias no SGBDD utilizado: pgGrid. As alterações e seus impactos no desempenho estão detalhados na Seção 6.4.2.7.

1.6 ORGANIZAÇÃO DO TEXTO

O trabalho está dividido em três partes principais: revisão bibliográfica (capítulos 2 a 4), o algoritmo (capítulo 5) e a implementação e avaliação (capítulo 6). A primeira parte descreve todos os conceitos relacionados ao trabalho e apresenta um estudo das implementações relacionadas na área de BDD. Na segunda parte são descritos o algoritmo e suas fases. A terceira parte apresenta a implementação do algoritmo, a avaliação qualitativa e o estudo de caso relacionado ao Intermine.

2 BANCOS DE DADOS DE LARGA ESCALA

Este capítulo descreve as tecnologias envolvidas com o gerenciamento de grandes bases de dados, fundamentais para análise de dados biológicos.

2.1 *BIG DATA SCIENCE*

Na época em que este trabalho foi escrito, *Big Data*, ou Grande Quantidade de Dados, é citado como um novo paradigma para armazenamento e análise de dados. No entanto, não existem referências na bibliografia e a maioria das informações são descritas em blogs (DEMCHENKO et al., 2013).

Demchenko et al. (2013) descreve *Big Data* como uma nova forma de lidar com grandes quantidades de dados, usando técnicas não usuais e algumas técnicas antigas que haviam sido até descontinuadas. Basicamente, o conceito de *Big Data* é descrito com cinco palavras (em língua inglesa) que começam com a letra V:

- Volume: Grande volume de dados
- Variedade: Diferentes tipos de dados são suportados, como estruturados, não-estruturados, probabilísticos, etc.
- Velocidade: as análises e cargas de dados executam em tempo real ou próximo a isto
- Veracidade: deve ser possível validar os dados, com foco na integridade, consistência e possibilidade de validação estatística
- Valor: Agregação de valor ao negócio

Aplicações *Big Data* geralmente utilizam processamento paralelo e computação em nuvem (MELL; GRANCE, 2011) para conseguir as taxas de processamento, elasticidade e escalabilidade necessárias.

A maioria dos autores cita *Big Data* como um novo paradigma, sendo necessário projetar as aplicações especificamente, ou seja, para quebrar o paradigma atual seria necessário praticamente refazer as aplicações, reprojetar os processos e as integrações entre elas. Erdman, Keefe e Schiestl (2013) descreve a mudança de paradigma atual com foco em aplicações em medicina, enumerando diversas possibilidades de

uso da tecnologia para melhorar e facilitar o trabalho dos profissionais da área.

2.2 TÉCNICAS PARA ESCALAR BANCOS DE DADOS RELACIONAIS CENTRALIZADOS

Este trabalho descreve um algoritmo que atua sobre dados estruturados, logo se adequa bem aos bancos de dados relacionais. Neste tipo de banco de dados, a necessidade de crescimento e velocidade em grandes volumes de dados fez com que fossem desenvolvidas técnicas de otimização para viabilizar este tipo de aplicação.

Nesta seção são descritas inúmeras técnicas que visam melhorar o desempenho das operações em bases de dados relacionais.

2.2.1 Particionamento

Em BD, particionamento é uma técnica de modelagem física que visa subdividir a relação em pedaços menores com o intuito de melhorar o desempenho das consultas. A técnica é utilizada em inúmeros SGBDs e é fundamental para manter grandes relações em um BD centralizado (NAVATHE et al., 1984).

A Figura 1 mostra como funciona o particionamento no Oracle Database (Oracle Corporation, 2007). É possível visualizar três formas de particionamento: intervalo de valores, hash dos valores e lista de valores de um determinado atributo. Cada uma das formas divide a tabela original (*sales*) em partições menores que permitem buscas mais rápidas. Além disso, é possível utilizar dois níveis de particionamento, ou seja, uma partição também pode ser particionada, o que é muito útil em casos específicos.

O trabalho de Parker e Ritchey (2012) mostra claramente os efeitos do particionamento. As consultas em uma tabela particionada chegam a levar *1/n* do tempo da mesma consulta em uma tabela não particionada, onde n é o número de partições.

2.2.2 Indexação

Índices são estruturas de dados utilizadas para descrever dados de uma forma que facilite as operações de busca. Os índices são estruturas ordenadas por um ou mais atributos da relação a qual o mesmo

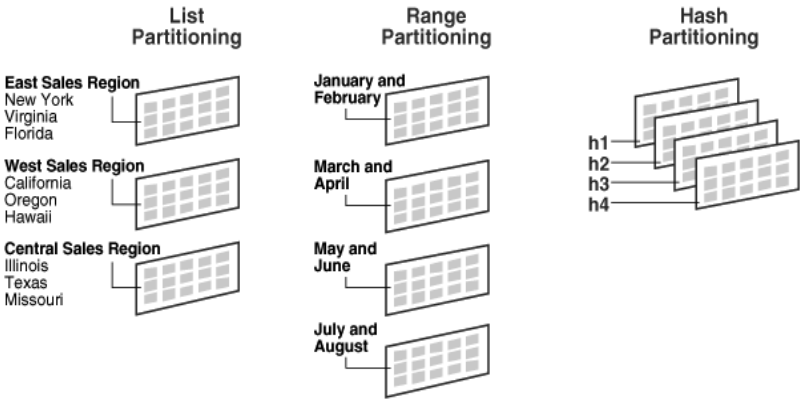


Figura 1 – Partitioning. Fonte: Oracle Corporation (2007)

pertence. Quando uma busca por um dos atributos do índice é disparada, os algoritmos de busca por índice são selecionados pelo planejador de consultas por apresentarem uma melhor eficiência (ELMASRI et al., 2005, p. 326).

Os tipos de índices disponíveis no SGBD PostgreSQL são (PostgreSQL Development Group, 2005b):

- B-tree: utilizam árvores B, que são estruturas de dados projetadas para armazenamento em memória secundária e conseguem manter um bom desempenho (complexidade de busca logarítmica) mesmo com o crescimento da relação indexada. São recomendados para uso geral, principalmente para consultas que envolvem operações de comparação com os operadores =, <, >, <= ou >=;

- R-tree: utilizam árvores R, que são estruturas parecidas com as árvores B, mas adaptadas para funcionar com dados espaciais (posições geográficas, distâncias, formas geométricas, etc). Por esse motivo, índices deste tipo são recomendados principalmente para sistemas de informações geográficas (SIG);

- *Hash*: utilizam funções *hash* para gerar o mapeamento da chave para a posição do dado na estrutura do índice. A utilização deste tipo de índice é recomendada para chaves únicas pesquisadas pelo operador =. O desempenho deste tipo de índice é muito superior

aos demais e a operação de busca de uma chave pode ser executada com apenas algumas instruções de CPU, dependendo da função *hash* empregada;

- Gist: trata-se de um estrutura de dados de árvore balanceada que serve como modelo para que o próprio usuário personalize e defina a estrutura, métodos de acesso e algoritmos de busca. Representam uma das mais importantes inovações do PostgreSQL perante os concorrentes, permitindo flexibilidade para as aplicações.

O PostgreSQL também permite indexar apenas uma parte de uma relação. Com os chamados índices parciais é possível definir um predicado para o índice. Por exemplo:

```
create index indice_xyz on relacao(atributo1 tipo)
where atributo1>atributo2
```

Desta forma, os índices são usados apenas quando a consulta atende o predicado definido no índice. Isso é muito importante quando deseja-se excluir das buscas dados poucos importantes ou históricos.

Índices Bitmap são índices que tem um desempenho melhor quando existem poucos valores distintos (baixa cardinalidade) para os atributos selecionados para o índice (SHARMA, 2005).

As elaboradas estruturas de índices disponíveis atualmente conseguem reduzir a complexidade de busca em várias ordens de grandeza, permitindo realizar operações que seriam inviáveis sem a existência dos mesmos. No entanto, para grandes volumes de dados, mesmo os melhores índices podem não executar as buscas dentro da janela de tempo disponível pela aplicação. Para estes casos, a solução pode ser a utilização de particionamento ou bancos de dados distribuídos, que são descritos na próxima seção.

2.3 BANCOS DE DADOS DISTRIBUÍDOS

Podemos definir um BDD 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. (ÖZSU; VALDURIEZ, 2011, p. 4).

Segundo Bell (1992), um BDD é um conjunto de BDs separados que é visto pelos usuários como um único BD.

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

Ozsu (2009) define os termos esquema conceitual global (ECG) e esquema conceitual local (ECL) para descrever, respectivamente, a visão do esquema de dados que os nós tem do sistema distribuído como um todo e deles mesmos.

2.3.1 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).

2.3.1.1 Fragmentação horizontal

Fragmentar uma relação horizontalmente significa particionar a mesma ao longo de suas tuplas. (ÖZSU; VALDURIEZ, 2011, 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.

2.3.1.2 Fragmentação vertical

A fragmentação vertical de uma relação R produz fragmentos R_1, R_2, ..., R_n que contém um subconjunto dos atributos de R e deve conter a chave primária da mesma. (ÖZSU; VALDURIEZ, 2011, 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".

2.3.1.3 Fragmentação híbrida

Chama-se de fragmentação híbrida a aplicação da técnica de fragmentação horizontal sobre um fragmento vertical ou vice-versa (ÖZSU; VALDURIEZ, 2011, p. 153).

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.

A Figura 2 ilustra os três possíveis tipos de fragmentação.

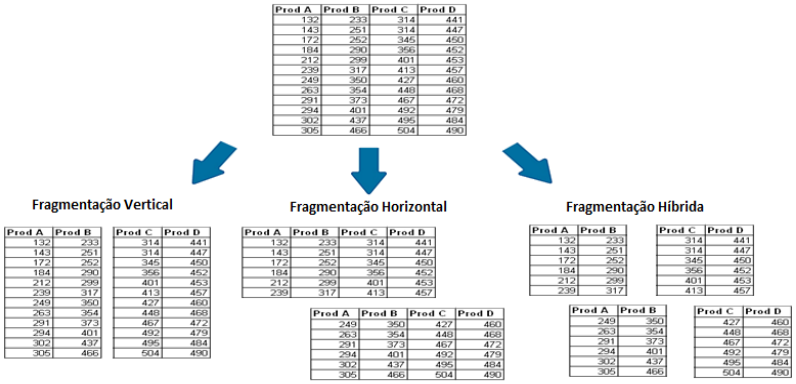


Figura 2 – Tipos de fragmentação

2.3.2 Alocação dos fragmentos

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

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

- Custo mínimo
- Desempenho

Muito estudo já foi dedicado ao assunto de produzir fragmentação ótima (ÖZSU; VALDURIEZ, 2011, p. 163). No entanto, este

assunto não será abordado neste trabalho, onde propõe-se uma solução por aproximação baseada em heurísticas.

2.3.3 Regras de uma fragmentação correta

Özsu e Valduriez (2011) definem três regras para a fragmentação, que juntas garantem que o BD relacional não sofre modificações semânticas devido à fragmentação:

- `Completude:` 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.

- `Reconstrução:` 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.

- `Disjunção:` Se uma relação R é decomposta horizontalmente em n fragmentos e uma tupla está em um destes fragmentos, a mesma tupla 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 $i! = j$, 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.

2.3.4 Sistema gerenciador de bancos de dados distribuídos

[...] Um SGBD distribuído é o software que permite a administração do BDD a deixa a distribuição transparente para os usuários. (ÖZSU; VALDURIEZ, 2011, 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 BDD formas de definir a maneira com que os dados são fragmentados e como devem ser alocados.

2.3.5 Otimização de consultas distribuídas

Da mesma forma que em um BD centralizado, é necessário decompor as consultas e gerar uma árvore algébrica para o otimizador de consultas (ÖZSU; VALDURIEZ, 2011, p. 245). O otimizador, por sua vez, faz modificações na árvore visando melhorar a eficiência da execução.

Inicialmente, gera-se a árvore algébrica como se estivesse em um ambiente centralizado. Depois verifica-se qual a melhor estratégia para reconstruir as relações básicas da consulta. O exemplo da Figura 3 mostra a configuração de uma árvore algébrica comum.

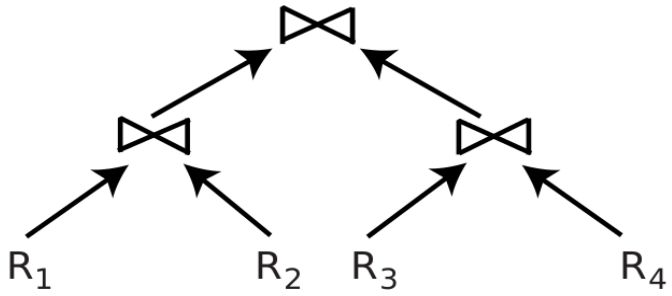


Figura 3 – Árvore algébrica. Fonte: Özsu e Valduriez (2011, p. 248)

O principal fator que afeta o desempenho de uma operação é o tamanho das relações envolvidas. Portanto, é fundamental ter disponíveis estatísticas sobre as relações e índices do sistema. No caso do BDD, é necessário ter as estatísticas de todos os nós, pois elas podem variar de acordo com a alocação.

Também é interessante obter estatísticas sobre as interconexões de rede entre os nós. Isso pode influenciar na ordenação das operações e em qual nó vai executá-las.

Outra otimização possível é a capacidade de realizar agregações em paralelo. O SGBDD deve ser capaz de decompor as agregações em consultas locais - que serão executadas em paralelo pelos nós do sistema - e depois agregar os resultados dos nós para formar o resultado final. Por exemplo, a consulta a seguir deve ser executada em dois passos: (1) enviar a consulta para os todos nós que possuem fragmentos da relação

"sequencefeature" que vão executar e devolver a contagem local. (2) O nó agregador recebe o resultado local de cada nó e agrega-os formando a contagem final.

```
select count(*) from sequencefeature
```

2.3.5.1 Otimização de junções Distribuídas

Em BDD, é possível obter proveito das características dos sistemas. A principal técnica é a escolha do melhor nó para executar junções [p. 262] (ÖZSU; VALDURIEZ, 2011). Imagine uma junção de duas relações que estão em dois nós diferentes. Obviamente, o nó que deve executar a operação é o nó onde a maior parte dos dados estão localizados, visando transferir a menor quantidade possível de dados pela rede.

O protocolo de junções distribuídas baseado em semi junções definido por Özsu e Valduriez (2011) consiste em quatro fases:

1. O nó A lista de valores-chave para o nó destino (B);
2. O nó B executa a semijunção das tuplas cujos atributos-chave pertencem à lista de valores-chave enviadas por A;
3. O nó B retorna o resultado calculado para A;
4. O nó A agrega os dados e continua o processamento da consulta.

Esta abordagem permite, inclusive, o processamento em paralelo de junções de acordo com a posição das operações na árvore algébrica.

2.3.5.2 Otimização baseada em custo

A técnica mais largamente utilizada para escolha da melhor estratégia para execução de consultas é atribuir uma grandeza de custo para cada etapa da estratégia de execução da consulta e, em seguida calcular o custo total da operação. Depois de calcular o custo da todas as estratégias disponíveis, basta escolher a com menor custo (FOMKIN; RISCH, 2007).

Em BDD existem custos adicionais como transferência de fragmentos e instruções pela rede e replicação. Porém algumas operações

são divididas entre os nós do cluster, diminuindo seu custo. Por exemplo: uma busca que teria um custo hipotético X em um banco centralizado pode ter seu custo reduzido para X/n se a relação estiver fragmentada em n nós, explorando o paralelismo.

2.3.6 PostgreSQL e pgGrid

PostgreSQL é um SGBD de código-fonte aberto, multiplataforma, com mais de 20 anos de desenvolvimento. Atende os padrões SQL99 e seu gerenciador de transações implementa as propriedades ACID (acrônimo formado pelas iniciais de atomicidade, consistência, isolamento e durabilidade) (PostgreSQL Development Group, 2005a).

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.

O *pgGrid* (TONINI; SIQUEIRA, 2010) é um sistema de replicação síncrono baseado no PostgreSQL. O sistema permite que os dados sejam fragmentados entre os nós do cluster de acordo com regras estabelecidas através de comandos DDL (*Data Definition Language* - Linguagem de Definição de Dados).

No pgGrid, a definição dos fragmentos e a alocação dos mesmos nos nós é feita através de três comandos DDL (Data Definition Language - Linguagem de Definição de Dados):

- CREATE SERVER: registra um nó no cluster
- CREATE FRAGMENT: define um fragmento
- PLACE: aloca um fragmento em um nó

Outras alternativas para replicação/fragmentação voltadas para PostgreSQL são as ferramentas Slony (Slony Development Group, 2007) e Bucardo (Endpoint Corporation, 2008). Elas permitem a replicação de tabelas específicas entre os nós do cluster. No entanto, não é possível definir regras de fragmentação no estilo do pgGrid, que permitem mais possibilidades de otimização.

2.3.6.1 O otimizador do pgGrid

O pgGrid usa as regras de fragmentação contidas no seu dicionário de dados para saber em quais nós do cluster os dados estão localizados.

Quando uma consulta é submetida para um dos nós, ele precisa primeiro reconstruir as relações fragmentadas para executar as demais operações. Se o arquiteto do sistema fragmentou a base adequadamente, a maioria das junções não devem envolver dados de nós diferentes (evitando transferências através da rede) e as buscas serão executadas em paralelo pelos nós.

O otimizador do pgGrid verifica se as cláusulas de filtragem (*where*) da consulta combinam com as regras de fragmentação definidas pelo arquiteto. Se ocorre "casamento" entre essas regras, a consulta é enviada somente para os nós para os quais os fragmentos estão alocados, melhorando o tempo de resposta.

Por exemplo, se o fragmento foi criado e alocado usando os comandos:

```
--Define o fragmento
CREATE FRAGMENT F ON SEQUENCEFEATURE WHERE CHROMOSOMEID=35;
--Aloca o fragmento em um servidor
PLACE F ON SERVIDOR1;
PLACE F ON SERVIDOR2;
```

A consulta abaixo será enviada apenas para o "SERVIDOR1" e o "SERVIDOR2" que executarão a busca em paralelo.

```
select id from sequencefeature where chromosomeid=35;
```

Na versão 0.1, ainda não estão implementadas otimizações para agregações paralelas - executar agregações locais e depois globais conforme Özsu e Valduriez (2011). Isso significa que as agregações são executadas em um único nó, forçando a reconstrução das relações envolvidas sempre.

A versão atual do pgGrid também não prevê otimização baseada em custos e estatísticas das relações fragmentadas. Nestes casos, todas as otimizações são baseadas em heurística. As otimizações por custo permitem, por exemplo, verificar qual o melhor nó para executar junções ou, se dois nós tem condições de executar a mesma consulta, qual deles vai terminar com menor custo.

2.4 MAP-REDUCE

Map-Reduce é um modelo de programação paralela, que visa dividir um grande problema em problemas menores (DEAN; GHEMAWAT, 2008). O processo de Map-Reduce é dividido em duas fases:

- **Map**: A grande tarefa é dividida em tarefas menores e enviadas para os nós do cluster.
- **Reduce**: O resultado do processamento de cada nó é agregado para formar a resposta final do problema.

O modelo é muito usado em programação paralela e distribuída. O Map-Reduce é utilizado, por exemplo, na execução de consultas do pgGrid.

Para exemplificar o modelo, pode-se considerar o problema de somar cem números. A tarefa pode ser distribuída (*map*) para dez nós, onde cada nó somaria dez números, e depois os resultados das somas de cada nó seriam agregados (*reduce*) para formar a soma final. Qualquer problema deste estilo (paralelizável ou escalável) pode ser implementado utilizado o modelo Map-Reduce.

2.5 CLASSIFICAÇÃO DOS SISTEMAS DE INFORMAÇÃO EM RELAÇÃO AOS DADOS

Esta seção pretende listar alguns tipos mais comuns de sistemas de BD segundo o tipo de informação armazenada e a estratégia de modelagem. A classificação aqui utilizada será utilizada mais tarde para personalizar a aplicação do algoritmo, tema deste trabalho.

