Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

VIRTUALIZAÇÃO DE GRANDES BASES DE DADOS IRREGULARMENTE DISTRIBUÍDAS E REPLICADAS

Dissertação apresentada ao Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

DIÊGO LOPES NOGUEIRA

Belo Horizonte, MG
01 de dezembro de 2006

UNIVERSIDADE FEDERAL DE MINAS GERAIS

FOLHA DE APROVAÇÃO

Virtualização de Grandes Bases de Dados Irregularmente Distribuídas e Replicadas

DIÊGO LOPES NOGUEIRA

Dissertação defendida e aprovada pela banca examinadora constituída por:

Ph. D. Renato Antônio Celso Ferreira – Orientador
Universidade Federal de Minas Gerais

Ph. D. Dorgival Olavo Guedes Neto – Co-orientador
Universidade Federal de Minas Gerais

Ph. D. Wagner Meira Jr.
Universidade Federal de Minas Gerais

Belo Horizonte, MG, 01 de dezembro de 2006

Resumo

Grandes volumes de dados são gerados diariamente por experimentos, simulações e vários outros tipos de aplicações. É comum observar situações nas quais porções desses dados são irregularmente replicadas e distribuídas em diferentes fontes de dados. A replicação e distribuição irregulares se dão pela geração independente dos diferentes intervalos de dados correlacionados e pela freqüente ausência de colaboração no compartilhamento de parte desses volumes de dados.

É desejável ser possível lidar com essas várias porções de dados irregulares (replicados ou não) como um único repositório. A virtualização de dados torna isto possível e é o principal foco deste trabalho. Nesta dissertação, exploramos um sistema capaz de lidar com dados irregularmente replicados e criar uma visão virtual única a partir da união de porções irregulares dos dados disponíveis em cada uma das fontes de dados que compõem o sistema.

Apresentamos uma modelagem geométrica dos intervalos de dados que suporta a virtualização de um repositório de dados irregularmente replicado e distribuído, assim como um mecanismo de indexação de meta-dados que permite o processamento de consultas por intervalos de dados submetidas ao repositório disponível no sistema de virtualização de dados.

Esta dissertação também propõe dois algoritmos de escalonamento de fragmentos de consultas baseados nas abordagens gulosa e recozimento simulado. Esses algoritmos são responsáveis pela seleção de qual servidor é responsável por servir cada fragmento de uma consulta por intervalo de dados. Os algoritmos buscam minimizar o tempo de resposta das consultas e balancear a carga de trabalho entre os servidores considerando sua capacidade de serviço e carga de trabalho instantânea. A comparação de desempenho dos algoritmos é baseada em simulações e os valores dos parâmetros utilizados foram obtidos a partir da caracterização da carga de trabalho de uma aplicação real e fortemente dependente de dados (o Microscópio Virtual).

Abstract

Large volumes of data are generated every day by experiments, simulations and all sorts of applications. It is common to observe situations where portions of data are irregularly replicated and distributed in different data sources. The independent generation of correlated data and lack of collaboration on sharing these data result in an irregularly replicated and distributed data set.

It would be desirable to be able to handle these several pieces of irregular data (replicated or not) as a unique large dataset. This is called data virtualization and is the focus of this work. On this dissertation, we explore a system which is capable of dealing with irregularly replicated data and is able to create a virtual view of the union of the individual irregular portions of data hosted by each data source.

We present a geometric model to represent data intervals. The model allows for virtualization of an irregularly replicated and distributed data set. The work also presents a meta-data indexing mechanism to allow the system to process ranged queries submitted to the data set available through the data virtualization system.

Two query fragment scheduling algorithms are proposed, based on the greedy and simulated annealing approaches. These algorithms are responsible for the selection of which server will be in charge of serving each data query fragment. The algorithms try to minimize the queries' response time and to balance the load between the servers, taking into account their different service capacities and the workload to which each server is submitted to at any given time. The performances of the algorithms are compared based on simulation results and the parameter values used were taken from the workload characterization of a real data-oriented application (the Virtual Microscope).

Em memória de minha avó, Marília, que muito torceu
pelo meu ingresso na vida acadêmica.

Agradecimentos

Agradeço enormemente aos meus pais, Waldir e Zoraide, sem os quais nada disto seria possível. Com o carinho e a total dedicação à minha educação, me ajudaram de forma decisiva a dar os passos certos para chegar até aqui. São os melhores pais que alguém poderia sonhar em ter, e por isso, muito obrigado. Por todo o apoio e companhia nos estudos, alegrias e etapas difíceis de minha vida, muito obrigado. Não poderia deixar de agradecer também minha irmã, Samantha, pela torcida, paciência e apoio durante todos esses anos. Amo vocês. Aos meus familiares, muito obrigado pela torcida!

Agradeço aos meus professores e mestres Renato Ferreira, Wagner Meira Jr. e Dorgival Guedes Neto, que cumpriram um papel imprescindível em minha formação acadêmica. Muito obrigado pelas lições, pela amizade e por todo o apoio que me ofereceram. Muito obrigado também ao Prof. Ricardo Bianchini, que me recebeu na universidade Rutgers como aluno e amigo e, em tão pouco tempo, tanto me ensinou.

Aos meus colegas do laboratório e-SPEED, obrigado pelo companheirismo, por toda a ajuda que me ofereceram, pelo excelente trabalho em equipe e por manter o ambiente descontraído mesmo nas épocas mais atribuladas. Em especial, muito obrigado a Bruno Diniz e André Cardoso, sem a ajuda dos quais este trabalho não seria finalizado. Aos amigos do Dark Lab, muito obrigado por me acolherem tão bem.

Durante a minha passagem pela UFMG, fiz várias e valiosas amizades que ainda hoje são muito importantes para mim. Gostaria de agradecer, em especial, aos "desmaiados" (Juliano Santos, Robert Pinto, Stenio Viveiros, Lucas Issa, Leonardo Rocha, Isabela Guimarães, Gisele Cardoso e André Goddard) pela indispensável companhia em incontáveis e intermináveis fins-de-semana e madrugadas de estudos. Foram eles quem me ensinaram o que era estudar de verdade.

Agradeço também aos companheiros da Base2 Tecnologia, por tudo que me ensinaram, pelo apoio que me deram e pelo excelente trabalho que desenvolvemos juntos. Desejo-lhes muito sucesso!

Não poderia deixar de agradecer aos meus amigos do peito, os chamados "X-Men". Lele, Gordão, Rom, Baguete e Cabeção, muito obrigado pela torcida, pela paciência e compreensão nos momentos em que não pude acompanhá-los e pela grande amizade que cultivamos.

Agradecimento Especial

Gostaria de agradecer alguém muito especial, minha noiva, Vanessa.

Você foi meu norte desde os primeiros passos desta caminhada. Antes mesmo da escolha do curso universitário, já me motivava e acompanhava nos estudos. Durante toda a trajetória do vestibular, esteve ao meu lado, me impulsionando com sua incansável torcida. E, sempre da primeira fila, aplaudiu minhas conquistas.