Sistemas de informações geográficas (SIG) ou, em inglês, Geographic information systems (GIS), são sistemas que lidam com dados geográficos (coordenadas, distâncias, mapas, rotas, etc). Os principais requisitos de bases de dados SIG são a otimização no armazenamento das estruturas de dados e agilidade para processamento destas informações. Um SGBD com suporte a SIG geralmente oferece estruturas de dados e algoritmos prontos para manipulação das informações geográficas (BERRY, 1996).

Sistemas Multimídia são tipos de sistemas que utilizam as chamadas mídias contínuas (vídeos, animações, imagens) para comunicação (ANDERSON et al., 1991). Este tipo de sistema necessita de ferramentas específicas para edição e tem requisitos diferentes dos de-

mais tipos de sistemas, como a necessidade de menor latência para comunicação e maior espaço de armazenamento por objeto.

Sistemas de tempo real são sistemas que possuem uma restrição de tempo para executar um determinado conjunto de operações (KOPETZ, 1997). Por exemplo: um sistema de controle de tráfego aéreo deve atualizar a situação de todas as aeronaves uma vez por minuto. Essa restrição exige que a infraestrutura e os subsistemas sejam planejados para atender a demanda dentro da janela de tempo necessária.

Datawarehouses (DW) são BDs modelados especificamente para execução de consultas complexas e para a geração de informações para tomada de decisão (geralmente vindas da agregação de dados com menor granularidade) (MOHAJIR; LATRACHE, 2012). Um DW é formado a partir de três processos: coleta dos dados de fontes externas, transformação para facilitar a recuperação e carga no banco de dados de destino.

O esquema de dados de um DW é pensado para favorecer o desempenho das consultas. Muitas vezes a normalização é deixada de lado visando obter ganhos no desempenho. Desta forma, os DWs guardam informações repetidas e com uma estrutura diferente de um BD convencional (normalizado).

O modelo multidimensional (THOMSEN, 2002) é utilizado para sumarizar e organizar dados visando facilitar a recuperação de informação. Os atributos das tabelas de fatos são decompostos em dimensões. Os dados são agregados formando múltiplas possibilidades de visualização de acordo com as dimensões. Os conjunto de possibilidades de visualização das informações agregadas são chamados de cubos.

Para exemplificar, a Figura 4 mostra um cubo gerado a partir de uma tabela que armazena as vendas de um determinado produto com as dimensões cidade, produto e tempo. O atributo "valor vendido" poderia ser agregado entre estas dimensões permitindo inúmeras possibilidades de consulta.

OLAP é uma abordagem que utiliza a modelagem multidimensional para responder consultas complexas rapidamente (JAECKSCH; LEHNER, 2013). *Online Transaction Processing* (OLTP), ao contrário dos sistemas OLAP, são sistemas voltados para o processamento de pequenas transações e consultas simples. Geralmente as tabelas de fatos que são a base para a formação dos sistemas OLAP são originadas em sistemas OLTP.

Datamarts são um subconjunto de dados de um DW (MOHAJIR; LATRACHE, 2012). Geralmente os dados sobre um determinado assunto são separados física ou logicamente por motivos de segurança

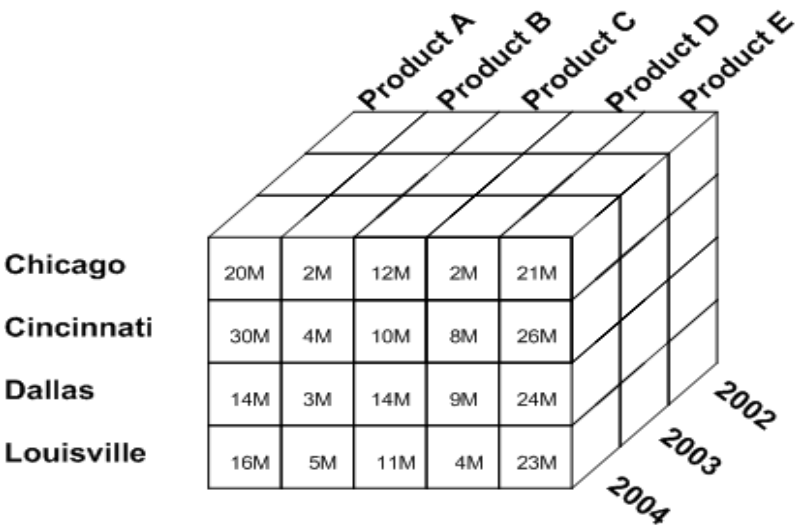


Figura 4 – Exemplo de cubo OLAP. Fonte: Oracle Corporation (2005)

ou simplesmente coesão.

Também pode-se classificar as aplicações segundo o recurso físico mais utilizado. Desta forma, é possível melhorar seu desempenho priorizando o acesso a este recurso ou entregando-o em maior quantidade, quando possível. Neste contexto, Jun e Tei-Wei (1999) define aplicações dependentes da Unidade Central de Processamento (*CPU-Bound*) como aplicações que gastam a maior parte do tempo em operações de CPU. Da mesma forma, aplicações *I/O-bound* são aplicações que passam a maior parte do tempo em operações de entrada/saída (geralmente acesso a armazenamento secundário em disco).

2.6 CONSIDERAÇÕES FINAIS

Este capítulo descreveu técnicas utilizadas para melhorar o desempenho e escalar BD centralizados como indexação e particionamento. Com o mesmo intuito foi introduzido o conceito de BDDs.

Conceitos mais recentes como *Big Data Science* também foram introduzidos para fins de comparação. Tais técnicas serão utilizadas nos próximos capítulos.

Finalmente foi estabelecida uma classificação das aplicações que utilizam BDs para guiar a utilização do algoritmo durante a avaliação qualitativa (Seção 6.3).

3 BANCOS DE DADOS BIOLÓGICOS

Um BD biológico é qualquer sistema de BD que armazena informações biológicas. Cada base armazena um tipo de informação que pode variar desde amostras colhidas em laboratórios até diagnósticos médicos (BOLSER et al., 2011). Estas bases geralmente se relacionam e estão disponíveis para o público na Internet.

Este trabalho é mais focado nas bases de dados de sequências de aminoácidos (DNA e RNA), proteínas e nas informações contidas nestes códigos que possam ter correlações e resolver problemas reais, principalmente no sistema datawarehouse Intermine (SMITH et al., 2012), que permite a consolidação de dados vindos de outras bases o cruzamento destas informações visando produzir novos conhecimentos.

3.1 GENOMAS, PROTEOMAS, VIAS METABÓLICAS E BIOMARCADORES

O processo químico que regula o funcionamento de um organismo é extremamente complexo. Tal processo é regulado por sequências de pares de bases nitrogenadas de DNA. A sequência está replicada em cada célula do organismo, dividida em estruturas chamadas cromossomos. O conjunto completo de sequências forma o genoma (SNUSTAD; SIMMONS, 2010).

O genoma é transmitido de forma hereditária na reprodução, onde sofre recombinações. O código genético se combina com diversas substâncias no interior da célula e gera outras sequências (RNA) que servem de "fôrma" para a produção de proteínas.

As proteínas, por sua vez, desempenham funções fundamentais no organismo, podendo citar: constituição dos tecidos, defesa, armazenamento de energia, catalisação de reações químicas. O conjunto contendo todos os tipos de proteínas de um organismo chama-se proteoma.

O ciclo metabólico celular é composto por séries de reações químicas, catalizadas por enzimas, que regulam o funcionamento do organismo. Cada série separada deste ciclo é chamada de via metabólica (NELSON; COX, 2004, p. 49).

Em cada via, uma substância é modificada por reações e se transforma em outra. O ciclo é regulado pelo código genético e por mensagens internas/externas que a célula gera ou recebe do meio em que se

encontra. É possível mapear as vias metabólicas através de um mapa. Este mapa é uma espécie de fluxograma das operações realizadas pela célula.

Algumas doenças causadas por mutações genéticas causam falhas no ciclo metabólico causando deturpações nas operações celulares. Conhecer quais mutações (biomarcadores) provocam a alteração em uma determinada via é importante para identificar e prevenir as consequências geradas pela mutação.

Biomarcadores são substâncias biológicas que podem ser medidos/detectados e indicar normalidade ou anormalidade (expressão de uma doença, por exemplo) (STRIMBU; TRAVEL, 2010).

3.2 TÉCNICAS DE EXTRAÇÃO DE INFORMAÇÕES BIOLÓGICAS

Diversas tecnologias permitem extrair dados do genoma ou proteoma dos organismos. Tratam-se de processos complexos que são a entrada dos demais processos de análise que tentam descobrir as funções dos genes e proteínas (PETTERSSON; LUNDEBERG; AHMADIAN, 2009).

As próximas seções descrevem algumas técnicas utilizadas atualmente para extrair informações biológicas.

3.2.1 Espectrometria de massa

Espectrometria de massa é um método que usa um equipamento chamado espectrômetro para bombardear moléculas com um feixe de íons ou elétrons que atravessam um campo magnético e sua trajetória depende da massa do elemento. Desta forma, é possível identificar quais elementos compõem a substância (BIENVENUT, 2005).

A espectrometria foi inventada em 1919 pelo físico britânico Francis William Aston e permite a caracterização rápida de microorganismos e proteínas.

3.2.2 Microarranjos

Um microarranjo de DNA ou chip de DNA é uma técnica de mapeamento que gera uma matriz com a representação de um segmento da sequência genômica ou protéica de um organismo. A técnica permite a análise da expressão gênica e função do produto dos genes (EKINS;

CHU, 1999).

A técnica é muito usada em genômica funcional (área que busca encontrar a função de genes e proteínas). É possível, por exemplo, usar o método para tentar identificar as mutações genéticas responsáveis por uma determinada doença, extraindo um trecho suspeito da sequência de DNA do paciente e comparando com a sequência de um indivíduo normal (LASHKARI et al., 1997).

Existem diversos repositórios de dados obtidos pela técnica, como Stanford Microarray Database (HUBBLE et al., 2009), Gene Expression Omnibus (GEO) e Princeton University MicroArray database.

3.2.3 Sequenciamento de DNA

Métodos de sequenciamento são procedimentos bioquímicos que visam determinar a ordem da composição de uma sequência de DNA (NELSON; COX, 2004).

No método de Sanger (SANGER; NICKLEN; COULSON, 1977), a fita de DNA é hibridizada com um Primer de desoxinucleotídeos que para a reação. Depois um gel de sequenciamento por eletroforese é aplicado para detectar cada uma das bases presentes na sequência.

Durante o complexo processo de síntese de DNA, ocorre a liberação do pirofosfato, que gera luz detectável. Tal reação pode ser detectada por um equipamento chamado pirosequenciador (RONAGHI; UHLéN; NYRéN, 1998). Desta forma, o DNA pode ser sequenciado quase na velocidade em que é sintetizado, fornecendo uma ferramenta eficiente para o mapeamento de pedaços de genoma. As principais vantagens da técnica são a precisão do resultado, a velocidade e, principalmente, a possibilidade de processamento paralelo.

Além das duas técnicas descritas brevemente aqui, Jay e Ji (2008) descrevem as diversas gerações de técnicas de sequenciamento e os principais métodos de cada uma das gerações.

3.3 REPOSITÓRIOS DE INFORMAÇÕES BIOLÓGICAS

As informações geradas pelos milhares de centros de pesquisa, usando as tecnologias de sequenciamento e análise citadas anteriormente são armazenadas em diversos repositórios. A maior parte deles tem acesso público liberado através da Web.

Pode-se citar como exemplos de repositórios: Genbank (BENSON

et al., 2005), Banco de Sequências do Laboratório Europeu de Biologia Molecular (EMBL) (COCHRANE; GALPERIN, 2010), o BD de DNA do Japão (DDBJ) (DNA DATA BANK OF JAPAN, 2014) e o Portal de Dados do projeto TCGA (NIH, 2005).

Cada repositório geralmente é voltado para uma espécie específica ou para um objetivo, como pesquisa de uma doença ou busca de medicamentos.

Na web também estão disponíveis "metarepositórios" contendo o catálogo dos repositórios biológicos disponíveis. Os metarepositórios mais famosos são o Metabase (BOLSER et al., 2011) e o *Registry of Biological Repositories* (NCBI, 2005) com mais de 6000 repositórios catalogados.

A próxima seção detalha melhor o sistema Intermine (utilizado para construção de BD biológicos) e cita os motivos pelos quais ele foi escolhido como objeto de estudo deste trabalho.

3.3.1 Intermine

Intermine é um sistema DW que pode ser usado para construir bases biológicas com o intuito de formatar e facilitar a análise de dados de diferentes fontes externas (SMITH et al., 2012).

Diversos projetos são baseados no Intermine, podendo-se citar:

- Flymine: dados de *Drosophila* (popularmente conhecida como mosca das frutas) muito estudada por ter padrões genéticos parecidos com os humanos;
- Ratmine: ratos;
- Modmine: elementos genéticos funcionais das espécies *C. elegans* e *D. melanogaster*;
- Targetmine: dados de genes favoráveis para produção de medicamentos;
- Yeastmine: *D. melanogaster* (fungo).

A maior parte dos dados vem de outras bases biológicas. Os dados são processados e são geradas visões para análises específicas. Os usuários podem montar as consultas através de uma interface web.

Atualmente, uma nova instância do Intermine é criada cada vez que um foco de análise diferente é necessário, dificultando a integração entre os *mines*. Quando uma integração entre os mines é necessária,

os dados são duplicados entre as instâncias e cada uma delas aparece para as demais como fonte externa.

O SGBD usado pelo Intermine é o PostgreSQL. Esta escolha se deve à possibilidade de criação de novos tipos de dados e índices com o GIST (mecanismo para criação de novos tipos de índices pelos usuários), além das funções avançadas para processamento de sequências de caracteres (strings). O PostgreSQL também possui a biblioteca *bioseg* (BIOINFORMATICS, 2008) com muitas funções úteis para a área biológica.

A característica mais interessante presente no Intermine (em relação a este trabalho) é que seu esquema de dados, gerado a partir de ontologias e classes Java, pode ser generalizado para praticamente qualquer aplicação na área biológica.

3.4 CONSIDERAÇÕES FINAIS

Este capítulo descreveu as principais técnicas para extração de informações biológicas e alguns repositórios onde elas são armazenadas, bem como sua estrutura e integração. Também foi apresentado o sistema DW Intermine, objeto de estudo neste trabalho.

Estes conceitos são necessários para entender as premissas, o impacto e onde os resultados deste trabalho podem ser aplicados.

4 TRABALHOS RELACIONADOS

Este capítulo descreve outros esquemas/algoritmos de alocação de dados distribuídos e algumas ferramentas que visam automatizar a solução deste problema, provendo a fragmentação das relações e a alocação de forma automática. Estas propostas são utilizadas para comparação durante a avaliação do algoritmo proposto neste trabalho.

4.1 OUTROS ESQUEMAS DE ALOCAÇÃO

A busca por um esquema ideal para alocação de dados distribuídos é, atualmente, objeto de estudo na área de BD.

Nos trabalhos de Daudpota (1998), Ahmad (2002) e Adl (2009) são propostos algoritmos para gerar a alocação de dados de uma base de dados genérica, baseando-se nas consultas mais comuns. Estes algoritmos aplicam heurísticas para dividir o BD de uma forma que não prejudique as consultas que o mesmo analisa, fornecendo uma alocação que explore o paralelismo dos sistemas de BDD. Cada algoritmo é avaliado utilizando modelos matemáticos que possam dar uma ideia do efeito da aplicação do algoritmo em uma base de dados genérica.

Já Abdalla (2012) construiu seu algoritmo de alocação baseado em heurísticas e visando equilibrar os custos de armazenamento entre os nós que compõem o sistema.

Cada esquema de alocação baseado em consultas tenta melhorar o desempenho utilizando algumas técnicas. Por exemplo, Daudpota (1998) aloca os dados com os seguintes objetivos:

- Aumentar a disponibilidade: replicar os dados para evitar indisponibilidade em caso de queda de um subconjunto do cluster;
- Minimizar os custos de comunicação: deixar dados correlatos no mesmo nó para resolver consultas localmente;
- Explorar o paralelismo: utilizar o maior número de nós possível para executar uma consulta, visando dividir a tarefa para terminar o mais rápido possível.

Bellatreche (2012) sugeriu que a fragmentação de DWs melhora o desempenho das consultas. A solução no trabalho é voltada inicialmente para DWs baseados em BD relacionais. A fragmentação é feita estipulando um limite de custo de processamento e armazenamento,

ou seja, o algoritmo tenta encontrar a melhor solução para executar as consultas com uma determinada eficiência, mas usando um número determinado de recursos. A Figura 5 permite visualizar melhor as etapas do algoritmo. A primeira fase (retângulo superior - fragmentação) tem como entrada os parâmetros definidos pelo usuário (requisitos e detalhes sobre as regras de negócio da aplicação) e as consultas mais comuns e como saída o esquema de fragmentação. A segunda fase (retângulo inferior - alocação) tem como entrada o esquema de fragmentação e os limites de armazenamento e gera o esquema de alocação.

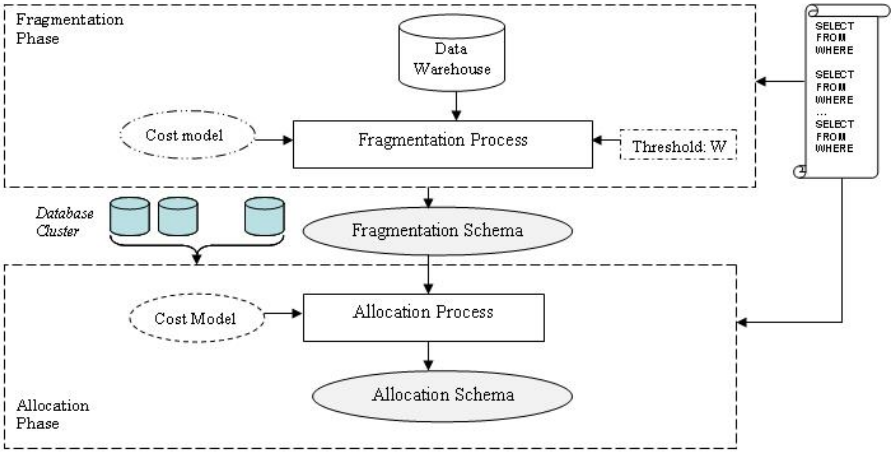


Figura 5 – Fases do algoritmo de alocação proposto por Bellatreche, Cuzzocrea e Benkrid (2010)

Nenhuma das pesquisas analisadas faz menção a dados biológicos. Isso ressalta a existência dessa lacuna nas pesquisas da área de BD que afeta também as pesquisas em biotecnologia, pois sem essas soluções não é possível analisar com eficiência o enorme volume de dados disponível (MUSHEGIAN, 2011).

4.1.1 Auto-Sharding

Podemos definir um *shard* (palavra em inglês para "pedaço") em BD como um fragmento horizontal. Segundo 10gen (2011), *auto-sharding* é definido como o processo de dividir automaticamente uma

base de dados entre um número limitados de máquinas visando escalar o sistema de acordo com a demanda dos usuários. Desta forma, quanto mais nós são adicionados ao cluster, maior se torna a capacidade do sistema em responder as requisições e analisar os dados armazenados.

Neste contexto, diversos softwares buscam se adaptar automaticamente ao crescimento dos dados com o mínimo possível de interferência dos administradores. MongoDB (10GEN, 2007) e o MySQL Cluster (Oracle Corporation, 2004) são dois exemplos comerciais que utilizam a técnica de Auto-Sharding como um requisito básico para prover escalabilidade e alta disponibilidade.

O MongoDB possui funções de balanceamento e migração de dados entre os nós do cluster. Conforme mais nós são adicionados ao sistema, o software é capaz de rearranjar os dados de forma com que os nós tenham uma quantidade equilibrada de dados. A Figura 6 ilustra bem o processo de adaptação do cluster MongoDB depois que um novo nó é adicionado.

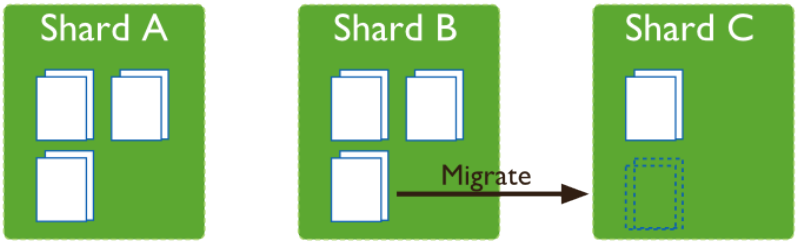


Figura 6 – Balanceamento de *shards* no MongoDB. Fonte: 10gen (2011)

No MongoDB a divisão dos dados entre os nós é feita através de um algoritmo de *hash* que calcula em qual nó determinado registro deve ficar. Portanto, podem acontecer situações em que dados correlatos sejam armazenados em nós diferentes. Já no MySQL Cluster, os dados vão sendo alocados conforme são requisitados em cada nó. Desta forma, dados relacionados (que são utilizados juntos) geralmente ficam disponíveis na mesma máquina, evitando custos adicionais para transferir estas informações entre os nós do cluster.

Como a técnica de *sharding* automático se encaixa bem no contexto deste trabalho, julgou-se interessante realizar testes para comparar o desempenho da alocação automatizada em contraste com as demais técnicas discutidas aqui. A ferramenta que aplica a técnica

de *Auto-Sharding* selecionada para os testes foi o HadoopDB que será descrito a seguir.

4.2 HADOOPDB

Hadoop (APACHE, 2005) é um framework para implementação de aplicações baseadas no paradigma Map-Reduce (DEAN; GHEMAWAT, 2008). Ele disponibiliza toda a implementação da infraestrutura de comunicação entre os nós do cluster, escalonamento de tarefas, replicação e sistema de arquivos distribuído. O usuário do framework precisa apenas implementar os métodos *Map* e *Reduce*.

Obviamente para a aplicação funcionar corretamente e obter os benefícios da computação distribuída, ela deve ser escrita utilizando os componentes fornecidos pelo Hadoop e os métodos *Map* e *Reduce* devem explorar ao máximo os recursos compartilhados pelos nós de processamento disponíveis na estrutura.

O framework Hadoop foi escrito em Java e está disponível para a maioria dos sistemas operacionais conhecidos.

HadoopDB (ABOUZEID et al., 2009) é uma extensão do Hadoop para trabalhar com BD relacionais como nós de processamento. Assim, o HadoopDB simula um BDD onde a entrada pode ser feita através de um comando SQL. A consulta submetida é otimizada para usufruir do paralelismo. A otimização é efetiva, principalmente, nas consultas com funções de agregação.

O HadoopDB divide automaticamente os dados inseridos entre os nós (*Auto-Sharding*) de acordo com um algoritmo de *hash*, mas também é possível definir as regras de alocação utilizando sua API Java (ABOUZEID et al., 2009).

A Figura 7 mostra o diagrama de componentes de um sistema com HadoopDB. Nota-se claramente que a entrada é feita através de SQL (parte superior). O componente "Hadoop Core", por sua vez, reconstrói para usufruir do paralelismo e dispara os trabalhos para os nós de processamento. Cada nó de processamento tem os componentes de comunicação com o nó mestre e um servidor de banco de dados que executa o trabalho pesado.

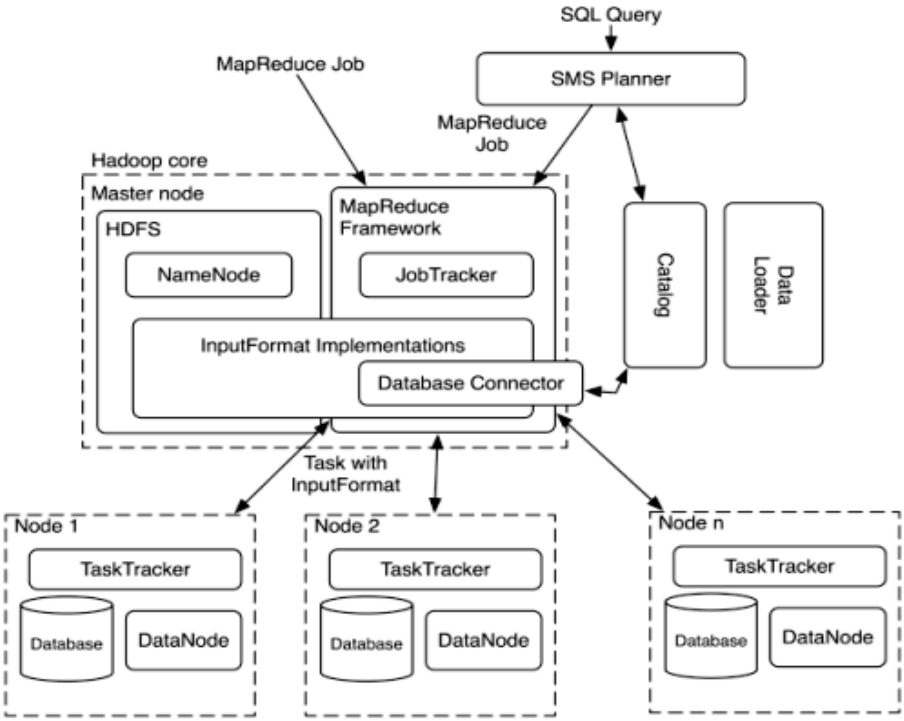


Figura 7 – Componentes do HadoopDB. Fonte: Abouzeid et al. (2009)

4.3 OPENMOLGRID

O OpenMolGrid é um sistema DW (com a mesma finalidade do Intermine) desenvolvido para ambientes de computação distribuída (DUBITZKY et al., 2004). A arquitetura permite adicionar recursos em cada uma das camadas do DW (coleta, transformação e armazenamento). O OpenMolGrid fornece toda a infraestrutura para construção de DWs distribuídos, mas a arquitetura do grid é deixada sob responsabilidade dos usuários. Este trabalho auxiliará também os usuários desta ferramenta a escolher o melhor esquema de alocação para suas aplicações.

Construído utilizando o framework UNICORE (UNICORE Forum, 2003), o OpenMolGrid fornece componentes para todas as camadas (armazenamento, serviço, carga, transformação e cliente) e cada classe de componente é preparada para fornecer balanceamento de carga. Por exemplo, se mais um nó for adicionado na camada de armazenamento, as bases vão dividir o trabalho até ficarem com o mesmo volume de dados e de requisições.

Como o balanceamento de carga é feito automaticamente, não é possível programar a alocação dos recursos para atender determinadas classes de requisições ou deixar dados relativos ao mesmo assunto mais próximos ou no mesmo nó. Isso faz com que o OpenMolGrid tenha o mesmo "problema"de otimização de consultas que o HadoopDB, sendo necessário intervenção no código-fonte ou programação manual de serviços para usufruir do paralelismo ou alocar os recursos de forma personalizada.

4.4 CONSIDERAÇÕES FINAIS

Este capítulo descreveu alguns trabalhos relacionados e ferramentas que serão utilizadas como base de comparação neste trabalho. Juntamente com os dois capítulos conceituais anteriores permitiram a generalização dos conceitos para chegar à proposta da pesquisa que será apresentada a seguir.

5 ALGORITMO DE ALOCAÇÃO

De acordo com as motivações deste trabalho, foram analisadas diversas aplicações biológicas baseadas no Intermine e a estrutura de suas bases de dados com o intuito de estabelecer um padrão e fornecer linhas que guiassem a elaboração de uma solução para otimização de tais sistemas. Pensou-se inicialmente em otimizar algumas consultas específicas e modificar o esquema de dados, desnormalizando-o de acordo com Shin e Sanders (2006) de forma a favorecer as consultas. No entanto, algumas das propostas se tornaram inviáveis pois o custo (processamento e tamanho das relações) para gerar as relações desnormalizadas ficou alto demais. Muitas consultas fazem junções entre tabelas grandes, geralmente com sequências de DNA. Materializar tais junções seria inviável, evidenciando a necessidade de usar BDDs para dividir a carga e diminuir o espaço de busca.

Os esquemas de dados analisados seguem uma modelagem objeto-relacional, geralmente gerada a partir de ferramentas que convertem classes Java para tabelas de bancos de dados relacionais. O Anexo A contém um exemplo, retirado de um subconjunto do esquema de dados do Intermine. Seguindo esta ideia, é possível fragmentar as relações por alguns atributos de ligação entre classes (identificadores das relações). Nos experimentos iniciais realizados, notou-se que a divisão seguindo esta lógica permite, muitas vezes, uma cardinalidade linear, ou seja, fragmentos com tamanhos parecidos.

Os experimentos iniciais realizados com algumas consultas coletadas do *log* de instâncias do Intermine também mostraram que é possível obter melhorias significativas de desempenho, indicando que muitas consultas são baseadas em entidades específicas, como cromossomos específicos, conjuntos de genes e organismos de uma determinada espécie. Nestas consultas, é possível descartar todas as demais entidades do sistema, diminuindo o espaço de busca através da criação de fragmentos que apresentem somente as entidades necessárias para cada tipo de consulta. Os experimentos iniciais estão detalhados na Seção 6.4.2.2.

Para conseguir estabelecer critérios de alocação, as consultas foram agrupadas segundo a similaridade e as bases foram fragmentadas para verificar que tipo de fragmentação atende melhor cada grupo. Foram testadas diversas combinações até que se pudesse definir uma forma de fragmentar as bases visando melhorar o desempenho da maior quantidade de grupos de consultas possível.

Usando esta lógica, notou-se que é possível automatizar a maioria dos testes tendo em mãos algumas informações básicas, como o esquema de dados e o *log* de consultas.

Visando formalizar o conhecimento adquirido com as experiências iniciais e os objetivos presentes no escopo, foi elaborada uma lista de premissas que guiaram a elaboração do algoritmo. Elas são discutidas na próxima seção.

5.1 PREMISSAS

O principal objetivo deste trabalho é propor um algoritmo de alocação de dados distribuídos que melhore o desempenho das consultas. Para alcançar este objetivo, as seguintes premissas foram estabelecidas:

1. Agrupar dados relacionados no mesmo nó de um sistema distribuído para melhorar o desempenho das consultas, visto que os custos de transferência de rede são evitados;

2. Separar dados não-relacionados, ou seja, dados que nunca são utilizados em conjunto, para melhorar o desempenho, porque permite que análises de dados separados executem em paralelo em servidores diferentes. Esta prática também permite escalar o sistema, mantendo nós com o menor volume de dados possível;

3. Fragmentar horizontalmente grandes relações. Desta forma, uma consulta sobre esta relação pode ser executada em paralelo por diversos nós do cluster;

4. Estabelecer o melhor esquema de alocação não é uma tarefa trivial. Cada esquema escolhido pode melhorar algumas consultas, mas também causar impacto negativo em outras. Portanto, apenas testes abrangentes podem gerar conclusões sobre os benefícios e perdas de cada esquema.

5.2 MÉTODO DE TRABALHO

Diversas etapas experimentais foram realizadas para gerar o algoritmo que está sendo apresentado. O fluxograma da Figura 8 descreve as etapas realizadas para a obtenção do algoritmo descrito neste capítulo.

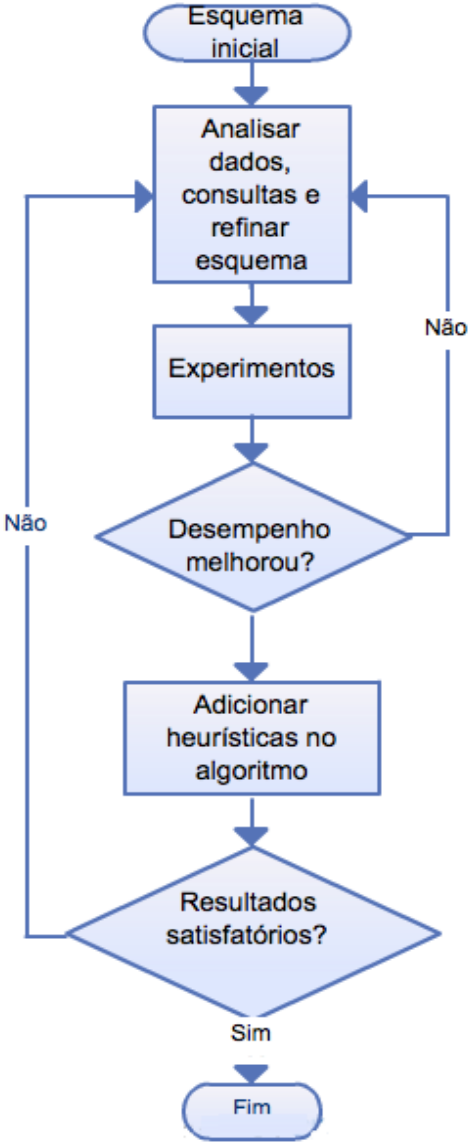


Figura 8 – Fluxograma da criação do algoritmo

Inicialmente o BD utilizado como estudo de caso (Intermine) foi analisado e fragmentado de acordo com heurísticas conhecidas pela experiência do autor juntamente com relatos dos administradores do aplicativo. Depois foram realizados experimentos que testaram a eficácia destas heurísticas. Quando os experimentos mostraram resultados positivos, as heurísticas foram adicionadas no algoritmo para formar o procedimento completo de fragmentação/alocação definido pelo algoritmo.

Somente as heurísticas que geram ganhos nos experimentos são adicionadas ao algoritmo (laço interno do fluxograma).

Todos os passos relatados foram executados de forma iterativa até que os resultados atingidos nos testes de desempenho foram considerados satisfatórios pelos autores deste trabalho (laço mais externo do fluxograma).

5.3 O ALGORITMO

O algoritmo de alocação é um procedimento heurístico e é dividido em três etapas para tratar das premissas discutidas anteriormente. Cada uma delas será detalhada nas próximas seções.

5.3.1 A primeira fase: Análise dos Dados

Na primeira fase, o algoritmo analisa todas as relações e seus atributos, procurando por grandes relações visando agrupar dados de acordo com a similaridade (premissas 1 e 2). O Algoritmo 1 descreve a primeira fase em pseudo-linguagem.

Os parâmetros esperados são

- nn: o número de nós disponíveis para formar o sistema;
- cm: o número mínimo de tuplas a ser armazenado em cada nó.

A primeira fase verifica cada atributo contando o número de valores distintos presentes e a média de ocorrências de cada valor. Com estes dados verifica se é possível fragmentar a relação de acordo com os parâmetros recebidos. Depois de verificar os parâmetros, a lista de atributos aptos para serem usados na fase seguinte é retornada.

Um histograma com a distribuição de dados de cada atributo pode ser gerado com base na lista de atributos retornada por esta fase do algoritmo. Este histograma pode ser usado para identificar os atribu-

Algoritmo 1 Análise dos dados.

Entrada: cm: inteiro, *núm. mínimo de tuplas para fragmentar relação*
nn: inteiro, *número de nós disponíveis*
Saída: L := ∅ : *lista de atributos que serão usados como predicados para a fragmentação*

```
início
    para cada relação r no BD faça
        n := número de tuplas em r
        para cada atributo c em r faça
            temp := 0
            para cada valor distinto v de c faça
                t := número de tuplas em r onde c = v
                se t >= cm então
                    temp := temp + 1
                fim
            fim
            se temp >= nn então
                adicione r.c em L
            fim
        fim
    fim
fim
```

tos que podem obter melhores resultados nas buscas paralelas conforme o sistema escala.

5.3.2 A segunda fase: Análise das Consultas

Na segunda fase, o resultado da primeira é cruzado com o comportamento das consultas executadas no sistema, visando garantir que a fragmentação das relações vai resultar em melhoria do desempenho de uma boa quantidade das requisições feitas no sistema. Em suma, fragmentar uma relação é uma tarefa custosa. A segunda fase garante que este custo vai trazer benefícios nas consultas.

Para executar esta análise, o *log* de consultas deve estar disponível. O objetivo é verificar quais consultas vão obter benefício se a base for fragmentada de acordo com o resultado da primeira fase. Para isso, são esperados os seguintes parâmetros:

- fm: a frequência mínima para considerar uma consulta. Por exemplo, deseja-se considerar apenas as consultas que representam mais de 10% (fm=0.1) de todas as consultas no *log*;
- tm: O tempo mínimo que uma consulta deve levar para executar. Somente as consultas com tempos maiores que este valor são consideradas na análise.

A segunda fase, descrita em pseudo-linguagem no Algoritmo 2, verifica se os atributos selecionados na primeira fase estão presentes nos filtros (cláusula *where* da SQL) e agregações (cláusula *GROUP BY* da SQL). Os atributos que estão nestas cláusulas obtém melhor desempenho no paralelismo (premissas 1, 3 e 4).

A saída da segunda fase é utilizada na terceira e última, onde a alocação dos dados é definida.

5.3.3 A terceira fase: Geração e Alocação dos Fragmentos

As duas primeiras fases do algoritmo geram uma lista de atributos com perfil favorável para fragmentação.

O objetivo da terceira fase é gerar fragmentos com base nestes atributos e alocar cada fragmento em um nó.

Os parâmetro esperados nesta fase são:

- *nn*: a quantidade de nós disponíveis para a alocação. Trata-se do mesmo parâmetro da primeira fase;

Algoritmo 2 Análise das consultas.

Entrada: *fm*: float, *frequência mínima para considerar uma consulta na análise*
tm: float, *tempo mínimo para considerar uma consulta na análise*
L: *lista de atributos retornada pela fase 1*
log: *o histórico de consultas usando variáveis bind para agrupar consultas similares*

Saída: R := ∅ : *lista de atributos selecionados como predicados*

```
início
    para cada consulta c no log faça
        q(c) := quantidade de ocorrências de c no log
        tempo(c) := tempo médio de execução de c
        calculado com base no log
        f(c) := q(c) / tamanho(log)
        se f(c) > fm e time(c)>tm então
            para cada atributo f usado nos filtros ou
            agregações de c faça
                se f ∈ L então
                    adicione f em R
                fim
            fim
        fim
    fim
    retornar R
fim
```

- L: a lista de atributos retornada pela fase 2.

Os valores distintos de cada atributo são analisados com o intuito de gerar nn fragmentos com quantidades parecidas de tuplas. Para isso, conta-se a quantidade de tuplas de cada valor distinto do atributo e formam-se fragmentos com esses valores até a quantidade necessária de tuplas ser atingida.

A ideia é acumular os valores distintos dos atributos para formar fragmentos com n/nn tuplas, onde n é a quantidade de tuplas da relação e nn é o número de nós. Os valores que sobram são alocados no último nó. Atributos com o mesmo nome, mesmo em relações diferentes, tem seus valores iguais alocados no mesmo nó (Premissa 1), evitando a necessidade de junções distribuídas.

O Algoritmo 3 descreve a Fase 3 em pseudo-linguagem. A Figura 9 mostra o algoritmo completo, suas fases, entradas e interações.

A próxima seção descreve como ajustar o comportamento do algoritmo para diferentes cenários.

5.4 OTIMIZAÇÃO DOS PARÂMETROS

Cada sistema de BD tem características diferentes e deve ser otimizado de forma específica. O administrador de um sistema OLTP, por exemplo, deseja que a maioria das consultas levem pouco tempo para executar. Por outro lado, o administrador de um DW quer melhorar grandes consultas, mesmo que sejam executadas com pouca frequência. Estes objetivos podem ser alcançados ajustando os parâmetros do algoritmo, permitindo otimizar qualquer tipo de sistema.