A minha dedicação aos estudos nos privou de muitos momentos juntos: os fins-de-semana no ICEx, as semanas em viagens a congressos e até mesmo os meses de ausência, em que estudei em outra universidade. Sacrifícios demais, muitos disseram. Mas lá estava você, na primeira fila, torcendo.

Essa caminhada não foi feita só de conquistas e alegrias. No entanto, tombo após tombo, sempre encontrei sua mão estendida, que me guiou para retomar cada passo perdido. Como conseguiu tudo isso enquanto traçava sua própria caminhada, é impossível compreender. Saiba que nenhuma dessas conquistas seria possível sem o seu carinho, sem o seu apoio, sem a sua paciência. Esta vitória é sua também. Muito obrigado por estar sempre ao meu lado.

Sei que a caminhada não termina aqui, apenas se transforma. Sei também que a próxima etapa não será feita apenas de conquistas e alegrias. Nada me deixa mais feliz do que saber que escolhemos fazer das nossas vidas uma única trajetória, juntos. Muito obrigado por existir em minha vida.

Eu te amo, muito e pra sempre.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Diversas categorias de aplicações, como simulações de fenômenos naturais e aplicações científicas, dependem fortemente de dados e geram grandes quantidades de informação. Frequentemente, o volume de dados tratado por essas aplicações é grande o suficiente para que seja necessário sua distribuição em diversos servidores. Em muitos casos, esses dados são correlacionados, ainda que possivelmente gerados por processos independentes. Por exemplo, diferentes grupos de pesquisa em física poderiam simular o mesmo fenômeno físico utilizando diferentes intervalos de parâmetros, podendo gerar porções disjuntas de informação. Possíveis intersecções entre esses intervalos de parâmetros permitem que dados replicados também sejam gerados por esses processos independentes.

Como normalmente altos custos computacionais estão associados à geração dos dados produzidos por essa categoria de aplicações, é desejável que diferentes organizações possam compartilhar suas porções dos dados, resultando na disponibilização de um repostiório de dados único e mais abrangente.

À medida que os dados das diferentes organizações são gerados independentemente, intervalos desses dados são replicados sem qualquer noção global. Esta replicação de maneira completamente desorganizada resulta em configurações irregulares de dados sob o ponto de vista de um repositório global.

Quando distribuídos, diferentes porções desses dados tendem a estar armazenadas através de diferentes sistemas, como bancos de dados ou sistemas de arquivos. O gerenciamento das diferentes interfaces de acesso aos vários sistemas de armazenamento de dados se torna uma tarefa muito complexa para ser desempenhada por todos os usuários dos dados. Assim, é importante que exista uma interface que torne transparente a localização dos dados, assim como seu mecanismo de armazenamento. Esta interface transparente de acesso aos dados é provida por um serviço conhecido como virtualização de dados.

Para compartilhar seus dados através dos sistemas de virtualização convencionais, as organizações precisariam delegar recursos extras de armazenamento, assim como os dados a serem compartilhados. O sistema de virtualização, por sua vez, utilizaria esses recursos de armazenamento para distribuir os dados de maneira organizada e otimizada, resultando em uma configuração regular dos dados. Além de demandar uma coordenação entre as diferentes

organizações detentoras dos dados, a utilização desses sistemas implicaria na transferência de grandes quantidades de dados através da rede. Como a movimentação de dados entre diferentes computadores, interligados por redes locais ou remotas, é uma operação computacionalmente cara, é desejável que os repositórios de dados, ainda que distribuídos, sejam acessados eficientemente.

As Figuras 1.1 e 1.2 exibem uma abstração de configurações de repositórios de dados com espaço de parâmetros bi-dimensional. Considerando que múltiplos servidores armazenam os dados, a escala de cores nestas figuras diferenciam os grupos de servidores que armazenam os intervalos de dados coloridos. Por exemplo, os dados coloridos de branco poderiam estar disponíveis nos servidores A, C e D, enquanto os coloridos de preto estariam disponíveis nos servidores A, D, G e H.

A Figura 1.1 ilustra uma configuração regular de dados. Nessa configuração, a distribuição e replicação de dados foi realizada de forma organizada. Para a criação de configurações como esta, é necessário que os recursos de armazenamento sejam delegados para um sistema central, que distribui os dados de maneira coordenada dentre os servidores do repositório.

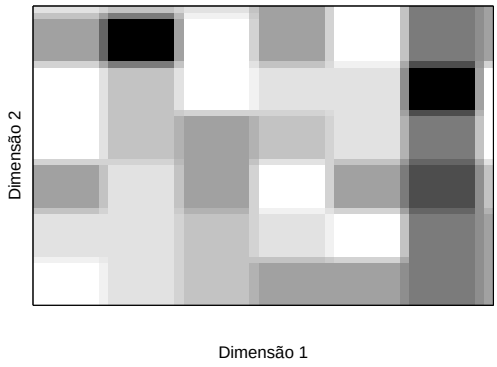


Figura 1.1: **Configuração regular de dados.** Os dados são distribuídos e replicados de maneira organizada. É possível especificar funções que descrevam a distribuição e replicação dos dados.

Em contraste, a Figura 1.2 ilustra uma configuração irregular de dados. Nessa configuração, não houve qualquer coordenação entre os processos na geração ou distribuição dos dados. Neste cenário, não há a necessidade de transferência dos dados e provimento de recursos extras de armazenamento para agregar os dados ao repositório virtual.

Ainda que existam inúmeros estudos sobre o provimento de acesso a dados regularmente distribuídos e sobre processamento distribuído de consultas, ainda é um desafio prover virtualização para dados distribuídos e replicados de maneira irregular.

Este trabalho propõe uma arquitetura que busca viabilizar o processamento de consultas por intervalos de dados submetidas a um repositório resultante da união de porções de dados distribuídas e replicadas irregularmente, complementando esforços anteriores em virtualização de dados. Essa arquitetura se baseia em uma representação de meta-dados para intervalos de

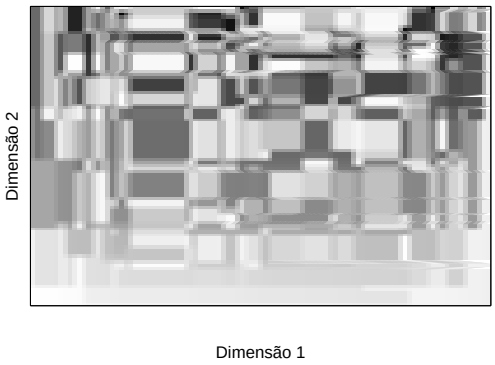


Figura 1.2: **Configuração irregular de dados.** Não houve noção global entre os geradores ou distribuidores de dados, o que resultou em uma replicação e distribuição desorganizada dos dados.

dados correlacionados e distribuídos em diferentes servidores e em um modelo de indexação dos meta-dados baseado em algoritmos geométricos.