De acordo com a relevância para o algoritmo, os tipos de sistemas de informação descritos na seção 2.5 podem ser agrupados nas seguintes classes:

1. **Sistemas de tempo real**: exige que os tempos das consultas sejam menores que um limiar aceitável. Portanto recomenda-se estabelecer um número pequeno de nós (parâmetro nn), grande quantidade de tuplas por nó (cm), tempos baixos (menores que o limiar do sistema) de consultas (tm) e frequência baixa das consultas (fm) para que todas as consultas sejam analisadas pelo algoritmo com o intuito de estabelecer uma espécie de tempo máximo de tolerância para as mesmas.

2. **OLAP e sistemas de apoio a decisão**: geralmente executam consultas complexas esporadicamente. Portanto, recomenda-se

Algoritmo 3 Geração e Alocação dos Fragmentos.

Entrada: *nn*: inteiro, número de nós disponíveis
L: lista de atributos selecionados na fase 2
Saída: A alocação de todas as tuplas nos nós disponíveis
início
  para cada *nome de atributo distinto a em L* **faça**
    t := a relação em L com maior quantidade de tuplas que possua o atributo a;
    n := quantidade de tuplas em t;
    $acum := 0$;
    $i := 1$;
    $T := \emptyset$;
    conte a quantidade de cada valor distinto de $t.a$;
    para cada *valor distinto v de t.a ordenado pela quantidade* **faça**
      se $i = nn$ **então**
        aloque as tuplas restantes de todas as relações que possuem o atributo a no nó i;
        saia do loop;
      fim
      $acum := acum$ + quantidade(v);
      adicione v na lista T;
      se $acum >= n/nn$ **então**
        aloque as tuplas de todas as relações onde $a \in T$ no nó i;
        $i := i+1$;
        $acum := 0$;
        $T := \emptyset$;
      fim
    fim
  fim
fim

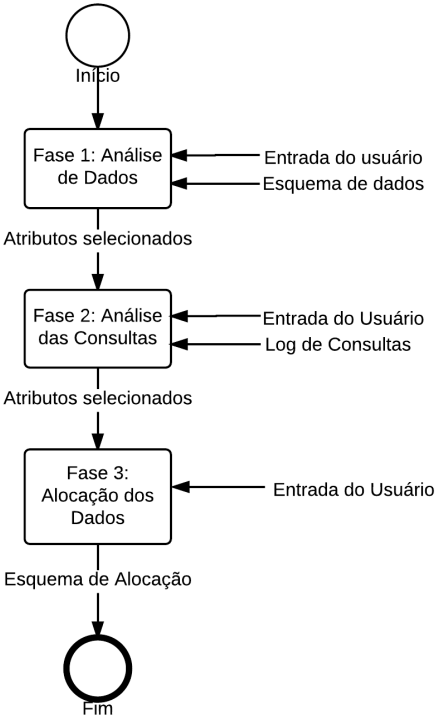


Figura 9 – Visão geral do algoritmo

usar um valor baixo para o parâmetro *fm* e alto (depende da aplicação) para o *tm*. Nesse caso, os valores para os parâmetros da primeira fase dependem muito da aplicação, logo devem ser analisados cuidadosamente. Recomenda-se usar o maior número de nós possível (*nn*) para tentar paralelizar ao máximo as operações.

3. **OLTP**: caracterizado pela maioria das consultas serem pouco complexas. Recomenda-se usar um valor baixo para o *fm* (para que todas as consultas sejam analisadas), *tm* (para otimizar tanto consultas demoradas quanto as que já estão rápidas) e *nn* (manter poucos nós para evitar custos de transferência de rede). O *cm* (quantidade de tuplas por nó) deve ser configurado de acordo com o tamanho dos dados da aplicação.

4. **Multimídia**: este tipo de sistema possui itens de dados grandes, geralmente trilhas de áudio e vídeo. Logo, sugere-se usar um valor alto para o parâmetro *nn* e baixo para o *cm* (utilizar grande número de nós com pequeno volume de dados em cada um). Também é sugerido usar um valor baixo para *fm* e *tm*, visando otimizar todas as consultas, sempre adaptando esses valores para a realidade da aplicação.

5. **GIS**: este tipo de sistema tem consultas complexas cujo desempenho é limitado pelo uso intenso de CPU (*CPU-Bound*). Portanto, recomenda-se adicionar grande número de nós (*nn*) para dividir a carga e pequeno volume de dados por nó (*cm*). Para otimizar o maior número de consultas deve-se manter um valor baixo para os parâmetros *fm* e *tm*.

Para sistemas especialistas ou fora da classificação acima, recomenda-se uma análise cuidadosa dos valores dos parâmetros pelos administradores de dados e experimentos com diversas combinações de valores antes da escolha final do esquema de alocação e da arquitetura do sistema.

A Tabela 1 contém o cruzamento entre o tipo dos sistemas e o conceito do valor dos parâmetros. Obviamente os conceitos “baixo”, “alto”, “máximo possível”e “mínimo possível”são totalmente empíricos e dependem de cada sistema especificamente. Estas suposições são feitas sobre as características conhecidas de cada tipo de sistema.

Tabela 1 – Valores recomendados para os parâmetros do algoritmo para cada tipo de aplicação

Classe sistema	**nn quant. nós**	**tm tuplas/nó**	**fm freq. mín.**	**tm tempo mín.**
Tempo real	Mín. possível	Depende aplic.	Baixo (<0.1)	Baixo
OLAP	Máx. possível	Alto	Baixo (<0.1)	Alto
OLTP	Mín.	Alto	Alto (>0.8)	Baixo
Multimídia	Máx. possível	Baixo	Alto	Baixo
GIS	Alto	Baixo	Baixo	Baixo

Relembrando:

- nn = Quantidade de nós disponíveis;
- cm = Quantidade desejada de tuplas por nó;
- fm = Frequência mínima para considerar uma consulta na análise;
- tm = Tempo mínimo para considerar uma consulta na análise.

5.5 ANÁLISE DE COMPLEXIDADE

O desempenho do algoritmo não tem grande relevância na sua utilização, visto que o mesmo é executado poucas vezes. O que importa mesmo é a qualidade do esquema de alocação de dados gerados. Apesar disso, é importante ter uma noção do tempo que o mesmo leva para executar.

Cada fase do algoritmo opera sobre dados diferentes e tem uma complexidade diferente. O tempo que cada fase consome depende da entrada e saída das anteriores.

Segundo a notação assintótica (BRUIJN, 1970), a primeira fase pode ser considerada $O(n.m)$, onde n é o número de relações e m é o tamanho médio de uma relação. O custo de execução é baseado na quantidade de operações de disco (*I/O- bound*) para ler e analisar todos os dados da base.

A segunda fase opera sobre o log de consultas, logo é $O(x.y.logz)$ onde x é a quantidade de consultas no log, y é a quantidade média de filtros e agregações nas consultas e $logz$ é o custo para verificar se cada filtro está contido na lista binária (de tamanho z) com os atributos

encontrados na primeira fase. Os recursos mais utilizado nesta fase são CPU e memória.

A terceira fase opera novamente sobre os dados. As relações selecionadas pelas fases 1 e 2 são analisadas mais profundamente para gerar a alocação. O loop mais interno executa no máximo uma vez para cada tupla da relação t. Logo, esta fase custa basicamente $O(a.b)$, onde a é a quantidade de atributos selecionados na segunda fase (iterações do loop mais externo) e b é a quantidade média de tuplas de cada relação selecionada para a contagem de valores distintos. O passo mais custoso dessa fase é a instrução "conte a quantidade de cada valor distinto", que exige uma varredura completa da relação em questão.

Portanto, sendo $f(x)$ o custo para executar o algoritmo, logo $f(x) = \theta(n.m + x.y.logz + a.b)$

Os SGBDs modernos mantém muitas das informações necessárias pelo algoritmo pré-calculadas e guardadas em tabelas de sistema. Logo, grande parte do esforço necessário pelo algoritmo pode ser evitado consultando tais tabelas.

5.6 CONSIDERAÇÕES FINAIS

Este capítulo apresentou o algoritmo genérico de alocação de dados distribuídos que analisa dados e o perfil de uso de um BD hipotético com o fim de identificar os possíveis pontos-chave para a fragmentação, neste caso atributos para formar os predicados.

O algoritmo inicialmente lista os atributos mais indicados para serem usados como predicados de fragmentação horizontal. A Fase 3 sugere um esquema de alocação simples baseada nestes atributos. Obviamente, o administrador pode modificar este esquema, melhorando-o com o conhecimento das regras de negócio de cada sistema.

No próximo capítulo é descrita a implementação do algoritmo e sua aplicação em um caso real, onde é possível notar claramente as oportunidades que as regras de negócio de cada sistema oferecem para a otimização da topologia dos sistemas distribuídos.

6 IMPLEMENTAÇÃO E AVALIAÇÃO

O presente capítulo avalia o algoritmo apresentado qualitativa (Seção 6.3) e quantitativamente (Seção 6.4), descreve a implementação e sua utilização em um caso real: o BD do Intermine.

6.1 IMPLEMENTAÇÃO DO ALGORITMO PARA O POSTGRESQL

Conforme comentado no Capítulo 5, o algoritmo é genérico e pode ser implementado em qualquer sistema de BD estruturado. Para que fosse possível realizar os experimentos com o Intermine, o algoritmo foi implementado em plpgSQL (linguagem de programação do PostgreSQL).

A primeira fase foi implementada em uma função que analisa todas as relações do BD e armazena os dados necessários em uma relação chamada *histograma*. Notou-se que a execução do algoritmo pode ser otimizada consultando apenas as estatísticas nas tabelas internas de cada SGBD. Deve-se consultar a documentação de cada SGBD para obter os locais onde a cardinalidade de cada atributo é armazenada. Com esta informação em mãos, é possível identificar os atributos mais adequados para fragmentação com algumas consultas nas tabelas de estatísticas. No caso do PostgreSQL, a tabela *pg_stats* contém todas as informações necessárias para resolver a fase 1 com apenas uma consulta SQL.

A segunda fase analisa os dados contidos em uma tabela especial com os logs das consultas. Há instruções mostrando como carregar o log do PostgreSQL nessa tabela. Cada consulta é analisada usando o comando "EXPLAIN". Cada filtro e agregação é analisado e cruzado com os resultados da primeira fase.

Com o resultado completo da segunda fase, a implementação inclui dois scripts que geram os comandos para fragmentação usando pgGrid e a criação das partições para PostgreSQL (terceira fase). Os scripts verificam as ocorrências de cada valor dos atributos e tentam gerar fragmentos com tamanhos parecidos.

Para facilitar o trabalho dos usuários, foi elaborado um guia com 5 passos simples:

1. Instalação dos objetos (funções e tabelas para a análise);

2. Carga do log de consultas para uma tabela;

3. Fase 1: Geração do histograma. Uma implementação menos custosa, usando as tabelas internas de estatísticas do PostgreSQL é disponibilizada para facilitar a análise;

4. Fase 2: Cruzamento dos predicados das consultas com o resultado da primeira fase;

5. Fase 3: Geração dos scripts para implementar a fragmentação usando o pgGrid ou particionamento de tabelas do PostgreSQL.

Os arquivos da implementação estão disponíveis para baixar em "https://svn.inf.ufsc.br/gustavotonini/dissertacao/src".

6.2 APRIMORAMENTOS NO PGGRID

Com o intuito de melhorar a ferramenta pgGrid, o otimizador de consultas foi alterado e preparado para executar o algoritmo de junções distribuídas baseado em semi junções descrito na Seção 2.3.5.1.

Nenhuma alteração no planejador de consultas foi realizada no sentido de escolher o melhor nó para executar as junções ou para executar processamento em paralelo de junções.

Quando uma consulta é submetida ao pgGrid, a mesma é decomposta e enviada para os nós que possuem os dados necessários, de acordo com as regras estabelecidas pelos comandos de configuração (*CREATE FRAGMENT*, *CREATE SERVER* e *PLACE*). No entanto, na versão anterior, mesmo que o SGBD soubesse de antemão que uma consulta retornaria um determinado número de tuplas (definido pela cláusula *LIMIT* da SQL/ANSI ou pela própria estrutura das relações envolvidas), o pgGrid aguardava a resposta de todos os nós. Para otimizar este tipo de situação, foi implementada uma melhoria chamada de "minimização das consultas paralelas"que consistiu em aguardar o retorno das N tuplas necessárias e, quando a quantidade é atingida, a consulta é automaticamente abortada e o resultado enviado para o usuário.

As duas melhorias permitiram ganhos no desempenho descritos no Experimento 4. O pgGrid modificado com as duas alterações descritas foi chamado de pgGrid 0.2, cujo código-fonte está disponível no site da ferramenta.

6.3 ANÁLISE QUALITATIVA

Antes de aplicar o algoritmo em um estudo de caso que analisa o desempenho da alocação gerada, achou-se conveniente analisar as características de cada um dos algoritmos citados neste trabalho.

As características foram selecionadas com base nos requisitos mais relevantes para os DBAs e as estratégias disponíveis em projetos de BDDs (ÖZSU; VALDURIEZ, 2011, p. 108):

- Recursos otimizados;
- Desempenho (promessas e garantias oferecidas);
- Operações beneficiadas;
- Parâmetros analisados.

Quanto a otimização de recursos foram analisados:

- CPU: Verifica-se se a aplicação do esquema de alocação pretende dividir a carga de trabalho entre os nós, visando diminuir os gargalos de CPU;
- Memória: avalia se o algoritmo tenta dividir a quantidade de memória alocada entre os nós do sistema, obtendo escalabilidade;
- Rede: verifica se o algoritmo considera diminuir as operações de rede, evitando junções distribuídas e perda de tempo com trânsito de fragmentos;
- Disco: verifica se o algoritmo divide a carga de operações de entrada/saída em disco entre os nós.

Quanto ao desempenho dos algoritmos, foram avaliados/analisados:

- Disponibilidade: o algoritmo aumenta a disponibilidade do sistema, replicando alguns dados para evitar que a queda de um nó comprometa o acesso a determinados dados;
- Tempo: o algoritmo se compromete em otimizar o tempo que um determinado conjunto de operações demanda;
- Localidade: o algoritmo garante que os dados ficarão armazenados no conjunto de nós onde serão utilizados;

- Integridade: a solução garante que haverá o cumprimento de todas as restrições de integridade e que não haverão perda ou falhas nas informações armazenadas;

- Escalabilidade: a solução garante que o crescimento da base não implicará em perdas no desempenho, se os recursos alocados crescerem na mesma proporção.

Quanto aos parâmetros analisados pelo algoritmo para estabelecer os critérios de alocação:

- Perfil de Uso: o algoritmo analisa o registro (*log*) de utilização do BD para traçar o perfil de uso, considerando estas informações para gerar o esquema de alocação;

- Esquema de Dados: a taxa de crescimento das tabelas, seus relacionamentos, estrutura e domínio dos dados são considerados na análise;

- Localidade: os perfis de localidade (onde cada dado é utilizado) são analisados pelo algoritmo.

Quanto as operações que obtém benefícios pela alocação gerada pelo algoritmo, foram consideradas:

- Seleção

- Agregação

- Ordenação

- Junção

O algoritmo apresentado neste trabalho foi comparado com as soluções apresentadas no Capítulo 4, segundo as características citadas nos parágrafos anteriores (Tabela 2).

Tabela 2 – Aderência dos algoritmos às características avaliadas

Caract.	Elemento	1	2	3	4	5	6	7	8	9
Recursos	CPU	X	X	X		X	X		X	X
	Memória	X	X			X	X	X		X
	Rede	X				X	X	X	X	
	Disco	X	X	X	X	X	X		X	X
Promessas	Disponibilidade							X	X	X
	Tempo (SLA)	X	X	X	X	X	X			X
	Integridade							X		
	Localidade				X			X		
	Escalabilidade	X	X	X				X	X	X
Operações beneficiadas	Seleção	X	X	X	X	X	X	X	X	X
	Agregação	X	X	X	X	X	X		X	
	Junção	X		X						X
	Ordenação	X		X	X	X	X			
Parâmetros	Perfil de Uso	X		X	X	X	X	X	X	
	Esquema de Dados	X			X			X	X	X
	Localidade							X		

1. O algoritmo apresentado neste trabalho;
2. HadoopDB (ABOUZEID et al., 2009);
3. O algoritmo apresentado por Bellatreche, Cuzzocrea e Benkrid (2010);
4. O algoritmo apresentado por Abdalla e Amer (2012);
5. O algoritmo apresentado por Adl e Rouhani (2009);
6. O algoritmo apresentado por Ahmad et al. (2002);
7. MySQL Cluster (Oracle Corporation, 2004);
8. O algoritmo apresentado por Daudpota (1998);
9. OpenMolGrid (DUBITZKY et al., 2004).

Pode-se notar que os algoritmos visam essencialmente melhorar o desempenho dos sistemas, deixando de lado quesitos importantes como disponibilidade e integridade dos dados. Os recursos que recebem mais

atenção são CPU e disco. As operações mais focadas são seleção e agregação.

Os algoritmos utilizados nesta comparação visam melhorar o desempenho das consultas mais comuns. No entanto, as soluções genéricas pecam por desconhecer as regras de negócio envolvidas na aplicação e os padrões de crescimento das bases de dados. Adicionando estes fatores no algoritmo, é possível oferecer uma solução mais otimizada em termos de desempenho e utilização de recursos.

Pode-se notar que o algoritmo aqui proposto se diferencia dos demais pela quantidade de operações afetadas e por analisar o esquema de dados, além do perfil de uso. É importante ressaltar novamente que o foco deste trabalho são aplicações biológicas, que possuem requisitos diferentes das demais aplicações (MUSHEGIAN, 2011). Isso não impede que outros tipos de BD façam uso do algoritmo, desde que sejam consideradas as premissas definidas na Seção 5.1.

As próximas seções mostram como os requisitos deste tipo de aplicação são endereçados pelo algoritmo e os efeitos causados pela sua aplicação.

6.4 ANÁLISE QUANTITATIVA - ESTUDO DE CASO

Esta seção descreve a execução do algoritmo sobre o BD do Intermine e a avaliação do impacto da fragmentação gerada no desempenho das consultas mais comuns.

6.4.1 Execução do algoritmo sobre a base do Intermine

Depois de implementado para o SGBD PostgreSQL, o algoritmo foi testado usando o BD do modMine (CONTRINO et al., 2012), um subprojeto do Intermine. A base utilizada é a versão 32 do modMine, disponível para baixar em http://www.flymine.org/download/modmine/r32/modmine-r32.

O algoritmo foi executado seguindo as instruções descritas na Seção 6.1.

Com ajuda da equipe que administra o modMine, o log de consultas referente ao período entre 28/11/2012 e 20/02/2013 foi coletado e utilizado para executar a fase 2 do algoritmo. O arquivo foi disponibilizado em http://www.flymine.org/download/modmine/misc/xgustavo/ e replicado para fins de preservação em http://www.inf.ufsc.br/ gusta-

votonini/logsmodmine/.

Os valores de parâmetros utilizados foram os seguintes:

- *nn*: 8 (número de nós disponíveis para os testes);
- *cm*: 2 milhões de registros (para garantir um bom tamanho de cada fragmento);
- *fm*: 0.1 (segue as recomendações de baixas frequências para sistemas tipo OLAP);
- *tm*: 6 segundos (s) (otimizar apenas as consultas mais lentas).

Depois de executar o algoritmo com os parâmetros acima, foram selecionados os atributos listados na Tabela 3. O Anexo A contém uma representação do esquema de dados do Intermine baseada nas relações de herança das classes da aplicação.

Tabela 3 – Saída da fase 2 do algoritmo executado sobre a base de dados do modMine (versão 32)

Relação	**Atributo**
bioentity	organismid
bioentity	primaryidentifier
featuressubmissions	features
featuressubmissions	submissions
geneflankingregion	organismid
location	chromosomeid
location	featureid
location	locatedonid
sequencefeature	chromosomeid

6.4.2 Experimentos

Nesta seção são descritos os experimentos realizados sobre a base do modMine fragmentada de acordo com a aplicação do algoritmo.

Foram executados quatro experimentos utilizando o mesmo ambiente computacional, mas em cenários e com roteiros distintos. No

primeiro experimento, foram executadas seis consultas específicas, escolhidas em conjunto com os desenvolvedores do Intermine, visando medir os ganhos em tempo de execução. Os outros três experimentos foram executados usando o mesmo log de consultas utilizado para fragmentar a base. Dessa forma, pode-se afirmar que os testes simulam exatamente a demanda do sistema atualmente em produção.

As próximas seções descrevem o ambiente e cada um dos cenários analisados.

6.4.2.1 Ambiente Computacional

Os experimentos foram executados em um servidor Dell PowerEdge R710, com dois processadores Intel Xeon X5690 de seis núcleos e 32GB de RAM. Máquinas virtuais foram criadas usando o hipervisor VMWare ESXi 5.1. No período dos testes, a taxa de transferência dos discos estava em 102MB/s com taxa de I/O de 917 operações de entrada/saída por segundo (IOPS).

Como o Intermine é baseado em PostgreSQL, foram testadas diversas configurações deste SGBD, que foram chamadas de plataformas de teste, utilizando a alocação proposta pelo algoritmo em diversos cenários. As configurações foram:

- PostgreSQL padrão sem extensões ou modificações nos parâmetros; sem a aplicação do algoritmo para fragmentação/particionamento;
- PostgreSQL usando o mecanismo de particionamento;
- PostgreSQL usando índices parciais;
- BDD usando o pgGrid;
- BDD usando o HadoopDB.

Algumas plataformas foram testadas com diferente número de partições/fragmentos e com quantidades diferentes de recursos (núcleos de CPU e memória), pois é necessário avaliar o comportamento das plataformas com a adição de mais recursos. Todos os programas foram executadas com as configurações padrão sem nenhum tipo de otimização.

Para simplificar os cenários dos testes, a base de dados do modMine foi analisada e foi escolhido apenas um atributo de duas relações

selecionadas pelo algoritmo (Tabela 3) para fragmentação e particionamento: o atributo *chromosomeid* das relações *location* e *sequencefeature*, pois notou-se que estas são as maiores em tamanho (bytes) e são utilizadas na maior parte das consultas mais custosas (tempo).

A Figura 10 mostra a estrutura das duas relações e o relacionamento entre as mesmas.

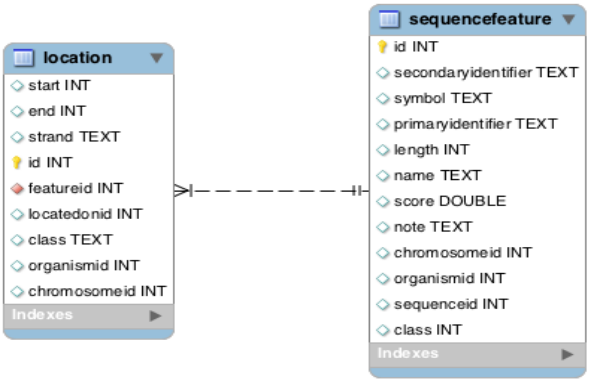


Figura 10 – Relações usadas nos experimentos. O atributo *chromosomeid* foi selecionado para a fragmentação e particionamento

A relação *location* possui cerca de 40 milhões de tuplas e ocupa 6 GB em disco. A *sequencefeature* possui 38 milhões de tuplas e 10 GB em disco.

A relação *location* original não possuia o atributo *chromosomeid*. Para fins de otimização, foi solicitado à equipe do Intermine que adicionasse este atributo para que fosse possível estabelecer a fragmentação/particionamento nesta relação. Isso não causa perdas significativas no esquema de dados original, considerando-se apenas uma "desnormalização" comum em DW's. Os valores para o novo atributo foram replicados a partir da relação *sequencefeature*.

As duas relações selecionadas foram fragmentadas no pgGrid e particionadas nos ambiente com particionamento e índices parciais em configurações com 4 e 8 partições/fragmentos, de acordo com as regras da Tabela 4. As demais relações do modelo de dados foram mantidas intactas nos ambientes centralizados e replicadas nas plataformas distribuídas. Por isso, o uso de espaço em disco nas plataformas com dados replicados (pgGrid e HadoopDB) praticamente dobrou.

Tabela 4 – Regras (predicados) utilizados para particionar/fragmentar o BD do Intermine

	Ambientes com 4 nós/partições	**Ambientes com 8 nós/partições**	**Predicado** (valores do *chromosomeid*)
Nó/ Partição	1	1	1000035
		2	11000006
	2	3	11000008
		4	11000004, 1000022
	3	5	1000015
		6	11000003, 11000005
	4	7	1000521
		8	Outros valores

Os conjuntos de scripts utilizados para estabelecer a fragmentação no pgGrid, para criar o ambiente com particionamento e índices parciais foram disponibilizados no endereço "https://svn.inf.ufsc.br/gustavotonini/dissertacao/src/experimentos".

As próximas seções descrevem os experimentos realizados e os cenários executados sobre as plataformas descritas aqui. É importante ressaltar que foram feitas comparações apenas entre plataformas com a mesma quantidade de recursos.

6.4.2.2 Experimento 1

No experimento inicial, buscou-se executar consultas específicas usando diversas configurações do SGBD para avaliar a viabilidade de cada configuração para os demais experimentos.

A Tabela 5 mostra as diferentes combinações (memória X *núcleos de CPU* X tipo de ambiente) das plataformas onde as consultas do primeiro experimento foram executadas.

A Figura 11 mostra um resumo da configuração do *hardware* físico e virtual utilizados no experimento. Nota-se que os mesmos recursos da máquina centralizada foram usados nos clusters pgGrid e HadoopDB e que as relações *location* e *sequencefeature* foram fragmentadas horizontalmente. O uso de espaço em disco é maior no pgGrid devido a replicação completa das relações que não foram fragmentadas em todos os nós.

Tabela 5 – Plataformas utilizadas no Experimento 1

ID	SGBD	Topologia	Recursos dos nós		Disco (GB)
			Núcleos	RAM	
C1	Centralizado	1 nó	8	8GB	471
P1	Particionamento	1 nó, 8 partições	8	8GB	525
I1	Índices parciais	1 nó, 8 partições	8	8GB	496
G1	pgGrid	8 nós	1	1GB	848
H1	HadoopDB	8 nós	1	1GB	859

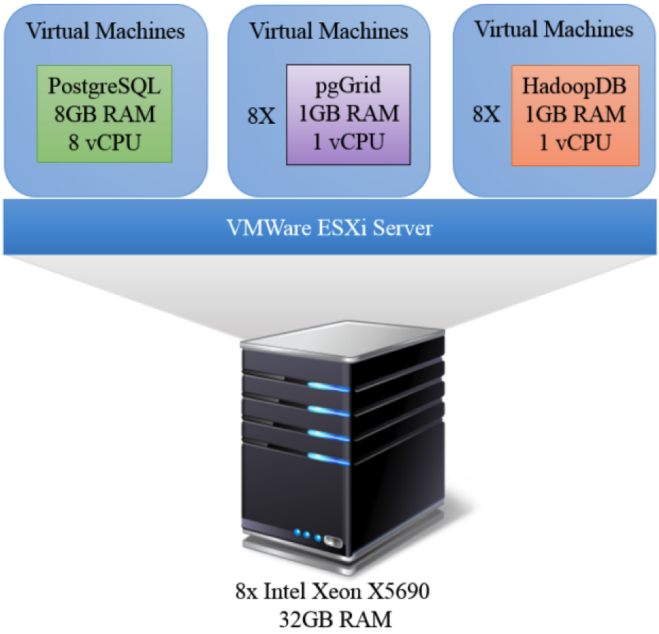


Figura 11 – *Hardware* físico e virtual usado no Experimento 1

Para o primeiro experimento foram selecionadas seis consultas problemáticas do *log* enviado pela equipe do Intermine. As consultas envolvem as duas maiores relações da base: *location* e *sequencefeature*. Cada uma das consultas selecionadas para o experimento está

detalhada nas Tabela 6. Foi necessário adicionar uma seleção (filtro em negrito na Tabela 6) por cromossomo (atributo *chromosomeid*) nas consultas para o pgGrid conseguir otimizá-las corretamente e enviá-las para apenas um nó.

Tabela 6 – Consultas selecionadas para o Experimento 1

ID	Código SQL	Descrição
1	SELECT sum(s.id) FROM sequencefeature s join location1 l1 on l1.featureid = s.id join location1 l2 on l1.locatedonid = l2.locatedonid WHERE 1002269 = l2.featureid and l1.featureid != l2.featureid and **l1.chromosomeid=1000035** and **l2.chromosomeid=1000035** and **s.chromosomeid=1000035**	Agregação de campo indexado
2	SELECT a1_.id AS a1_id FROM SequenceFeature a1_, OverlappingFeatures AS i WHERE 73448132 = I.SequenceFeature AND i.OverlappingFeatures = a1_.id	Busca sequências sobrepostas
3	select count(*) from location1 l1 join location1 l2 on (l1.locatedonid = l2.locatedonid) WHERE 73448132 = l2.featureid and l1.featureid !=l2.featureid and **l1.chromosomeid=1000035** and **l2.chromosomeid=1000035**	Verifica se uma sequência existe
4	SELECT sum(s.length) FROM sequencefeature s join location1 l1 on l1.featureid = s.id join location1 l2 on (l1.locatedonid = l2.locatedonid) WHERE 73448132 = l2.featureid and l1.featureid != l2.featureid and **l1.chromosomeid=1000035** and **l2.chromosomeid=1000035** and **s.chromosomeid=1000035**	Agregação de campo não indexado
5	select * from sequencefeature where clob like '%TCGAAA%'	Busca por uma sequência com um padrão
6	select * from sequencefeature s where **chromosomeid=1000035** and exists (select id from location l where l.featureid=s.id and **l.chromosomeid=1000035**) limit 10;	Busca por sequências com dados de localização

As consultas foram executadas em cada uma das plataformas e os tempos medidos estão na Tabela 7, juntamente com a diferença, em percentual, dos tempos de cada plataforma em relação a C1. A Figura 12 mostra o resultado de forma gráfica.

Tabela 7 – Tempos de execução do Experimento 1

ID	C1 centr.	P1 part.		I1 índ. parc.		G1 pgGrid		H1 HadoopDB	
Cons.	s.	s.	Δ%	s.	Δ%	s.	Δ%	s.	Δ%
1	392	126	-67	117	-70	74	-81	>600	N/A
2	8	118	+1375	137	+1612	3	-63	>600	N/A
3	37	5	-86	166	+348	15	-59	>600	N/A
4	139	81	-42	123	-11	65	-53	>600	N/A
5	158	80	-49	158	0	13s	-92	19	-88
6	20	1	-95	32	+60	1s	-95	>600	N/A

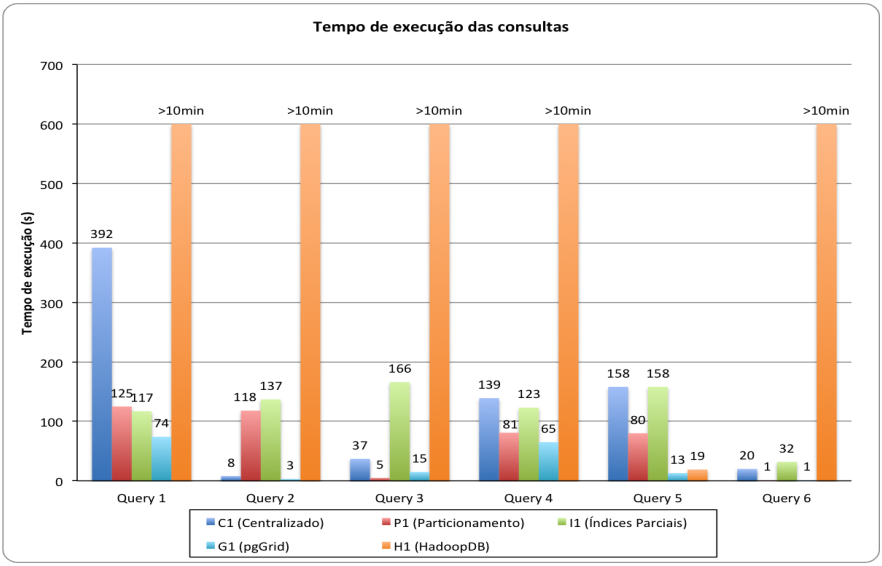


Figura 12 – Resultados do experimento 1

6.4.2.3 Análise dos resultados do Experimento 1

Os ganhos em termos de tempo de execução são notáveis, principalmente porque a diferença aparece em plataformas que utilizam exatamente os mesmos recursos físicos. Por outro lado, apenas uma consulta apresentou ganho usando a plataforma HadoopDB.

O menor ganho entre os ambientes centralizados e distribuídos foi observado nas consultas 3 e 4, onde as consultas executaram no pgGrid em menos da metade do tempo do ambiente centralizado. Os ganhos podem chegar a 20 vezes em consultas similares à consulta 6, onde a localização dos dados é especificada de acordo com o predicado de fragmentação. Isso indica que o algoritmo proposto, combinado com um SGBD Distribuído eficiente, é um método muito eficaz para otimizar BD relacionais biológicos.

É importante notar que as consultas 1, 3, 4 e 6 executam em apenas um nó nos ambientes distribuídos, devido ao fato de seus predicados casarem com as regras de fragmentação. Mesmo assim estas consultas tiveram ganhos devido ao tamanho reduzido das relações nestes nós que permitiram buscas mais rápidas do que no ambiente centralizado (oito vezes maior). Isso demonstra que o algoritmo gera ambientes escaláveis: o sistema distribuído pode aumentar em tamanho e número de nós sem afetar o desempenho das consultas.

O processamento em paralelo permitiu que as consultas 1, 4 e 5 executassem de forma mais eficiente no pgGrid. Neste caso, índices não ajudam muito (a diferença entre as consultas 1 e 4 é pequena) porque os fragmentos são divididos usando a mesma chave do índice. A consulta 5 é onde a busca em paralelo obtém o melhor resultado (G1 e H1), apresentando diminuição de tempo proporcional ao número de nós das plataformas distribuídas.

O HadoopDB apresentou desempenho pior porque é baseado na tecnologia Auto-Sharding com fragmentação baseada em *hash*. A ferramenta não sabe quais padrões de dados estão em cada nó por não ter um dicionário de dados com os predicados de fragmentação e a alocação dos fragmentos. Por isso, em cada consulta é necessário contactar todos os nós para encontrar os dados solicitados. Esse problema pode ser resolvido implementando as consultas usando a API Java do Hadoop ao invés de submetê-las em SQL. Nesse caso o usuário pode "programar"em quais nós elas serão executadas.

O maior problema do HadoopDB é que o *middleware* não sabe como a base de dados está dividida, ou seja, não conhece o esquema dos dados de cada nó. Caso as consultas sejam escritas em SQL e não

programadas em Java, a otimização é pobre, pois é o desenvolvedor que deve usufruir dos benefícios do paralelismo através da programação dos métodos Map/Reduce em Java.

6.4.2.4 Experimento 2

O log executado no Experimento 2 contém mais de trezentas mil consultas referentes a praticamente três meses de uso do ambiente modMine de produção. Com o intuito de focar a análise nas consultas que apresentam maior impacto no desempenho do sistema, foram selecionadas apenas as consultas que levam mais de 700 milissegundos para executar no ambiente centralizado (C1). A quantidade de consultas selecionadas segundo este critério foi 15343.

O HadoopDB foi excluído dos demais experimentos devido aos problemas na execução do Experimento 1. A plataforma com índices parciais (I1) também foi excluída devido ao desempenho similar a da plataforma com particionamento (P1).

No experimento 2 foram introduzidas plataformas com maior quantidade de memória RAM para avaliar o impacto, em relação ao tempo de execução, em cada configuração (centralizado, particionado e fragmentado). As novas plataformas estão descritas na Tabela 8.

Tabela 8 – Plataformas utilizadas no Experimento 2

ID	**SGBD**	**Topologia**	**Recursos dos nós**		**Disco (GB)**
			Núcleos	**RAM**	
C1	Centralizado	1 nó	8	8GB	471
C2	Centralizado	1 nó	8	16GB	471
P1	Particionamento	1 nó, 8 partições	8	8GB	525
P2	Particionamento	1 nó, 8 partições	8	16GB	525
G1	pgGrid	8 nós	1	1GB	848
G2	pgGrid	8 nós	1	2GB	848

O experimento 2 consistiu, basicamente, em executar nas plataformas C1, C2, P1, P2, G1 e G2 todas as consultas registradas no *log* com o intuito de medir o tempo total de execução do *log* em cada plataforma. Durante os testes com cada plataforma, todas as outras

máquinas virtuais foram desligadas para evitar interferências e trocas de contexto de CPU.

A Figura 13 mostra o tempo que cada plataforma levou para executar as consultas selecionadas. As barras em azul representam as plataformas do Experimento 1 (com 8GB de RAM) e as barras em vermelho as novas plataformas (com 16GB de RAM).

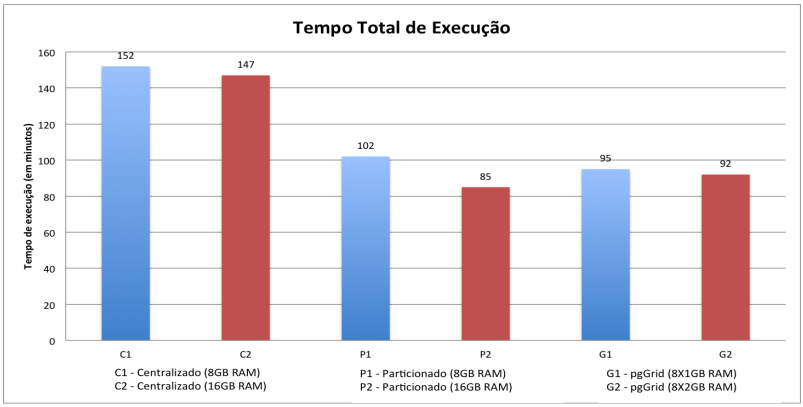


Figura 13 – Tempos de execução do Experimento 2

A próxima seção discute os resultados obtidos e analisa com mais atenção as principais consultas do log.

6.4.2.5 Análise dos resultados do Experimento 2

Novamente os resultados obtidos nas plataformas particionadas e fragmentadas usando o algoritmo são notáveis se comparados com as plataformas centralizadas. O sistema consegue executar mais consultas no cenário P2 do que em qualquer outro, incluindo C1, o atual cenário de produção.

Também é notável o efeito causado pelo aumento na quantidade de memória na plataforma particionada: P2 teve ganho de 17 % no tempo de execução do log sobre P1. A memória adicional aumenta o espaço disponível para cache de disco e diminui o tempo gasto com leituras de blocos de dados e índices. No entanto, espera-se que tal efeito não tenha boa escalabilidade, ou seja, aumentar indefinidamente a quantidade de RAM não continuará diminuindo os tempos de con-

sulta porque todos os blocos de disco estarão em cache, mas as buscas continuarão sendo sequenciais.

Para analisar melhor os resultados, a Figura 14 mostra o tempo (em milissegundos) gasto pelas cinco consultas mais demoradas (em relação a C1) do log. A Tabela 9 descreve cada uma das consultas cujos códigos SQL foram disponibilizados no Anexo B. Os tempos estão na Tabela 10. A Tabela 11 mostra a diferença, em percentual, entre C1 e as demais plataformas.