Esse modelo permite que um sistema de virtualização de dados opere em configurações de dados distribuídos e replicados de maneira irregular. O modelo também se estende ao processamento das consultas em repositórios de dados com essas características. Tanto as porções de dados quanto as consultas são representadas geometricamente e são utilizados algoritmos de decomposição de polígonos [O'Rourke (1998)] para gerar retângulos maximais disjuntos, que representam intervalos de dados armazenados em um ou mais servidores. Este trabalho não explora a representação de bases de dados baseadas em atributos através de dimensões espaciais. Apesar do modelo não se restringir a bases de dados definidas através de coordenadas espaciais, como imagens ou objetos georeferenciados, este trabalho apresenta exemplos concretos apenas para essa categoria de dados, uma vez que o mapeamento de dados baseados em atributos requer um estudo mais aprofundado.

Uma vez identificados os servidores candidatos, é necessário que um algoritmo guie a escolha de qual servidor atenderá, completa ou parcialmente, cada consulta por intervalos de dados. Este trabalho explora dois algoritmos de seleção de fontes de dados e os contrasta com algoritmos básicos de escalonamento previamente estudados [Diniz et alia (2006)]. O primeiro algoritmo explorado neste trabalho é baseado em uma abordagem gulosa enquanto o segundo se baseia na abstração de recozimento simulado (do inglês, *simulated annealing*) buscando minimizar o tempo de resposta das consultas. Esses algoritmos tomam decisões com base na carga instantânea imposta aos servidores e buscam minimizar o tempo de resposta das consultas e balancear a carga entre os servidores considerando as suas diferentes capacidades de serviço.

Para avaliar o desempenho dos algoritmos propostos, o trabalho faz uso de simulações e variou uma vasta combinação de parâmetros como: taxa de chegada de consultas, tamanho do repositório, nível de replicação de dados e outros. Todos os parâmetros foram obtidos a partir da caracterização de carga de uma aplicação real fortemente dependente de grandes

repositórios de dados: o Microscópio Virtual [Ferreira et alia (1997)]. Este estudo permite a análise dos efeitos desses algoritmos, que poderiam ser implementados e integrados a muitas das ferramentas de virtualização de dados existentes [Weng et alia (2004); Chervenak et alia (1999); Moore e Baru (2003)], para acrescentar o suporte à distribuição e replicação irregular de dados. As ferramentas apresentadas até então suportam apenas configurações regulares de distribuição e replicação de dados, assim commo bancos de dados distribuídos, de uma maneira geral.

A aplicação dessa arquitetura não se restringe apenas a virtualização de dados sobre grades de dados [Chervenak et alia (1999)]. É possível aplicar este conceito sobre qualquer sistema distribuído no qual os dados são replicados ou distribuídos de maneira desorganizada (sem uma noção global entre as fontes de dados) e que suporte um índice global de meta-dados. Exemplos de aplicações são índice para sistema de caches distribuídos e redes de sensores para disseminação de dados [Luo et alia (2005)]. O trabalho não considera operações de atualização dos dados do repositório, assumindo que os dados estão disponíveis somente para leitura. No entanto, não é complicado permitir atualizações e inserções de dados à base, como será explicado na Seção 3.1.

1.1 Estrutura da Dissertação

Cinco capítulos seguem esta introdução. O Capítulo 2 discute alguns trabalhos relacionados e contrasta suas contribuições com as apresentadas neste trabalho. O Capítulo 3 apresenta os conceitos relacionados a virtualização de dados, além do modelo proposto que permite a indexação de meta-dados para cenários de dados irregularmente replicados (Seção 3.1) e os algoritmos propostos para escalonamento de consultas (Seção 3.3). Em seguida, o Capítulo 4 discute as decisões de implementação e os resultados do estudo de caracterização de carga realizado. O Capítulo 5 apresenta os resultados obtidos pelas simulações e compara o desempenho dos algoritmos propostos ao desempenho de outros algoritmos básicos. Finalmente, o Capítulo 6 apresenta as conclusões do trabalho e direções futuras.

Capítulo 2

Trabalhos Relacionados

2.1 Acesso a Dados Através da Grade

Uma grade computacional é composta por vários computadores interconectados, formando um computador virtual capaz de distribuir processos entre seus componentes. Essa arquitetura é utilizada principalmente para solucionar problemas computacionais de larga escala. Nos últimos anos, alguns trabalhos foram desenvolvidos voltados para infra-estruturas em grade [The Globus Alliance (2005)] e soluções e aplicações baseadas na Grade computacional. Em [Chervenak et alia (1999)], os autores apresentaram uma arquitetura para gerenciamento de dados na Grade. Essa arquitetura é baseada nos Serviços para Dados em Grade (do inglês, *Grid Data Services* ou *GDS*), que são extensões dos conhecidos *Grid Services*. Para suportar acesso e integração de dados na Grade, um grupo inglês criou a *Open Grid Services Architecture — Data Access and Integration* (*OGSA-DAI*)
[Database Access and Integration Services Working Group (2005)]. Esta é baseada na arquitetura para grade de dados (do inglês, *data grid*) [Chervenak et alia (1999)] e serve como base para OGSA-DQP [Alpdemir et alia (2003)], em que os autores apresentam um arcabouço baseado em serviços que permite que consultas sejam submetidas para dados distribuídos em grade e disponíveis através dos *GDS*. [Watson (2005)] explorou OGSA-DQP no contexto de localidade e distribuição de dados. Todos esses trabalhos diferem do trabalho apresentado nesta dissertação por não atacarem o problema de replicação de dados, além de assumirem que os dados estão organizados em bancos de dados relacionais.

Em [Allcock et alia (2002)], os autores tratam o problema de replicação de dados, porém não lidam com a submissão de consultas a esses dados. Eles apresentam uma solução denominada GridFTP, que permite a transferência de dados entre nós de uma grade computacional. Além de não apresentarem suporte a consultas, os autores assumem que os dados estão replicados de forma regular. [Baer e Wyckoff (2004)] apresentaram uma otimização para o GridFTP que utiliza MPI-I/O para a comunicação na grade. Um arcabouço de escalonamento de tarefas em grade foi descrito por [Ranganathan e Foster (2002)]. Esse arcabouço desvincula a computação e a movimentação de dados da tarefa sendo escalonada e é capaz de mover os dados (criando replicação) para melhorar o desempenho da execução da tarefa.

A principal diferença entre os trabalhos de pesquisa em acesso de dados em grade e este trabalho é a não consideração de cenários de dados irregularmente replicados e a maioria dos estudos supõe que os dados estão organizados em bancos de dados relacionais e que são passíveis de realocação entre os nodos da Grade. A arquitetura discutida nesta dissertação suporta repositórios de dados irregularmente distribuídos e replicados e pode ser utilizada sobre quaisquer tipos de dados que permitam consultas por intervalos.