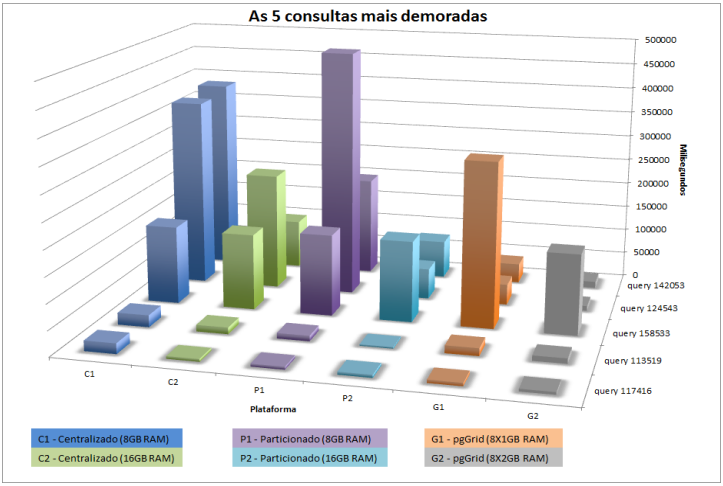


Figura 14 – Experimento 2 - Tempos gastos pelas 5 consultas mais demoradas do *log*

ID	Descrição
142053	Listar classes de sequências sobrepostas
124543	Listar cópias anormais de DNA nas submissões
158533	Procurar determinada característica de um organismo
113519	Procura por genes com tamanhos semelhantes para identificar problemas na cópia
117416	Verifica se existem íntrons [1] em uma determinada amostra

Tabela 9 – As 5 consultas mais demoradas (SQL no Anexo B)

[1] região de DNA não codificante (SNUSTAD; SIMMONS, 2010, p. 309)

Tabela 10 – Experimento 2 - Tempos para execução (em segundos) das 5 consultas mais demoradas

ID Consulta	C1 (s.)	C2 (s.)	P1 (s.)	P2 (s.)	G1 (s.)	G1 (s.)
142053	383.7	97.4	196.6	75.3	38.8	13
124543	374.1	232.5	488.3	60.8	40.9	9.7
158533	155.3	161.2	150.7	160	324	158.5
113519	23.7	10.6	13	2.8	15	10.2
117416	20.4	5.4	5.1	5.5	6	5

Tabela 11 – Experimento 2 - Diferença (em percentual) comparando C1 e as demais plataformas nas 5 consultas mais demoradas

ID Consulta	C2 Δ%	P1 Δ%	P2 Δ%	G1 Δ%	G1 Δ%
142053	-74	-49	-80	-90	-96
124543	-38	+31	-84	-89	-97
158533	+4	-3	+3	+109	-2
113519	-55	-45	-88	-36	-56
117416	-73	-75	-73	-70	-76

Relembrando:

- **C1** = Centralizado 8GB RAM;
- **C2** = Centralizado 16GB RAM;
- **P1** = Particionado 8GB RAM;
- **P2** = Particionado 16GB RAM;
- **G1** = 8 X pgGrid 1GB RAM;
- **G2** = 8 X pgGrid 2GB RAM.

É fácil notar que, em geral, as consultas executam entre 30 e 60 por cento mais rapidamente nos ambientes particionados/fragmentados de acordo com o algoritmo do que nos centralizados. Algumas exceções em G1 foram causadas pela quantidade reduzida de RAM para *cache* em cada nó do pgGrid. Isso forçou o sistema operacional (SO) a liberar memória *cache*, forçando o aumento das operações de disco.

Por outro lado, foram selecionadas as cinco consultas mais frequentes do log (Figura 15). A Tabela 12 descreve cada uma delas. Os tempos da figura representam a média das execuções de cada consulta em cada plataforma. Pode-se observar as melhorias obtidas pelas plataformas pgGrid e particionadas. A adição de mais RAM aos sistemas causam melhorias no tempo de resposta, mas a memória é melhor utilizada nos ambientes particionados/fragmentados. É fácil encontrar evidências desse comportamento analisando os contadores de falhas de cache no SO.

Cada melhoria pontual obtida é fundamental quando se executa algoritmos de detecção de padrões, muito comuns em BD biológicos. Neste casos, as melhorias no tempo de execução podem se tornar exponenciais, visto que muitas buscas são recursivas. Neste experimento, as consultas 124963, 53389 e 136931 representam cerca de 30% do total de tempo gasto com as consultas selecionadas para a análise. Por isso, as melhorias obtidas nas plataformas pgGrid e particionadas podem reduzir dramaticamente o tempo e recursos consumidos, logo mais consultas podem ser executadas em um determinado período de tempo, aumentando a capacidade do sistema.

Tabela 12 – As 5 consultas mais frequentes (SQL no Anexo B)

ID	**Descrição**	**Ocorrências no log**
124963	Listar sequências sobrepostas	145
53389	Verificar a localização de um determinado gene, sequência, íntron, éxon, etc.	112
136931	Verificar se as amostras de um organismo tem regiões codificantes	105
57018	Listar todos os genes e suas quantidades presentes nas amostras de um organismo	77
247582	Busca por um objeto persistente do Intermine	56

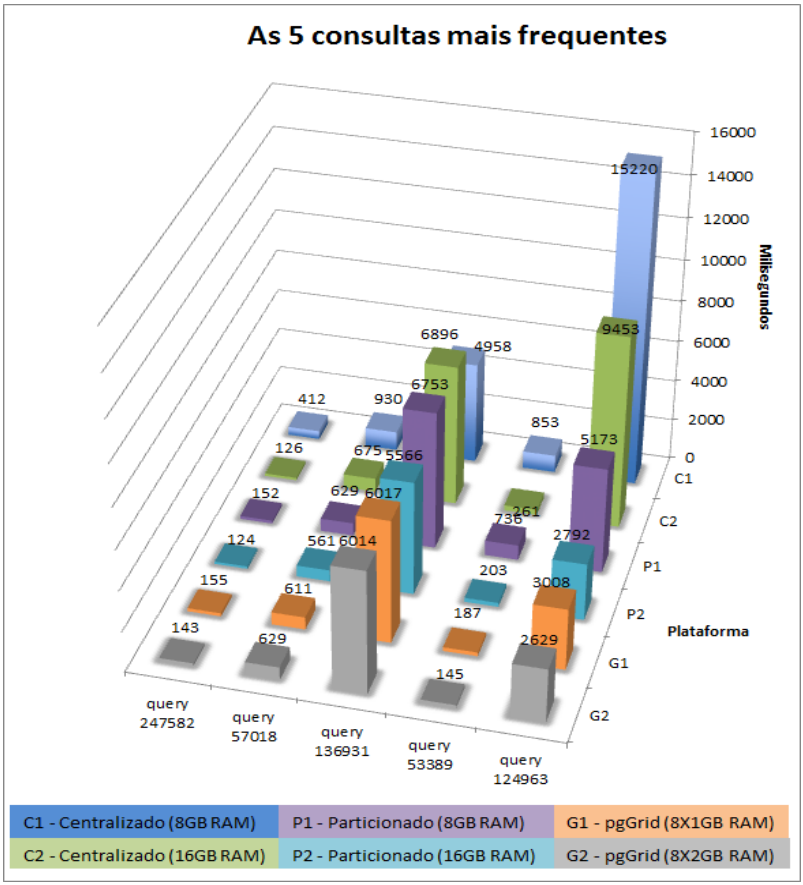


Figura 15 – Experimento 2 - Tempos gastos pelas 5 consultas mais frequentes do log

Também pode-se notar que algumas consultas como 136931 e 158533 não mostraram melhorias com os novos ambientes apresentados. Estas consultas não utilizam as relações particionadas. Por isso, o desempenho em todas as plataformas ficou parecido com o desempenho da plataforma inicial (C1).

Analisando as consultas mais frequentes, é possível ter mais evidências que particionar grandes tabelas em partições menores faz com que as consultas apresentem melhor desempenho. É importante lembrar que as melhorias em tempo de execução nas consultas da Fi-

gura 15 são multiplicadas pela quantidade de ocorrências de cada uma.

Os resultados globais mostrados na Figura 13 estão mais relacionados com as consultas mais frequentes (Figura 15) do que as mais longas (Figura 14), pois as cinco consultas mais frequentes representam cerca de 50% do tempo gasto para executar o log completo. Isso indica que as plataformas com pgGrid e particionamento são mais escaláveis e indicadas para sistemas com uso intenso de determinadas consultas.

A Figura 15 mostra que o tempo de quase todas as consultas mais frequentes no log diminuiu (drasticamente em alguns casos como a 124963 e a 53389). Estes resultados estão relacionados a buscas sequenciais em tabelas particionadas. Como o espaço de busca é cerca de oito vezes menor nos ambientes com pgGrid e particionamento, algumas consultas chegam a executar seis vezes mais rápido do que na versão centralizada.

Particionamento é uma boa opção para DBAs que querem evitar grandes mudanças na arquitetura. Particionar uma relação é uma operação simples na maioria dos SGBDs atualmente em produção.

6.4.2.6 Experimento 3

No experimento 3 foram introduzidas plataformas com menos partições e nós para verificar o impacto sobre os tempos de execução do *log*. As novas plataformas estão descritas na Tabela 13.

Tabela 13 – Plataformas utilizadas no Experimento 3

ID	**SGBD**	**Topologia**	**Recursos dos nós**		**Disco**
			Núcleos	**RAM**	**(GB)**
C2	Centralizado	1 nó	8	16GB	471
P2	Particionamento	1 nó, 8 partições	8	16GB	525
P3	Particionamento	1 nó, 4 partições	8	16GB	521
G2	pgGrid	8 nós	1	2GB	848
G3	pgGrid	4 nós	2	4GB	662

O experimento 3 consistiu, basicamente, em executar o log completo nas novas plataformas e comparar o resultado com as versões com mais nós/partições do Experimento 2. A Figura 16 mostra o tempo que cada plataforma levou para executar as consultas selecionadas.

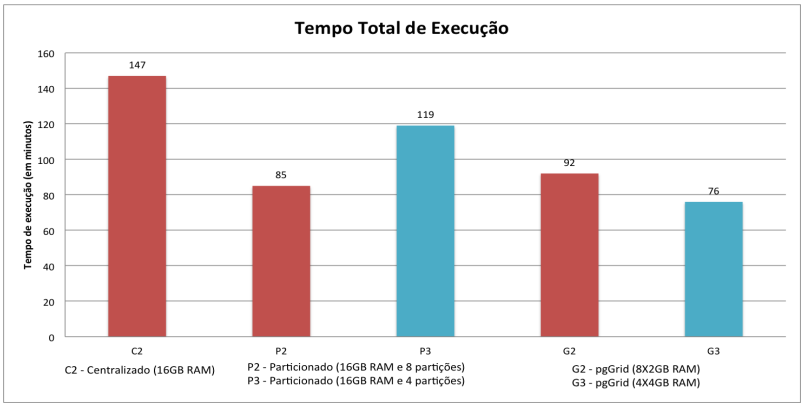


Figura 16 – Tempos de execução do Experimento 3

O maior número de partições em P2 resultou em tempos melhores. O pgGrid não seguiu a mesma lógica: a configuração com menos partições e mais RAM por nó (G3) foi melhor que G2. Este comportamento evidencia o impacto causado pela utilização de memória RAM para *cache* de blocos de disco.

No pgGrid, os administradores podem especificar quais padrões de dados serão alocados em cada máquina (com a quantidade de memória também definida pelos DBAs). Portanto, é possível alocar, de forma indireta, uma quantidade específica de memória para cada fragmento, permitindo mais flexibilidade que o mecanismo de particionamento do PostgreSQL. Os benefícios disso podem ser visualizados comparando os resultados de P3 e G3.

A relação entre quantidade de memória e número de partições deve ser cuidadosamente estudada antes de estabelecer a alocação de um sistema distribuído. Testes com diferentes cenários devem ser executados para garantir que a melhor combinação foi escolhida. Por esse motivo, é útil automatizar alguns testes aqui apresentados. Isso ajudaria os administradores a avaliar as melhores combinações de cenários.

Novamente foram analisadas as consultas mais demoradas e frequentes para entender melhor os resultados gerais. As Figuras 17 e 18 contém os gráficos onde é possível verificar que cada consulta em G3 obteve benefícios da maior quantidade de RAM por nó. G3 foi a configuração com a melhor relação entre volume de dados e memória por nó.

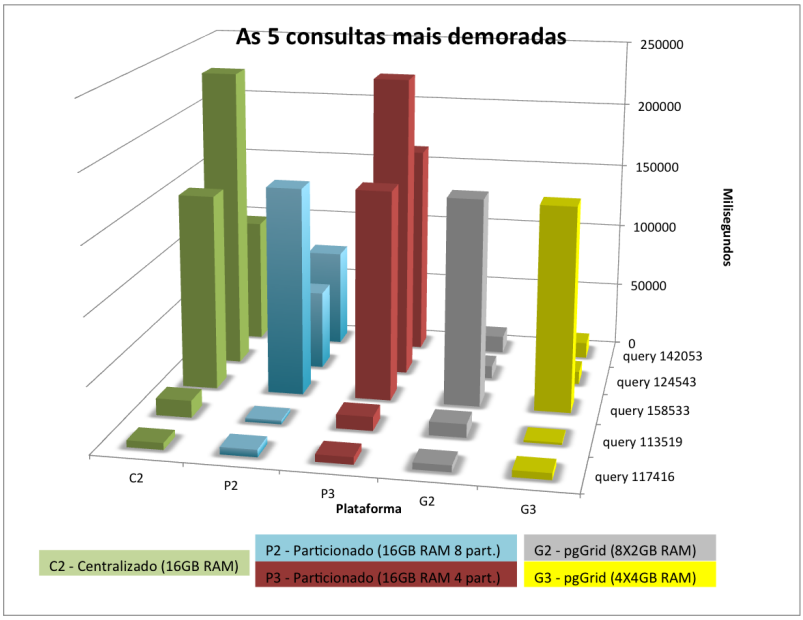


Figura 17 – Experimento 3 - Tempos gastos pelas 5 consultas mais demoradas do log

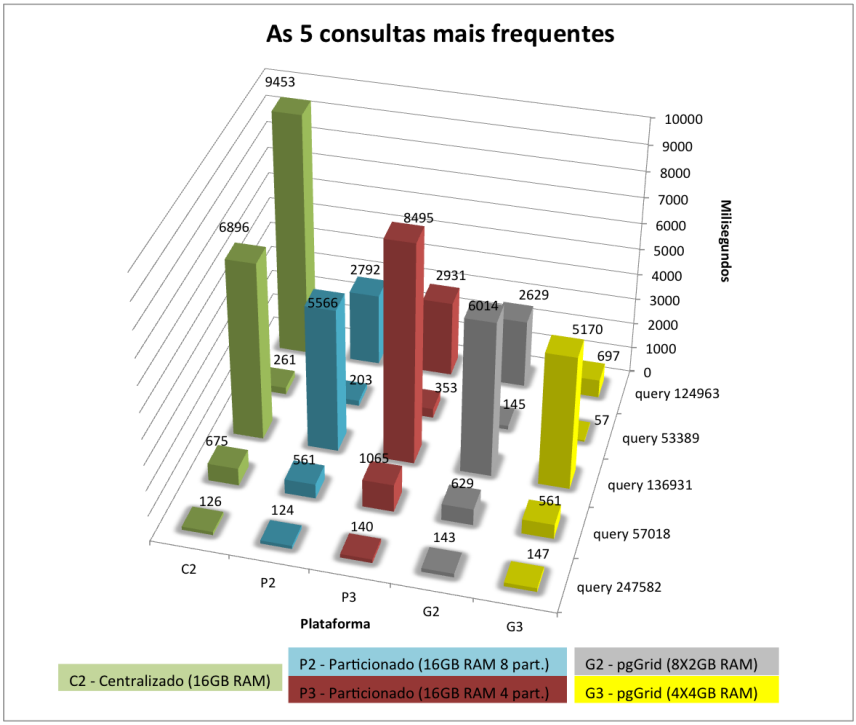


Figura 18 – Experimento 3 - Tempos gastos pelas 5 consultas mais frequentes do log

Notou-se, ao executar o experimento 1, que a quantidade maior de memória nos ambientes centralizados (C1, P1 e I1) fez com que o SO utilizasse essa memória para *cache* de disco. Então, se uma segunda consulta envolvendo os mesmos dados da primeira for executada, os tempos de resposta serão drasticamente reduzidos. Por isso, a primeira execução de cada consulta foi desconsiderada e os resultados correspondem à segunda execução. Tal comportamento não se aplica aos demais experimentos porque as consultas foram executadas na ordem em que aparecem no log.

Uma vez que o padrão de distribuição de dados e o perfil de consultas podem mudar, o algoritmo deve ser executado de tempos em tempos para manter a topologia atualizada e evitar degradações no desempenho.

Os experimentos podem ser reproduzidos em outros SGBDs para avaliar os ganhos do algoritmo. Os parâmetros de entrada devem ser ajustados de acordo com a aplicação testada.

6.4.2.7 Experimento 4

O experimento 4 consistiu, basicamente, em executar o log completo no pgGrid 0.2 (com as melhorias descritas na Seção 6.2), chamada de plataforma G4, e comparar o resultado com a versão equivalente do Experimento 2: G1. Dessa forma, o impacto das alterações pode ser avaliado sobre o BD do modMine fragmentado de acordo com o algoritmo proposto.

A Figura 19 mostra o tempo que cada plataforma levou para executar as consultas selecionadas.

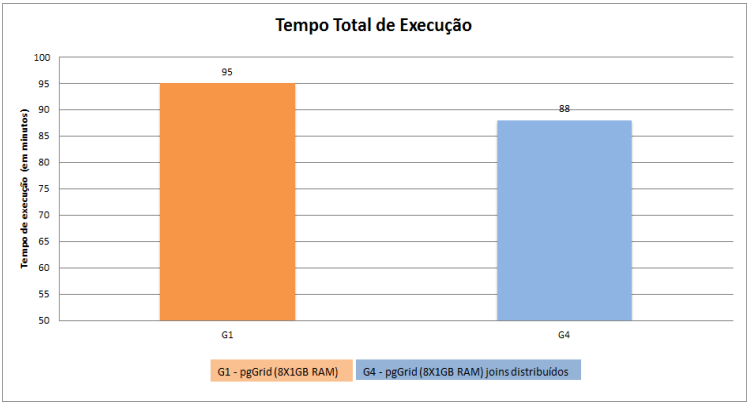


Figura 19 – Tempos de execução do Experimento 4

Não foram observadas melhorias drásticas no desempenho da nova versão do pgGrid porque a forma com que as relações foram fragmentadas evita as junções distribuídas. O algoritmo de alocação trata essa questão na Fase 2, de acordo com a premissa 1.

Mesmo assim, as principais consultas foram novamente analisadas para verificar em quais casos houve alteração no tempo de execução. Os gráficos de tempo de execução estão nas Figuras 20 e 21.