2.2 Bancos de Dados Distribuídos

Os trabalhos de pesquisa realizados sobre bancos de dados distribuídos também se relacionam com esse trabalho. Em sua maioria, os esforços nessa área vêm se concentrando no projeto de um sistema de gerência de bancos de dados (do inglês, *Data Base Management System* ou *DBMS*) baseado em aglomerados de computadores, com dados possivelmente replicados e potencial para processamento de consultas. [DeWitt e Gray (1992)] apresentaram técnicas utilizadas para implementar bancos de dados distribuídos (chamados de "bancos de dados paralelos").

O mais importante desafio relacionado a bancos de dados distribuídos é a habilidade de processar consultas de maneira distribuída. Em [Epstein et alia (1978); Yu e Chang (1984); Kossman (2000)], os autores apresentam a maior parte dos conceitos envolvidos no processo de processamento de consultas, dado que os dados estão distribuídos em bancos de dados distintos. Em [Haas et alia (1997)], os autores introduzem um *middleware* que otimiza consultas que deveriam ser servidas por fontes de dados distintas. O *middleware* permite que cada fonte de dados (servidor de banco de dados) tenha uma capacidade de atendimento de consultas diferente, e explora esta heterogeneidade no processamento das consultas. Em [Kemper e Wiesner (2001)], uma nova arquitetura é apresentada, baseada em híper-consultas (do inglês, *hyper queries*) para processar consultas distribuídas no contexto de comércio eletrônico entre empresas (do inglês, *business-to-business* ou *B2B*). [Kossman (2000)] apresentou um estudo que descreve várias técnicas de processamento distribuído de consultas, como técnicas de união (*join*), técnicas que exploram paralelismo intra-consulta, técnicas projetadas para diminuir os custos de comunicação, entre outras.

[Rahm e Marek (1995)] abordaram o problema de balanceamento de carga no contexto de sistemas de bancos de dados paralelos, nos quais a decisão de escalonamento de tarefas não é simples. O problema de gerenciamento de dados distribuídos é abordado em [Bartal et alia (1992)]. Os autores apresentam uma análise comparativa de algoritmos para gerenciar os dados em um ambiente distribuído, estudando em particular a replicação e alocação de arquivos. Em [Gribble et alia (2000)], os autores apresentam um conjunto de abstrações encapsuladas em uma camada com o objetivo de simplificar a construção de serviços de Internet baseados em aglomerados de computadores. Eles fazem uso de tabelas *hash* distribuídas (do inglês, *distributed hash tables* ou *DHT*) para gerenciar réplicas e executar os serviços.

Trabalhos que tratam de processamento distribuído de consultas se assemelham a este trabalho no sentido de que ambos provêem uma visão unificada dos dados sendo consultados

(virtualização dos dados). No entanto, este trabalho lida com repositórios de dados irregularmente distribuídos e replicados. Sistemas de bancos de dados distribuídos normalmente desencadeiam replicação de dados para favorecer tolerância a falhas devido à redundância e para alcançar melhores tempos de resposta no atendimento de consultas, como em [Amza et alia (2003); Soundararajan et alia (2005)]. Essa replicação, no entanto, acontece de maneira regular, seguindo critérios estabelecidos pelo usuário ou pelo próprio sistema de bancos de dados.

2.3 Virtualização de Dados

Algumas conquistas foram alcançadas em virtualização de dados [Weng et alia (2004); Narayanan et alia (2003); Moore e Baru (2003)], porém todas assumem cenários de dados regularmente distribuídos. Além disso, algumas das soluções propostas dependem de especificações internas de representação de dados. Este trabalho provê virtualização de dados, mas não apresenta qualquer restrição quanto à maneira na qual os dados se encontram distribuídos ou replicados.

[Sarawagi e Stonebraker (1994)] mostraram como acessar trechos de vetores como objetos em bancos de dados objeto-relacionais enquanto em [Stolte et alia (2002a,b)], os autores lidam com esses vetores e informações multi-dimensionais como cubos de dados. Um exemplo de banco de dados comercial que lida com dados multi-dimensionais pode ser encontrado em [Baumann et alia (1997)]. Este trabalho apresenta um modelo de indexação de dados que também suporta repositórios multi-dimensionais. No entanto, o modelo permite que o mapeamento entre os parâmetros que definem a base de dados e as dimensões do repositório seja especializado de acordo com a natureza dos dados armazenados.

Capítulo 3

Virtualização de Dados Irregularmente Replicados e Distribuídos

Algumas categorias de aplicações, principalmente aplicações científicas ou simuladores de fenômenos naturais, precisam acessar e analisar enormes volumes de dados. As comunidades científicas que demandam acesso a esses recursos são numerosas e geograficamente distribuídas, assim como os recursos de armazenamento e computação utilizados por essas comunidades [Moore et alia (1999)]. Os dados analisados por essas comunidades são volumosos o suficiente para que se torne inviável seu armazenamento em um único componente de hardware e, em muitos casos, em um único domínio administrativo. As grades computacionais apresentam uma capacidade de armazenamento virtualmente ilimitada e portanto são os principais ambientes utilizados para atender às demandas por grandes quantidades de dados. As grades computacionais, no entanto, têm a heterogeneidade de hardware e software como uma de suas características principais, o que torna um desafio o acesso a recursos de seus nodos.

A habilidade de organizar e gerenciar dados que estão distribuídos em diferentes localidades e domínios administrativos é provida por grades de dados, que se baseiam no conceito de virtualização de dados. A virtualização de dados é alcançada através da inserção de softwares de gerenciamento de dados e conversão de consultas entre as aplicações ou serviços de usuários e os repositórios de dados, onde se encontram os dados. O software de gerenciamento de dados provê identificadores persistentes únicos para a descoberta e acesso aos dados e operações padronizadas para interagir com os sistemas de armazenamento [Moore e Baru (2003)].

Os sistemas de virtualização até então propostos operam sobre um ambiente no qual os dados são organizados de acordo com um critério pré-definido. Os sistemas são ativos, no sentido de que têm a autonomia para disparar processos de replicação de dados, que buscam favorecer um critério de desempenho pré-estabelecido para o serviço dos dados. Esses sistemas buscam distribuir os dados de maneira ordenada, sendo assim possível determinar a localização de uma porção de dados através de uma função. Assim, os sistemas até então propostos lidam com configurações regulares de repositórios de dados.

Nesses repositórios, os dados são gerados de forma independente, sem uma noção global e portanto, o sistema não tem a autonomia para gerar replicação de dados entre as várias fontes

de dados que o compõe. Nesse caso, existe um senso de colaboração no sentido de que há o interesse por parte de uma das fontes de dados que outros usuários acessem os dados, mas a colaboração não necessariamente se estende ao ponto em que as fontes de dados compartilham seus recursos e confiam os dados ao sistema como um todo, permitindo que porções dos dados sejam removidas de seu repositório ou que outras porções de dados sejam acrescidas a ele.