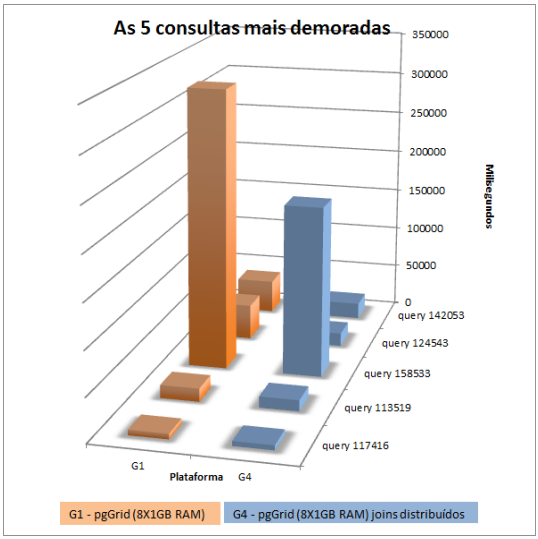


Figura 20 – Experimento 4 - Tempos gastos pelas 5 consultas mais demoradas do log

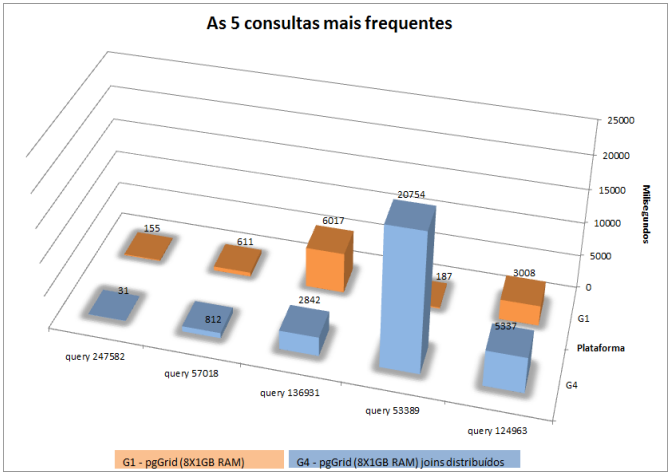


Figura 21 – Experimento 4 - Tempos gastos pelas 5 consultas mais frequentes do log

Algumas consultas, principalmente as que retornam mais tuplas, apresentaram melhorias com o algoritmo de junções distribuídas devido ao menor tráfego de rede. Porém, outras consultas, principalmente as que retornam poucas tuplas, pioraram devido às mensagens adicionais introduzidas pelo algoritmo de junções de quatro fases.

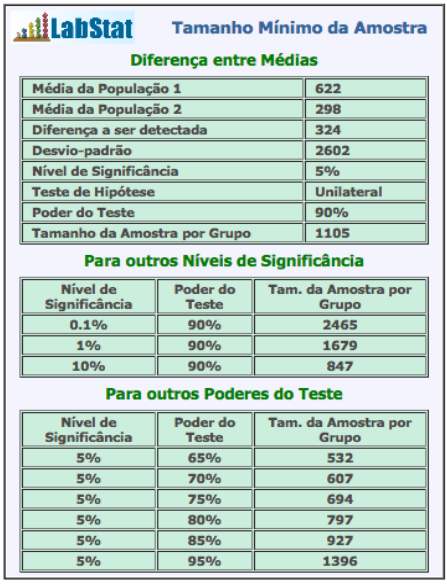
6.5 VALIDAÇÃO DOS RESULTADOS

Com o fim de validar, com rigor científico, a amostra selecionada, foi utilizado o teste de hipóteses t de Student (GOSSET et al., 1990). O cálculo do tamanho da amostra necessária foi realizado com auxílio do software Sestat.net (NASSAR et al., 2005) efetuando o cálculo para diferença entre médias de variáveis qualitativas e quantitativas (teste QLQT) com base na média e desvio padrão referentes ao tempo de execução das consultas selecionadas. A Figura 22 mostra a simulação realizada no software.

Teoricamente seria necessário calcular o tamanho da amostra para todas as combinações de cenários. No entanto, de posse dos dados finais do experimento, verificou-se que a maior diferença nos tempos de execução ocorreu entre as plataformas C1 e G3. Portanto, foi calculado o tamanho da amostra apenas para este caso e os demais exigiriam, teoricamente, tamanhos de amostras menores.

A Hipótese H1 do teste é: existe diferença de tempo executando as mesmas consultas nas plataformas C1 e G3?

A média dos tempos de execução de cada consulta na plataforma C1 é 622 milissegundos. A média dos tempos de execução de cada consulta na plataforma G3 é 298 milissegundos. O desvio padrão máximo (C1, G3 e C1-G3) é de 2602 milissegundos.



LabStat — Tamanho Mínimo da Amostra

Diferença entre Médias

Média da População 1	622
Média da População 2	298
Diferença a ser detectada	324
Desvio-padrão	2602
Nível de Significância	5%
Teste de Hipótese	Unilateral
Poder do Teste	90%
Tamanho da Amostra por Grupo	1105

Para outros Níveis de Significância

Nível de Significância	Poder do Teste	Tam. da Amostra por Grupo
0.1%	90%	2465
1%	90%	1679
10%	90%	847

Para outros Poderes do Teste

Nível de Significância	Poder do Teste	Tam. da Amostra por Grupo
5%	65%	532
5%	70%	607
5%	75%	694
5%	80%	797
5%	85%	927
5%	95%	1396

Figura 22 – Cálculo do tamanho da amostra para os experimento 2, 3 e 4

O tamanho da amostra necessário para provar a hipótese H1 (O tempo de execução de uma determinada plataforma é melhor ou pior que o tempo de outra) é 2465 com nível de significância de 0.1%. Logo, as 15343 consultas selecionadas (cujo tempo $>$ 700 ms.) para os experimentos 2, 3 e 4 são suficientes para provar a hipótese.

7 CONCLUSÕES

O presente trabalho apresentou uma proposta de emprego de BDDs visando solucionar os problemas de desempenho do sistema DW Intermine, seus subprojetos e outros sistemas de BD com características semelhantes.

Para auxiliar os administradores de tais sistemas de BD, foi proposto um algoritmo que analisa o esquema de dados e o perfil de uso e gera a alocação dos dados para um ambiente de BDD definido pelo usuário através dos parâmetros.

O algoritmo foi baseado em algumas premissas, apresentadas na Seção 5.1, sobre os padrões de dados observados no Intermine, mas que podem ser generalizados para qualquer tipo de sistema.

Para avaliar o algoritmo, foi escolhido o modMine (instância do sistema DW Intermine). Quatro experimentos foram conduzidos, um com consultas específicas e os demais com o log completo da aplicação, usando diversas topologias de BD. A diminuição no tempo de execução (cerca de 50% no cenário Experimento 3) usando as plataformas que usam a fragmentação gerada pelo algoritmo sobre a plataforma atual de produção é notável e prova as hipóteses consideradas como base.

As modificações sugeridas especificamente para o sistema DW Intermine foram submetidas para avaliação da equipe que administra o sistema, visando a evolução da plataforma para um ambiente de nuvem computacional.

7.1 PUBLICAÇÕES

Este trabalho será publicado em três partes:

- Revisão bibliográfica e problema de pesquisa: *Electronic Journal of Biotechnology* (EJB). Extrato Qualis B3. (TONINI; SIQUEIRA, 2013b).

- Algoritmo e Experimento 1: *2013 IEEE 16th International Conference on Computer and Information Technology* (CIT). Extrato Qualis B1. (TONINI; SIQUEIRA, 2013a).

- Os Experimento 2, 3 e 4, com resultados mais abrangentes, serão publicados posteriormente.

7.2 LIMITAÇÕES E TRABALHOS FUTUROS

A maior limitação identificada foi que o algoritmo gera fragmentos baseados em predicados com apenas um atributo. Combinar atributos pode gerar predicados mais precisos criando fragmentos/índices menores, favorecendo mais ainda as consultas e suas condições de filtragem/agregação/ordenação. Essa função seria de grande relevância para a versão 2.0 do algoritmo. O algoritmo de Daudpota (1998) já possui esta funcionalidade e poderia ser tomado como base.

A saída do algoritmo pode ser ordenada segundo a relevância de cada atributo selecionado. Dessa forma, os primeiros atributos da lista podem ser priorizados na fragmentação da base. A versão 2.0 do algoritmo pode incluir esta funcionalidade.

Fragmentar a base segundo o algoritmo apresentado pode trazer um aumento na quantidade de recursos necessários, principalmente espaço em disco, visto que alguns fragmentos precisam ser replicados para manter os dados necessários para as consultas no mesmo nó. Este problema pode ser resolvido se o algoritmo analisar as consultas e fragmentar verticalmente algumas relações, de acordo com a necessidade, visando diminuir o espaço ocupado em cada nó.

Quanto aos testes de desempenho, a virtualização pode apresentar pequena interferência nos resultados dos mesmos. Também é importante avaliar mais métricas nos cenários de teste, como quantidade de operações de Entrada/Saída, uso de CPU e memória. Mais testes podem ser executados fragmentando a base do Intermine usando mais atributos selecionados na execução do algoritmo. Outros tipos de BD devem ser usados para avaliar o algoritmo.

Também estão planejados para o futuro testes em ambientes maiores, com 32 e 64 nós, usando servidores da nuvem Microsoft Azure ou Amazon EC3. Estes testes devem mostrar claramente os pontos de estagnação da escalabilidade vertical e a partir de onde que a fragmentação da base se torna a única alternativa viável.

Os testes de desempenho mostraram problemas em consultas baseadas em CPU (*CPU-Bound*). Esse problema pode ser endereçado em pesquisas posteriores ou como demanda para melhorias e otimizações no PostgreSQL.

Conforme foi identificado na Seção 6.4.2.5, seria de grande valor automatizar a execução dos testes de desempenho, visando auxiliar os DBAs a encontrar a combinação ideal de predicados para fragmentar seus BDs.

Também seria de grande relevância executar os outros algorit-

mos utilizados na Análise Qualitativa (Seção 6.3) sobre a mesma base do Intermine utilizada no Estudo de Caso e analisar as diferenças na alocação gerada. Além disso, pode-se executar os testes de desempenho com os outros esquemas de alocação para avaliar as diferenças.

Algumas otimizações serão feitas no pgGrid para melhorar os planos de consulta e tratar alguns gargalos identificados neste trabalho. As alterações planejadas também envolvem disponibilizar mais estatísticas sobre os fragmentos e o custo de interconexão entre cada par de nós para escolher os melhores nós para executar junções e agregações. O trabalho de Abdalla e Amer (2012) pode ser usado como base para a escolha das métricas de custo envolvidas na alocação e manutenção dos dados distribuídos.

Todas as limitações descritas aqui podem gerar valiosos trabalhos futuros.

REFERÊNCIAS

10GEN. *MongoDB*. 2007. <https://www.mongodb.org>. Acessado em 05/05/2013.

10GEN. *Sharding*. 2011. <http://docs.mongodb.org/manual/sharding/>. Acessado em 05/05/2013.

ABDALLA, H.; AMER, A. Dynamic horizontal fragmentation, replication and allocation model in ddbss. In: *Information Technology and e-Services (ICITeS), 2012 International Conference on*. [S.l.: s.n.], 2012. p. 1–7.

ABOUZEID, A. et al. Hadoopdb: An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proc. VLDB Endow.*, VLDB Endowment, v. 2, n. 1, p. 922–933, 2009. ISSN 2150-8097.

ADL, R. K.; ROUHANI, S. M. T. R. A new ant colony optimization based algorithm for data allocation problem in distributed databases. *Knowledge and Information Systems*, Springer-Verlag, v. 20, n. 3, p. 349–373, 2009. ISSN 0219-1377. <http://dx.doi.org/10.1007/s10115-008-0182-y>.

AHMAD, I. et al. Evolutionary algorithms for allocating data in distributed database systems. *Distributed and Parallel Databases*, Kluwer Academic Publishers, v. 11, n. 1, p. 5–32, 2002. ISSN 0926-8782.

ANDERSON, D. et al. Multimedia systems. In: *Proceedings of the Tenth Annual International Phoenix Conference on Computers and Communications*. [S.l.: s.n.], 1991. p. 719–724.

ANDERSON, D. P. Boinc: A system for public-resource computing and storage. In: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA: IEEE Computer Society, 2004. (GRID '04), p. 4–10. ISBN 0-7695-2256-4. <http://dx.doi.org/10.1109/GRID.2004.14>.

APACHE. *Apache Hadoop*. 2005. <http://hadoop.apache.org/>. Acessado em 05/05/2013.

BELLATRECHE, L.; CUZZOCREA, A.; BENKRID, S. Query optimization over parallel relational data warehouses in distributed environments by simultaneous fragmentation and allocation. In: *Algorithms and Architectures for Parallel Processing*. [S.l.]: Springer Berlin Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6081). p. 124–135. ISBN 9783642131189.

BENSON, D. A. et al. Genbank. *Nucleic Acids Research*, v. 33, n. Database-Issue, p. 34–38, 2005. <http://dblp.uni-trier.de/db/journals/nar/nar33.html#BensonKLOW05>.

BERRY, J. *Beyond Mapping: Concepts, Algorithms, and Issues in GIS*. [S.l.]: Wiley, 1996. ISBN 9780470236765.

BIENVENUT, W. V. *Acceleration and Improvement of Protein Identification by Mass Spectrometry*. [S.l.]: Springer, 2005. 298 p.

BIOINFORMATICS. *Bioseg*. 2008. <http://www.bioinformatics.org/bioseg/>. Acessado em 05/05/2013.

BOLSER, D. M. et al. Metabase the wiki-database of biological databases. *Nucleic Acids Research*, v. 1, p. 1–5, 2011.

BRUIJN, N. *Asymptotic Methods in Analysis*. [S.l.]: Dover Publications, 1970. (Bibliotheca mathematica). ISBN 9780486642215.

COCHRANE, G.; GALPERIN, M. Y. The 2010 nucleic acids research database issue and online database collection: a community of data resources. *Nucleic Acids Research*, v. 38, n. Database-Issue, p. 1–4, 2010. <http://dblp.uni-trier.de/db/journals/nar/nar38.html#CochraneG10>.

CODD, E. F.; CODD, S. B.; SALLEY, C. T. Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT Mandate. 1993.

CONTRINO, S. et al. modmine: flexible access to modencode data. *Nucleic Acids Research*, v. 40, n. Database-Issue, p. 1082–1088, 2012.

COULOURIS, G. et al. *Distributed Systems: Concepts and Design*. Boston, USA: Addison-Wesley, 2011. ISBN 0132143011.

DAUDPOTA, N. H. Five steps to construct a model of data allocation for distributed database systems. *Journal of Intelligent Information*

Systems, Kluwer Academic Publishers, v. 11, n. 2, p. 153–168, 1998. ISSN 0925-9902.

DEAN, J.; GHEMAWAT, S. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, ACM, New York, NY, USA, v. 51, n. 1, p. 107–113, jan. 2008. ISSN 0001-0782. <http://doi.acm.org/10.1145/1327452.1327492>.

DEMCHENKO, Y. et al. Addressing big data issues in scientific data infrastructure. In: *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. [S.l.: s.n.], 2013. p. 48–55.

DNA DATA BANK OF JAPAN. *Introduction of DDJB*. 2014. <http://www.ddbj.nig.ac.jp/intro-e.html>. Acessado em 05/05/2014.

DUBITZKY, W. et al. Grid-enabled data warehousing for molecular engineering. *Parallel Computing*, v. 30, n. 9-10, p. 1019–1035, 2004. ISSN 0167-8191. <http://www.sciencedirect.com/science/article/pii/S0167819104000845>.

EKINS, R.; CHU, F. W. Microarrays: their origins and applications. *Trends in Biotechnology*, v. 17, n. 6, p. 217–218, 1999.

ELMASRI, R. et al. *Sistemas de banco de dados*. [S.l.]: Pearson Addison Wesley, 2005.

Endpoint Corporation. *Bucardo Wiki*. 2008. <http://bucardo.org/wiki/Bucardo>. Acessado em 05/05/2013.

ERDMAN, A.; KEEFE, D.; SCHIESTL, R. Grand challenge: Applying regulatory science and big data to improve medical device innovation. *Biomedical Engineering, IEEE Transactions on*, v. 60, n. 3, p. 700–706, March 2013. ISSN 0018-9294.

FOMKIN, R.; RISCH, T. Cost-based optimization of complex scientific queries. In: *Scientific and Statistical Database Management, 2007. SSBDM '07. 19th International Conference on*. [S.l.: s.n.], 2007. p. 1–1. ISSN 1551-6393.

GOSSET, W. W. et al. *Student: a statistical biography of William Sealy Gosset*. [S.l.]: Clarendon Press, 1990. 142 p.

GUIZELINI, D. *Banco de dados biológico no modelo relacional para mineração de dados em genomas completos de procariotos disponibilizados pelo NCBI GenBank*. Dissertação (Mestrado) — Universidade Federal do Paraná, Curitiba-PR, 2010.

HUBBLE, J. et al. Implementation of genepattern within the stanford microarray database. *Nucleic Acids Research*, p. D898–901, 2009.

JAECKSCH, B.; LEHNER, W. The planning olap model – a multidimensional model with planning support. In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems VIII*. [S.l.]: Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 7790). p. 32–52. ISBN 978-3-642-37573-6.

JAY, J. S.; JI, H. Next-generation dna sequencing. 2008.

JUN, W.; TEI-WEI, K. Real-time scheduling of cpu-bound and i/o-bound processes. In: *Real-Time Computing Systems and Applications, 1999. RTCSA 99. Sixth International Conference on*. [S.l.: s.n.], 1999. p. 303–310.

KOPETZ, H. *Real-time Systems: Design Principles for Distributed Embedded Applications*. [S.l.]: Kluwer Academic Publishers, 1997. (Kluwer international series in engineering and computer science). ISBN 9780792398943.

LASHKARI, D. A. et al. Yeast microarrays for genome wide parallel genetic and gene expression analysis. *Proceedings of the National Academy of Sciences*, v. 94, n. 24, p. 13057–13062, 1997.

MELL, P.; GRANCE, T. *The NIST Definition of Cloud Computing*. [S.l.], September 2011. <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.

MOHAJIR, M. E.; LATRACHE, A. Unifying and incorporating functional and non functional requirements in datawarehouse conceptual design. In: *2012 Colloquium in Information Science and Technology (CIST)*. [S.l.: s.n.], 2012. p. 49–57.

MUSHEGIAN, A. Grand challenges in bioinformatics and computational biology. *Frontiers in Genetics*, v. 2, n. 60, 2011. ISSN 1664-8021.

NASSAR, S. et al. *SEstatNet - Sistema Especialista para o Ensino de Estatística na Web*. 2005. <http://www.sestat.net>. Acessado em 05/05/2014.

NAVATHE, S. et al. Vertical partitioning algorithms for database design. *ACM Trans. Database Syst.*, ACM, New York, NY,

USA, v. 9, n. 4, p. 680–710, dez. 1984. ISSN 0362-5915. <http://doi.acm.org/10.1145/1994.2209>.

NCBI. *Registry of Biological Repositories*. 2005. <http://www.biorepositories.org>. Acessado em 05/05/2013.

NCBI. *Genbank Report*. 2013. <ftp://ftp.ncbi.nih.gov/genbank/gbrel.txt>. Acessado em 05/07/2014.

NELSON, D. L.; COX, M. M. *Lehninger Principles of Biochemistry, Fourth Edition*. Fourth edition. [S.l.: s.n.], 2004.

NIH. *The Cancer Genome Atlas homepage*. 2005. <http://cancergenome.nih.gov/>. Acessado em 01/05/2013.

Oracle Corporation. *MySQL Cluster*. 2004. <http://www.mysql.com/products/cluster/>. Acessado em 05/05/2013.

Oracle Corporation. *Oracle OLAP*. 2005. <http://docs.oracle.com/html/B13915_04/i_olap_chapter.htm>. Acessado em 05/05/2013.

Oracle Corporation. *Partitioning Concepts*. 2007. <http://docs.oracle.com/cd/B28359_01/server.111/b32024/partition.htm>. Acessado em 05/01/2014.

PARKER, T.; RITCHEY, P. *High Capacity Single Table Performance Design Using Partitioning in Oracle or PostgreSQL*. 2012. <www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA559376>. Acessado em 06/01/2014.

PAUL, T. K.; IBA, H. Prediction of cancer class with majority voting genetic programming classifier using gene expression data. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. [S.l.: s.n.], 2009. v. 6, p. 353–367.

PETTERSSON, E.; LUNDEBERG, J.; AHMADIAN, A. Generations of sequencing technologies. *Genomics*, v. 93, n. 2, p. 105 – 111, 2009. ISSN 0888.

PostgreSQL Development Group. *About PostgreSQL*. 2005. <http://www.postgresql.org>. Acessado em 05/05/2013.

PostgreSQL Development Group. *PostgreSQL Documentation: Indices*. 2005. <http://www.postgresql.org/docs/9.1/static/indices.html>. Acessado em 05/02/2014.

RONAGHI, M.; UHLéN, M.; NYRéN, P. A sequencing method based on real-time pyrophosphate. *Science*, v. 281, n. 5375, p. 363–365, 1998. <http://www.sciencemag.org/content/281/5375/363.short>.

SANGER, F.; NICKLEN, S.; COULSON, A. DNA sequencing with chain-terminating inhibitors. *Proceedings of The National Academy of Sciences of The United States Of America*, NATL ACAD SCIENCES, 74, n. 12, p. 5463–5467, 1977. ISSN 0027-8424.

SHARMA, V. *Bitmap Index vs. B-tree Index: Which and When?* 2005. <http://www.oracle.com/technetwork/articles/sharma-indexes-093638.html>. Acessado em 05/02/2014.

SHIN, S. K.; SANDERS, G. L. Denormalization strategies for data retrieval from data warehouses. *Decis. Support Syst.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 42, n. 1, p. 267–282, out. 2006. ISSN 0167-9236. <http://dx.doi.org/10.1016/j.dss.2004.12.004>.

Slony Development Group. *Slony-I web site*. 2007. <http://slony.info/>. Acessado em 05/05/2013.

SMITH, R. N. et al. Intermine: a flexible data warehouse system for the integration and analysis of heterogeneous biological data. *Bioinformatics*, p. 3163–3165, 2012.

SNUSTAD, P. D.; SIMMONS, M. J. *Fundamentos de Genética*. [S.l.]: Guanabara Koogan, 2010. 903 p. ISBN 9788527713740.

SPARKMAN, O. D. *Mass spectrometry desk reference*. Pittsburgh, USA: Global View, 2006. 198 p. ISBN 0-9660813-2-3.

STRIMBU, K.; TRAVEL, J. A. What are biomarkers? *Current Opinion on HIV/AIDS*, v. 5, n. 6, p. 463–466, 2010.

STURGEON, B. et al. Can the grid help to solve the data integration problems in molecular biology? In: *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*. [S.l.: s.n.], 2003. p. 594–600.

THOMSEN, E. *OLAP Solutions: Building Multidimensional Information Systems*. [S.l.]: Wiley, 2002. 661 p. (Wiley Computer Publishing). ISBN 9780471400301.

TONINI, G.; SIQUEIRA, F. pggrid: uma implementação de fragmentação de dados para o postgresql. In: *Anais do XV Workshop sobre Software Livre*. [S.l.: s.n.], 2010. p. 48–53.

TONINI, G.; SIQUEIRA, F. A distributed data allocation algorithm for biological databases. In: *Computational Science and Engineering (CSE), 2013 IEEE 16th International Conference on*. [S.l.: s.n.], 2013.

TONINI, G.; SIQUEIRA, F. Evaluation of computer methods for biomarker discovery on computational grids. *Electronic Journal of Biotechnology*, v. 16, p. 13–13, 09 2013. ISSN 0717-3458.

UNICORE Forum. *Unicore Plus Final Report*. 2003. <http://www.unicore.eu/documentation/files/erwin-2003-UPF.pdf>. Acessado em 05/05/2013.

WEISS, V. A. *Estratégias de finalização da montagem do genoma da Bactéria Diazotrófica Endofítica Herbaspirillum Seropedicae SmR1*. Dissertação (Mestrado) — Universidade Federal do Paraná, Curitiba-PR, 2010.

WU, M. et al. Biomarker identification and cancer classification based on microarray data using laplace naive bayes model with mean shrinkage. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. [S.l.: s.n.], 2012. v. 9, p. 1649–1662.

ÖZSU, M. T.; VALDURIEZ, P. *Principles of Distributed Database Systems*. New York, USA: Springer, 2011. ISBN 1441988335.

ANEXO A – Esquema de dados do Intermine

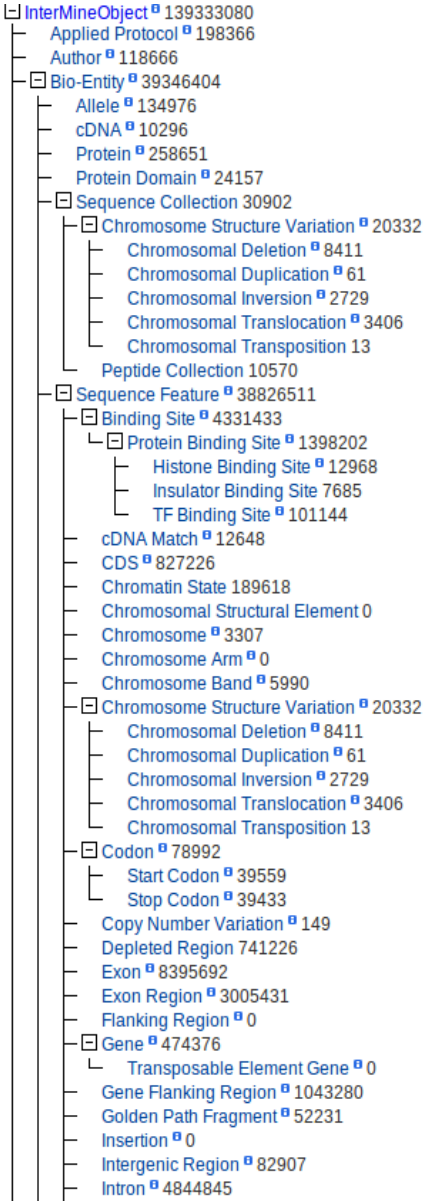


Figura 23 – Esquema de dados Intermine. Fonte: Smith et al. (2012)

ANEXO B – Consultas do Experimento 2

Consulta 142053:

```
SELECT DISTINCT a1\_.class AS a2\_
FROM SequenceFeature AS a1\_,
OverlappingFeaturesSequenceFeature AS indirect0
WHERE 11000443 = indirect0.SequenceFeature
AND indirect0.OverlappingFeatures = a1\_.id
ORDER BY a1\_.class LIMIT 1000
```

Consulta 124543:

```
SELECT a3\_.id AS a1\_
FROM CopyNumberVariation AS a2\_,
Gene AS a3\_, Location AS a4\_,
Location AS a5\_,
FeaturesSubmissions AS indirect0
WHERE a5\_.featureId = a3\_.id
AND a3\_.chromosomeId = a2\_.chromosomeId
AND a3\_.organismId = a2\_.organismId
AND a3\_.organismId = 1000001
AND 63000018 = indirect0.Submissions
AND indirect0.Features = a2\_.id
AND a4\_.locatedOnId = a5\_.locatedOnId
AND bioseg\_create(a4\_.intermine\_start, a4\_.intermine\_end) &&
 bioseg\_create(GREATEST(1,(a5\_.intermine\_start - 0)),
(a5\_.intermine\_end + 0))
ORDER BY a3\_.id LIMIT 100000
```

Consulta 158533:

```
SELECT DISTINCT a1_.id AS a2_, a1_.primaryIdentifier AS a3_,
a1_.secondaryIdentifier AS a4_, a1_.symbol AS a5_,
a1_.name AS a6_
FROM BioEntity AS a1_, Organism AS a7_
WHERE (LOWER(a1_.primaryIdentifier) IN ('fbgn0000363',
'fbgn0001205',
'fbgn0002968', 'fbgn0003425', 'fbgn0003600', 'fbgn0003659',
'fbgn0004603','fbgn0011577', 'fbgn0013948', 'fbgn0017397',
'fbgn0024252', 'fbgn0024308','fbgn0026087', 'fbgn0029764',
'fbgn0029846', 'fbgn0029933', 'fbgn0030248','fbgn0030842',
'fbgn0031993', 'fbgn0036115', 'fbgn0039060', 'fbgn0051048',
```

```
'fbgn0052541', 'fbgn0052743', 'fbgn0053205', 'fbgn0262718'
'fbgn0053555', 'fbgn0053936', 'fbgn0064121', 'fbgn0064122',
'fbgn0085403', 'fbgn0086689', 'fbgn0259100', 'fbgn0259743',
'fbgn0259934', 'fbgn0260232', 'fbgn0260966', 'fbgn0261812',
'fbgn0263610') OR LOWER(a1_.secondaryIdentifier) IN ('fbgn0000363',
'fbgn0001205', 'fbgn0002968', 'fbgn0003425', 'fbgn0003600',
'fbgn0003659','fbgn0004603', 'fbgn0011577', 'fbgn0013948',
'fbgn0024252', 'fbgn0024308', 'fbgn0026087', 'fbgn0029764',
'fbgn0029846', 'fbgn0029933', 'fbgn0017397', 'fbgn0260232',
'fbgn0030248', 'fbgn0030842', 'fbgn0031993', 'fbgn0036115',
'fbgn0039060', 'fbgn0051048', 'fbgn0052541', 'fbgn0052743',
'fbgn0053205', 'fbgn0053555', 'fbgn0262718', 'fbgn0263610',
'fbgn0053936', 'fbgn0064121', 'fbgn0064122', 'fbgn0085403',
'fbgn0086689', 'fbgn0259100', 'fbgn0259743', 'fbgn0259934',
'fbgn0260966', 'fbgn0261812') OR
LOWER(a1_.symbol) IN ('fbgn0000363', 'fbgn0001205', 'fbgn0002968',
'fbgn0003425', 'fbgn0003600', 'fbgn0003659', 'fbgn0004603',
'fbgn0011577', 'fbgn0013948', 'fbgn0017397', 'fbgn0024252',
'fbgn0024308', 'fbgn0026087', 'fbgn0029764', 'fbgn0029846',
'fbgn0029933', 'fbgn0030248', 'fbgn0030842', 'fbgn0031993',
'fbgn0036115','fbgn0039060', 'fbgn0053555',
'fbgn0051048', 'fbgn0052541', 'fbgn0052743', 'fbgn0053205',
'fbgn0053936', 'fbgn0064121', 'fbgn0064122', 'fbgn0085403',
'fbgn0086689', 'fbgn0259100', 'fbgn0259743', 'fbgn0259934',
'fbgn0260232', 'fbgn0260966', 'fbgn0261812', 'fbgn0262718',
'fbgn0263610') OR LOWER(a1_.name) IN ('fbgn0000363',
'fbgn0001205', 'fbgn0002968', 'fbgn0003425', 'fbgn0003600',
'fbgn0003659', 'fbgn0004603', 'fbgn0011577', 'fbgn0013948',
'fbgn0017397', 'fbgn0024252', 'fbgn0024308', 'fbgn0026087',
'fbgn0029764', 'fbgn0029846', 'fbgn0029933', 'fbgn0030248',
'fbgn0030842', 'fbgn0031993', 'fbgn0036115', 'fbgn0039060',
'fbgn0051048', 'fbgn0052541', 'fbgn0052743', 'fbgn0053205',
'fbgn0053555', 'fbgn0053936', 'fbgn0064121', 'fbgn0064122',
'fbgn0085403', 'fbgn0086689', 'fbgn0259100', 'fbgn0259743',
'fbgn0259934', 'fbgn0260232', 'fbgn0260966', 'fbgn0261812',
'fbgn0262718', 'fbgn0263610')) AND a1_.organismId = a7_.id
AND a7_.shortName = 'D. melanogaster' ORDER BY a1_.id LIMIT 10000
```

Consulta 113519:

```
SELECT DISTINCT a1_.a6_ AS a2_, a1_.a5_ AS a3_,
a1_.a7_ AS a4_, a1_.a8_ AS a5_, a1_.a10_ AS a6_,
```

```
a1_.a9_ AS a7_, SUM(a1_.a3_) AS a8_
FROM (SELECT COUNT(*) AS a3_, a1_.a2_ AS a4_,
a2_.a3_ AS a5_, a2_.a2_ AS a6_, a2_.a4_ AS a7_,
a2_.a5_ AS a8_, WIDTH_BUCKET(a1_.a2_, a2_.a2_,
((a2_.a3_)::NUMERIC * (1.01)::NUMERIC), a2_.a6_) AS a9_,
a2_.a6_ AS a10_ FROM (SELECT a1_.a3_ AS a2_
FROM (SELECT a1_.briefDescription AS a1_briefDescription,
a1_.chromosomeId AS a1_chromosomeId,
a1_.chromosomeLocationId AS a1_chromosomeLocationId,
a1_.cytoLocation AS a1_cytoLocation,
a1_.description AS a1_description,
a1_.downstreamIntergenicRegionId AS
a1_downstreamIntergenicRegionId,
a1_.id AS a1_id, a1_.length AS a1_length,
a1_.name AS a1_name, a1_.note AS a1_note,
a1_.organismId AS a1_organismId, a1_.predictionStatus AS
a1_predictionStatus, a1_.primaryIdentifier AS
a1_primaryIdentifier, a1_.score AS a1_score,
a1_.scoreProtocolId AS a1_scoreProtocolId,
a1_.scoreType AS a1_scoreType, a1_.secondaryIdentifier AS
a1_secondaryIdentifier, a1_.sequenceId AS a1_sequenceId,
a1_.sequenceOntologyTermId AS a1_sequenceOntologyTermId,
a1_.source AS a1_source, a1_.symbol AS a1_symbol,
a1_.upstreamIntergenicRegionId AS a1_upstreamIntergenicRegionId,
a2_.commonName AS a2_commonName, a2_.genus AS a2_genus,
a2_.id AS a2_id, a2_.name AS a2_name,
a2_.shortName AS a2_shortName,
a2_.species AS a2_species, a2_.taxonId AS a2_taxonId,
a1_.length AS a3_
FROM Gene AS a1_, Organism AS a2_
WHERE a1_.organismId = a2_.id
AND LOWER(a2_.shortName) LIKE 'c. elegans') AS a1_) AS a1_,
(SELECT DISTINCT MIN(a1_.a3_) AS a2_, MAX(a1_.a3_) AS a3_,
AVG(a1_.a3_) AS a4_, STDDEV(a1_.a3_) AS a5_,
LEAST(20,(MAX(a1_.a3_) - MIN(a1_.a3_))) AS a6_
FROM (SELECT a1_.briefDescription AS a1_briefDescription,
a1_.chromosomeId AS a1_chromosomeId,
a1_.chromosomeLocationId AS a1_chromosomeLocationId,
a1_.cytoLocation AS a1_cytoLocation,
a1_.description AS a1_description,
a1_.downstreamIntergenicRegionId AS
```

```
a1_downstreamIntergenicRegionId,
a1_.id AS a1_id, a1_.length AS a1_length, a1_.name AS a1_name,
a1_.note AS a1_note, a1_.organismId AS a1_organismId,
a1_.predictionStatus AS a1_predictionStatus,
a1_.primaryIdentifier AS a1_primaryIdentifier,
a1_.score AS a1_score, a1_.scoreProtocolId AS a1_scoreProtocolId,
a1_.scoreType AS a1_scoreType, a1_.secondaryIdentifier AS
a1_secondaryIdentifier, a1_.sequenceId AS a1_sequenceId,
a1_.sequenceOntologyTermId AS a1_sequenceOntologyTermId,
a1_.source AS a1_source, a1_.symbol AS a1_symbol,
a1_.upstreamIntergenicRegionId AS a1_upstreamIntergenicRegionId,
a2_.commonName AS a2_commonName, a2_.genus AS a2_genus,
a2_.id AS a2_id, a2_.name AS a2_name,
a2_.shortName AS a2_shortName, a2_.species AS a2_species,
a2_.taxonId AS a2_taxonId, a1_.length AS a3_
FROM Gene AS a1_, Organism AS a2_
WHERE a1_.organismId = a2_.id
AND LOWER(a2_.shortName) LIKE 'c. elegans') AS a1_) AS a2_
GROUP BY a1_.a2_, a2_.a3_, a2_.a2_, a2_.a4_, a2_.a5_, a2_.a6_)
AS a1_
GROUP BY a1_.a6_, a1_.a5_, a1_.a7_, a1_.a8_, a1_.a9_, a1_.a10_
ORDER BY a1_.a9_, a1_.a6_, a1_.a5_, a1_.a7_, a1_.a8_, a1_.a10_, a8_
LIMIT 5000
```

Consulta 117416:

```
SELECT DISTINCT '1' AS a3_
FROM Gene AS a1_, Organism AS a2_, Transcript AS a4_,
Intron AS a5_, IntronsTranscripts AS indirect0
WHERE a2_.name = 'Caenorhabditis elegans'
AND a1_.organismId = a2_.id
AND a1_.id = a4_.geneId AND a4_.id = indirect0.Transcripts
AND indirect0.Introns = a5_.id LIMIT 1
```

Consulta 124963:

```
SELECT a1_.id AS a1_id FROM SequenceFeature AS a1_,
OverlappingFeaturesSequenceFeature AS indirect0
WHERE 63266113 = indirect0.SequenceFeature
AND indirect0.OverlappingFeatures = a1_.id
AND a1_.id > 73598029 ORDER BY a1_.id
LIMIT 1000 OFFSET 4001
```

Consulta 53389:

```
SELECT a1_.id AS a1_id FROM Location AS a1_
WHERE 56059255 = a1_.featureId
ORDER BY a1_.id LIMIT 1000
```

Consulta 136931:

```
SELECT DISTINCT '1' AS a3_ FROM Gene AS a1_,
Organism AS a2_, Transcript AS a4_, CDS AS a5_
WHERE a2_.name = 'Caenorhabditis elegans'
AND a1_.organismId = a2_.id
AND a1_.id = a4_.geneId
AND a4_.id = a5_.transcriptId LIMIT 1
```

Consulta 57018:

```
SELECT DISTINCT a1_.a4_ AS a2_, COUNT(*) AS a3_ FROM
(SELECT a1_.briefDescription AS a1_briefDescription,
a1_.chromosomeId AS a1_chromosomeId,
a1_.chromosomeLocationId AS a1_chromosomeLocationId,
 a1_.cytoLocation AS a1_cytoLocation,
a1_.description AS a1_description,
a1_.downstreamIntergenicRegionId AS
a1_downstreamIntergenicRegionId, a1_.id AS a1_id,
a1_.length AS a1_length, a1_.name AS a1_name,
a1_.note AS a1_note, a1_.organismId AS a1_organismId,
a1_.predictionStatus AS a1_predictionStatus,
a1_.primaryIdentifier AS a1_primaryIdentifier,
a1_.score AS a1_score, a1_.scoreType AS a1_scoreType,
a1_.scoreProtocolId AS a1_scoreProtocolId,
a1_.secondaryIdentifier AS a1_secondaryIdentifier,
a1_.sequenceId AS a1_sequenceId,
a1_.sequenceOntologyTermId AS a1_sequenceOntologyTermId,
a1_.source AS a1_source, a1_.symbol AS a1_symbol,
a1_.upstreamIntergenicRegionId AS
a1_upstreamIntergenicRegionId,
a2_.commonName AS a2_commonName, a2_.genus AS a2_genus,
a2_.id AS a2_id, a2_.name AS a2_name,
a2_.shortName AS a2_shortName, a2_.species AS a2_species,
a2_.taxonId AS a2_taxonId, a3_.dataSourceId AS a3_dataSourceId,
a3_.description AS a3_description, a3_.id AS a3_id,
a3_.name AS a3_name, a3_.url AS a3_url,
```

```
a3_.version AS a3_version, a3_.name AS a4_
FROM Gene AS a1_, Organism AS a2_, DataSet AS a3_,
BioEntitiesDataSets AS indirect0
WHERE a1_.organismId = a2_.id
AND a1_.id = indirect0.BioEntities
AND indirect0.DataSets = a3_.id
AND a1_.length IS NULL AND
LOWER(a3_.name) LIKE 'ncbi entrez gene identifiers') AS
a1_ GROUP BY a1_.a4_
ORDER BY COUNT(*) DESC, a1_.a4_ LIMIT 5000
```

Consulta 247582:

```
SELECT a1_.OBJECT AS a1_, a1_.id AS a1_id
FROM InterMineObject AS a1_ WHERE a1_.id IN (76005653)
ORDER BY a1_.id
```