[Chervenak et alia (1999)] discute os pré-requisitos, em diferentes níveis, para se projetar uma arquitetura de grade de dados. Em um cenário no qual se tem acesso irrestrito para replicar, remover e gravar novos dados nos nodos da grade, o trabalho apresenta as principais características dos serviços fundamentais necessários para a implementação de uma grade de dados. Alguns desses serviços se relacionam diretamente com o serviço de virtualização de dados, foco desta dissertação. Os dois serviços fundamentais para a implementação de uma grade de dados são: acesso a dados e acesso a meta-dados.

O serviço de acesso a dados deve prover mecanismos para acessar, gerenciar e iniciar transferências de dados presentes em sistemas de armazenamento. O serviço de acesso a meta-dados deve prover mecanismos para acessar e gerenciar informações sobre os dados presentes nos sistemas de armazenamento. A distinção explícita entre armazenamento e meta-dados em nível de arquitetura favorece a flexibilidade na implementação de sistemas de armazenamento enquanto minimiza o impacto em implementações que combinem acessos a meta-dados e acessos aos sistema de armazenamento. Permite, por exemplo, que o serviço de acesso a dados não seja afetado pela configuração irregular do repositório. Nesse caso, apenas o processamento de meta-dados seria afetado. O modelo de indexação de meta-dados proposto neste trabalho é apresentado na Seção 3.1.

[Chervenak et alia (1999)] apresenta gerenciamento e seleção de réplicas como os componentes mais significativos da camada de mais alto nível da arquitetura de uma grade de dados. O gerente de réplicas tem como papel criar ou remover cópias de instâncias de arquivos, ou réplicas, em determinados sistemas de armazenamento. Esse agente não se aplica ao cenário de distribuição de dados tratado nesta dissertação, no qual nenhum agente centralizado tem autoridade sobre os dados disponíveis na grade de dados, impedindo a criação e remoção de réplicas nos nodos da grade. O serviço de seleção de réplicas tem como papel escolher uma réplica dos dados que oferecerá a uma aplicação com características de acesso a dados que otimize algum critério de desempenho desejado, como tempo de resposta, custo ou segurança. Neste trabalho, a seleção de réplicas é tratada como escalonamento de consultas e é discutida na Seção 3.3.

3.1 Indexação de Meta-dados

Quando uma consulta é feita a um sistema de virtualização de dados o software de gerenciamento de dados, neste trabalho chamado de processador de consultas, deve ser capaz de identificar a que porção do repositório a consulta se refere, qual(is) servidor(es) (ou fontes de dados) possui(em) toda ou parte do conteúdo requisitado, além de informações sobre os servidores como mecanismos de armazenamento utilizados e capacidade de servir os dados

em termos dos critérios de desempenho. Neste trabalho, os meta-dados são definidos como toda informação sobre os servidores, instâncias de repositórios e tipos de mecanismos de armazenamento.

No entanto, não é viável que todas essas informações sejam adquiridas pelo processador de consultas sob demanda, a cada consulta recebida. Os principais fatores que tornam essa prática inviável são desempenho e limitação de recursos de rede. Assim, é necessário que todas essas informações sejam adquiridas previamente pelo processador de consultas. Ainda assim, essas informações precisariam ser processadas todas as vezes que uma nova consulta fosse feita ao repositório de dados. Por esse motivo, os softwares de processamento de consultas normalmente realizam de antemão todo o processamento de meta-dados, sumarizando-os para agilizar o processamento das consultas por dados. A este pré-processamento dos meta-dados dá-se o nome de indexação.

Na inicialização do sistema explorado neste trabalho, é necessário que todas as fontes de dados enviem seus meta-dados, que consistem em informações sobre os limites de cada intervalo de dados disponível na fonte de dados e informações sobre a capacidade do nodo para prover dados. A única restrição sobre a métrica utilizada para expressar tal capacidade para provimento de dados é que todas as fontes de dados devem utilizar a mesma métrica em seus meta-dados. Esta métrica deve representar um gargalo do sistema, como por exemplo: banda da conexão de rede, tempo de acesso a dispositivos de armazenamento de dados (como arranjos de discos rígidos), tempo de processamento de uma consulta por um mecanismo de armazenamento de dados (como banco de dados), etc. Nos experimentos realizados neste trabalho, a métrica utilizada foi largura de banda de rede para transmissão dos dados.

A Figura 3.1 esquematiza, de maneira geral, o processo de indexação dos meta-dados para a criação do índice global. Em um primeiro momento, todas as fontes de dados enviam seus meta-dados para o processador de consultas. Não há qualquer restrição quanto ao protocolo de comunicação entre as fontes de dados e o processador de consultas, desde que haja garantia de entrega dos dados enviados na ausência de falhas de dispositivos.

O processador de consultas deverá ser capaz de identificar intervalos de dados replicados ao longo de todo o repositório, ainda que a replicação e distribuição desses dados ocorra de maneira irregular. Para isto, este trabalho propõe uma modelagem geométrica dos meta-dados para fins de indexação e processamento de consultas por intervalos de dados. Uma explicação detalhada desta modelagem e a computação associada é apresentada mais adiante nesta seção. A transformação de meta-dados em sua representação geométrica é o que ocorre no segundo passo esquematizado na Figura 3.1.

O terceiro passo consiste na realização de uma série de operações geométricas sobre as representações dos meta-dados de cada um dos intervalos de dados. Dentre as operações geométricas estão operações de verificação, como intersecção e subtração de poliedros, e uma operação de decomposição de poliedros em sub-poliedros retos, que será descrita em detalhes posteriormente. O resultado desse processo é um conjunto de poliedros maximais disjuntos, que unidos representam todo o repositório de dados. Cada um desses poliedros representa um intervalo de dados e possui uma lista de nodos que servem esses dados.

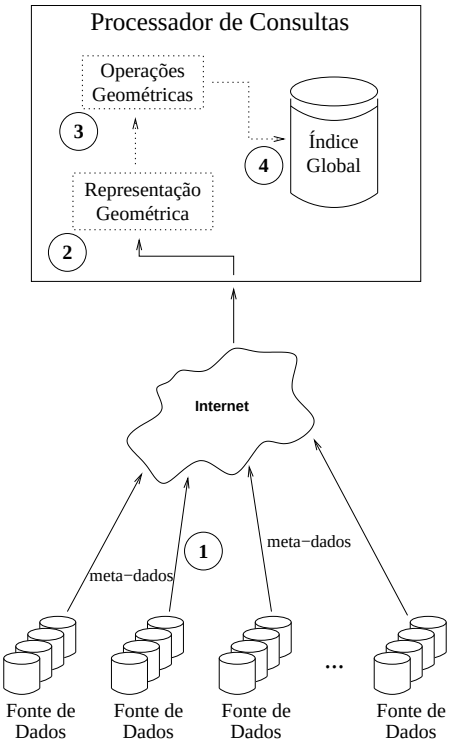


Figura 3.1: **Processo de criação do índice global.** As fontes de dados enviam informações sobre os intervalos de dados disponíveis. Os meta-dados são pré-processados pelo processador de consultas para a criação de um índice global.

Para que as operações geométricas sejam realizadas entre os vários poliedros e para que os poliedros resultantes possam ser unidos para representar todo o repositório, é necessário que exista um mapeamento entre os parâmetros ou atributos que definem os dados e dimensões espaciais. Esta tarefa é trivial quando os parâmetros do repositório se tratam de coordenadas espaciais, como no caso de imagens ou bases de dados de geoprocessamento. Repositórios de dados definidos através de atributos, como em bancos de dados relacionais, fazem dessa uma tarefa mais complexa que requer um estudo mais aprofundado.

Para melhor detalhar o processo de indexação dos meta-dados, podemos supor um repositório de dados formado por apenas 3 servidores, ou fontes de dados, A, B e C. Esses servidores enviam para o processador de consultas informações sobre sua capacidade de prover dados. Além disto, cada servidor informa os limites dos intervalos de dados que hospedam. Neste exemplo, dois parâmetros definem os limites de um intervalo de dados. O processador de consultas mapeia esses limites em coordenadas espaciais e representa cada um dos intervalos de dados como um poliedro, como ilustrado na Figura 3.2. Como no exemplo o espaço de dados é bi-dimensional, os intervalos de dados são representados como polígonos.

A primeira rodada de operações geométricas a ser realizada é a decomposição de todos os poliedros não retos no menor número possível de poliedros retos que unidos compõem o poliedro original. No caso ilustrado, o polígono que representa o intervalo de dados hospedado pelo servidor C seria decomposto em dois polígonos retângulos menores. Essa rodada é

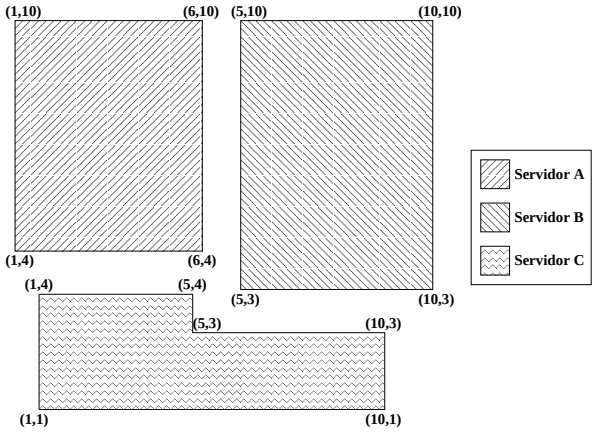


Figura 3.2: **Representação geométrica de intervalos de dados.** O processador de consultas representa os intervalos de dados como poliedros.

realizada para permitir algumas simplificações no algoritmo de decomposição de poliedros.

Neste trabalho, assume-se que todos os poliedros – e por consequência todos os intervalos de dados – são retos, ou seja, apresentam apenas ângulos retos entre lados adjacentes. Esta premissa é assumida para simplificar os algoritmos geométricos utilizados na implementação, e não é fruto de limitação do modelo de indexação e processamento de consultas propostos neste trabalho. O impacto dessa simplificação no âmbito da aplicação do sistema é que os intervalos de dados sejam sempre limitados por pares de valores atribuídos a parâmetros ou atributos que deverão ser ortogonais em seu mapeamento para coordenadas espaciais.

Na próxima rodada, todos os poliedros são tomados aos pares. Se dois poliedros possuem uma interseção não nula, a porção comum a ambos é subtraída e as partes restantes são submetidas à decomposição em poliedros retos. A porção comum se torna um novo poliedro (reto) a ser tratado e a lista de servidores dos dois poliedros originais é combinada para criar a lista de servidores da porção comum. O novo poliedro também é tomado par-a-par com os demais, com exceção do par que o originou. No repositório de exemplo, os polígonos relacionados aos intervalos de dados dos servidores A e B se intersectariam e a intersecção seria subtraída dos polígonos originais. O resultado de $B - (A \cap B)$ não seria um retângulo e teria então que ser decomposto.

Essa operação ocorre até que nenhum par de poliedros se intersecte. O resultado é um conjunto de poliedros retos maximais disjuntos, que serão daqui em diante referidos como *blocos de dados*. Cada bloco de dados representa um intervalo de dados hospedado por um grupo de fontes de dados. Um bloco de dados associado a múltiplos servidores é um indício de replicação no repositório. O conjunto de todos os blocos de dados constitui o índice global, exemplificado na Figura 3.3. A união de todos os poliedros retos maximais disjuntos mapeados no mesmo espaço de coordenadas forma a representação geométrica de todo o repositório de dados disponível através do sistema de virtualização.

Através da Figura 3.3 é possível observar que o polígono referente ao intervalo de dados do servidor C não intersectou nenhum dos demais, mas ainda assim foi decomposto em dois

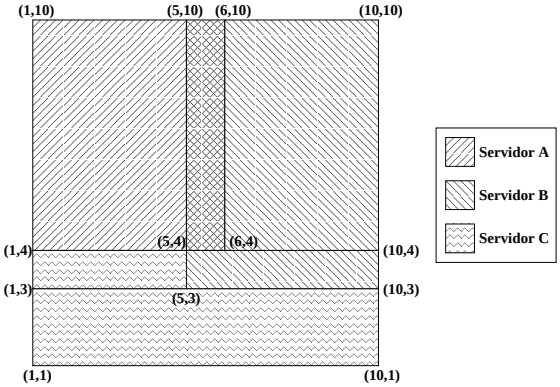


Figura 3.3: **Índice global.** O índice global compreende um conjunto de blocos de dados, cada um com uma lista de servidores associados.

retângulos na primeira rodada do processo de indexação. Na rodada seguinte, a intersecção dos polígonos referentes aos intervalos de dados dos servidores A e B gerou a criação de um novo bloco de dados, com ambos os servidores em sua lista de fontes de dados. O que restou do polígono referente ao servidor B foi então decomposto em dois retângulos.

Apesar de neste trabalho os dados terem sido considerados disponíveis apenas para leitura, não há qualquer restrição que impeça que novos intervalos de dados sejam inseridos no sistema ou que atualizações sejam feitas sobre os dados já presentes no índice global. No caso de atualizações de dados, seria necessário remover do índice global todas as entradas referentes ao servidor que hospeda os dados atualizados. Em seguida, os meta-dados devem ser enviados novamente ao processador de consultas, que deve reinserir essas informações no índice global através da execução dos algoritmos geométricos descritos nesta seção. No caso de inserção de dados, é necessário apenas que os meta-dados que descrevem o novo intervalo de dados sejam integrados ao índice global. No caso de um índice não centralizado, técnicas convencionais de sincronização de índices distribuídos devem ser aplicadas [Özsu e Valduriez (1999)].

Uma vez criado o índice global, o processador de consultas está apto a receber consultas por intervalos de dados e desmembrá-las em múltiplas consultas menores a serem distribuídas para as várias fontes de dados que compõem o sistema. O algoritmo de processamento de consultas é muito semelhante ao de indexação de meta-dados e é descrito na Seção 3.2.

3.2 Processamento de Consultas

Uma vez que o índice global esteja completo, o processador de consultas está pronto para receber consultas. O processamento de consultas é muito semelhante ao processo de indexação de meta-dados apresentado. A Figura 3.4 esquematiza o processamento e escalonamento de consultas.

Inicialmente, os clientes enviam consultas por intervalos de dados para o processador de consultas. Não há qualquer restrição quanto ao protocolo de comunicação entre as aplicações clientes e o processador de consultas, desde que haja uma garantia de entrega das requisições enviadas. Ao receber a consulta por intervalo de dados, o processador de consultas utiliza o

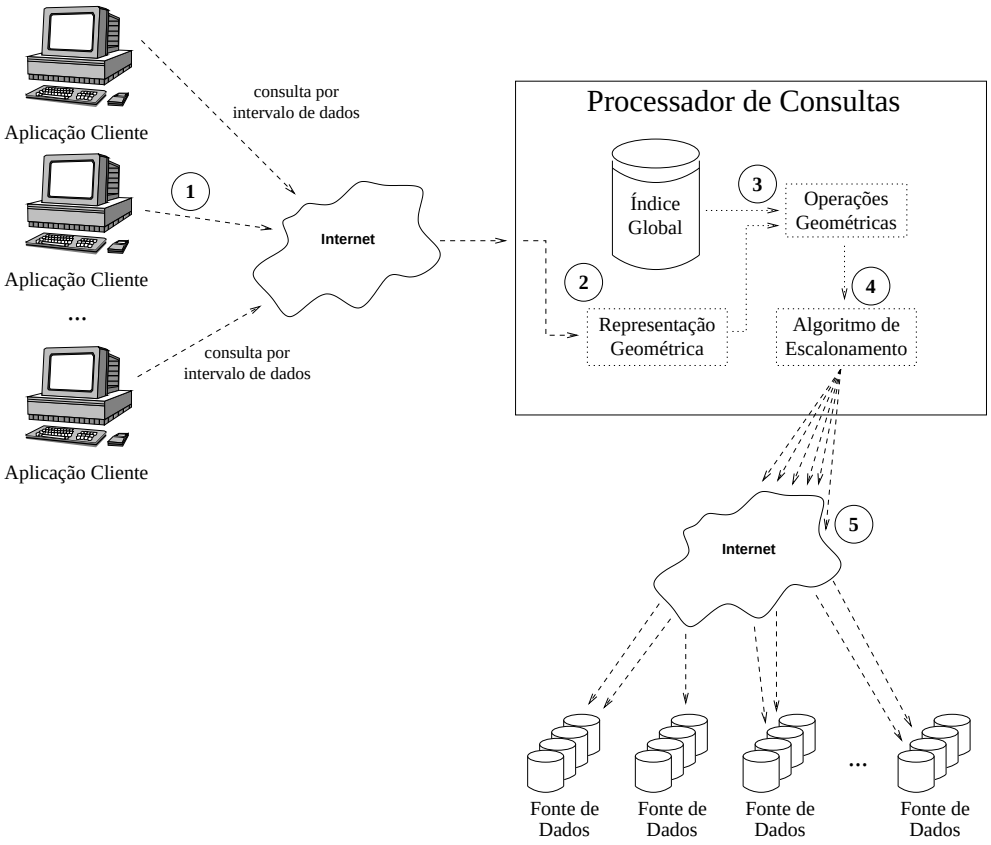


Figura 3.4: **Processamento e escalonamento de consultas.** As consultas são representadas geometricamente e processadas para identificar o conjunto de fontes de dados aptas a prover os dados requisitados.

mesmo critério de mapeamento para coordenadas espaciais utilizado no processo de indexação e gera uma representação geométrica da consulta por intervalos (etapa 2 da Figura 3.4). Essa representação geométrica pode resultar em múltiplos poliedros, caso os intervalos de dados requisitados não sejam adjacentes.

A implementação apresentada neste trabalho assume a premissa de que todas as consultas por intervalos de dados resultam em poliedros retos quando representadas geometricamente, assim como para os intervalos de dados disponíveis nas fontes de dados.

A Figura 3.5 ilustra uma consulta por intervalo de dados no repositório suposto como exemplo na Seção 3.1. Na figura, é realizada uma consulta pelo intervalo de dados que, quando mapeado para coordenadas espaciais, é limitado pelos valores 2 e 7 em ambas as dimensões (atributos ou parâmetros do repositório de dados).

A terceira etapa do processamento de consultas (Figura 3.4) consiste em posicionar a representação geométrica da consulta no espaço de coordenadas, juntamente com todos os blocos de dados que formam o índice global. Então, os poliedros retos que representam a consulta são fragmentados em suas intersecções com os blocos de dados. A cada poliedro resultante da intersecção da consulta com um bloco de dados é associada a lista de fontes de dados referentes ao bloco de dados. O resultado dessa etapa é um conjunto de fragmentos retos (maximais

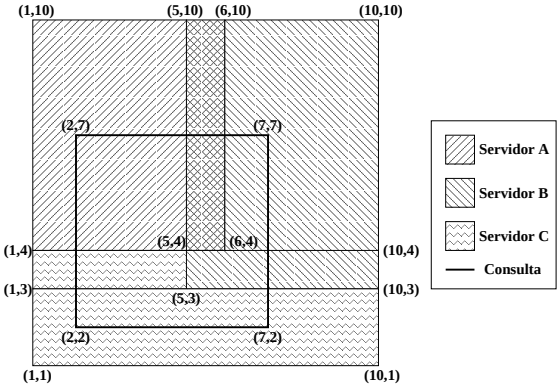


Figura 3.5: **Representação geométrica da consulta.** A consulta por intervalos de dados é mapeada para o mesmo espaço de coordenadas espaciais dos blocos de dados presentes no índice global.

e disjuntos) que unidos formam a representação geométrica da consulta. A cada um desses fragmentos está associada a lista de fontes de dados candidatas para prover aquela porção dos dados requisitados. Esse conjunto de poliedros retos representa os fragmentos disjuntos e maximais da consulta que podem ser repassados, em grupos ou separadamente, para as fontes de dados que hospedam os intervalos de dados requisitados. Esses fragmentos resultantes são tratados neste trabalho como *fragmentos da consulta*. A Figura 3.6 esquematiza esse resultado para o suposto repositório.

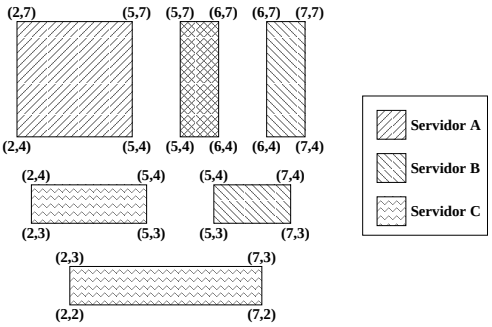


Figura 3.6: **Fragmentos da consulta.** O resultado do processo é um conjunto de fragmentos da consulta que podem ser repassados individualmente às fontes de dados.

Na Figura 3.6 é possível observar que um dos seis fragmentos da consulta possui mais de uma fonte de dados em sua lista (servidores A e B). Como já mencionado, esse fenômeno indica a presença de replicação de intervalos de dados dentre as fontes de dados. O sistema pode se beneficiar dessa informação para buscar um melhor desempenho no provimento de dados para as aplicações clientes. De posse de uma lista de fontes de dados aptas a atender a uma requisição por intervalo de dados e com informações instantâneas sobre a capacidade de cada um desses servidores prover os dados requisitados, é possível escalonar os fragmentos da consulta de forma a favorecer o desempenho, de acordo com uma métrica pré-estabelecida. O escalonamento dos fragmentos da consulta entre os servidores candidatos finaliza o proces-

samento de consultas, como mostra a Figura 3.4. A Seção 3.3 propõe alguns mecanismos de escalonamento para esse fim.

Todas as discussões e resultados apresentados neste trabalho se referem a abstrações bidimensionais dos intervalos de dados e consultas, como em uma porção de uma imagem visualizada utilizando o Microscópio Virtual [Ferreira et alia (1997)]. Algumas aplicações podem demandar mais dimensões para representar seus meta-dados e consultas. Os algoritmos utilizados podem ser facilmente generalizados para lidar com mais de duas dimensões. Naturalmente, adicionar dimensões ao processamento de consultas e indexação de meta-dados proporcionará um impacto negativo no tempo de execução dos algoritmos. O tempo de execução de ambos os processos é da ordem de $O(p^2v^2)$, para p representando o número de poliedros e v representando o número médio de vértices por poliedro. O aumento no número de dimensões acarreta um aumento significativo de v. Mais especificamente no caso da implementação utilizada neste trabalho (que se limita a polígonos retos), o aumento de dimensões dobra v.

3.3 Escalonamento de Consultas

Esta seção apresenta os algoritmos desenvolvidos para guiar o processador de consultas na escolha de qual fonte de dados será responsável por servir cada fragmento da consulta, gerado através de operações geométricas entre o índice global e a consulta por intervalos de dados. Naturalmente, os algoritmos de escalonamento de consultas apenas se aplicam aos fragmentos de consulta que estão replicados entre os servidores e, conseqüentemente, podem ser servidos por mais de um servidor.

Para fins de comparação, os seguintes algoritmos básicos [Diniz et alia (2006)] serão utilizados:

Aleatório (AL): seleciona aleatoriamente o servidor responsável por atender a um fragmento de consulta. As principais limitações desse algoritmo são não considerar a carga imposta aos servidores em qualquer instante e não considerar a capacidade de servir dados de cada servidor. Esse algoritmo falha em manter um balanceamento de carga quando os servidores são heterogêneos.

***Round-Robin* Ponderado (RRP):** cada servidor possui uma fila de requisições, e as requisições são distribuídas proporcionalmente entre os servidores, de acordo com sua capacidade de atender consultas. Esse algoritmo resulta em um melhor balanceamento de carga uma vez que servidores com maior capacidade tendem a ser mais freqüentemente selecionados. Apesar desse algoritmo considerar a capacidade de cada servidor, a carga instantânea imposta aos servidores não é levada em consideração.

É interessante considerar a capacidade dos servidores no escalonamento de consultas. O motivo para isto é que se espera que uma consulta seja atendida mais rapidamente por um servidor com maior capacidade de prover dados. No entanto, por maior que seja a capacidade de um servidor, a carga imposta a ele em um dado instante pode ser grande o suficiente para

que o seu desempenho no atendimento de consultas seja inferior ao desempenho que teria o menos capaz dentre os servidores. Com esta motivação, este trabalho propõe dois algoritmos que buscam, através de diferentes abordagens, um compromisso entre a capacidade e a carga instantânea dos servidores para melhor escalonar as consultas por intervalos de dados.

Para que os algoritmos de escalonamento possam considerar a carga instantânea imposta aos servidores, esse conhecimento deve chegar até o processador de consultas. Um protocolo simples de notificação uni-direcional é capaz de suprir essa necessidade do sistema. Sempre que um servidor acabar de servir um fragmento de uma consulta, ele deve notificar o processador de consultas do seu novo número de fragmentos de consultas pendentes. O número de fragmentos pendentes em cada servidor, assim como o conhecimento prévio da capacidade de serviço dos servidores, permite ao processador de consultas calcular a capacidade de serviço disponível em cada um dos servidores. A métrica de capacidade disponível $c(s,t)$ de um servidor s no instante t é definida como:

$$c(s,t) = \frac{C(s)}{R(s,t)+1} \tag{3.1}$$

onde $C(s)$ é a capacidade absoluta (ou, não instantânea) do servidor s e $R(s,t)$ é a quantidade de requisições pendentes no servidor s no instante t. Essa métrica é utilizada pelos algoritmos que são apresentados a seguir para representar a carga instantânea imposta a cada servidor no sistema. Ao se utilizar esta métrica, assume-se que a capacidade absoluta de um servidor é compartilhada igualmente no serviço de cada requisição pendente. Por exemplo, um servidor com capacidade para prover dados a 100 Mbps, alocaria 20 Mbps para cada um dos 5 fragmentos de consulta a ele submetidos.

3.3.1 Escalonamento Baseado em Heurística Gulosa

O algoritmo de escalonamento baseado em heurística gulosa (GUL) utiliza uma lógica simples para tentar garantir a alocação dos maiores fragmentos de consulta para os servidores em melhores condições para provê-los. Ao contrário dos dois algoritmos já citados, esse algoritmo considera não só a capacidade dos servidores para prover os dados como também a carga instantânea imposta a cada servidor. Para tal, o algoritmo considera a capacidade disponível dos servidores, como explicado acima.

O algoritmo recebe como entrada os fragmentos da consulta a ser escalonada, assim como a capacidade disponível em cada servidor candidato para prover os fragmentos da consulta. Com base nessas informações, o algoritmo segue os seguintes passos:

1. É criada uma lista com os fragmentos da consulta. Essa lista é ordenada na ordem decrescente da quantidade de dados requisitados em cada fragmento da consulta;

2. Uma outra lista é criada, com os servidores candidatos a prover os fragmentos da consulta. Esta lista é ordenada em ordem decrescente de capacidade disponível por cada servidor;

3. A lista de fragmentos da consulta é percorrida, começando pelo maior fragmento. Para cada fragmento da lista:

    a) A lista de servidores candidatos é percorrida, começando pelo servidor com maior capacidade disponível. A varredura acontece até que o primeiro servidor que hospeda o atual fragmento da consulta seja encontrado;

    b) O fragmento da consulta é atribuído ao servidor;

    c) A capacidade disponível do servidor é recalculada e a lista de servidores candidatos é reordenada, para ser mantida na ordem decrescente de capacidade disponível;

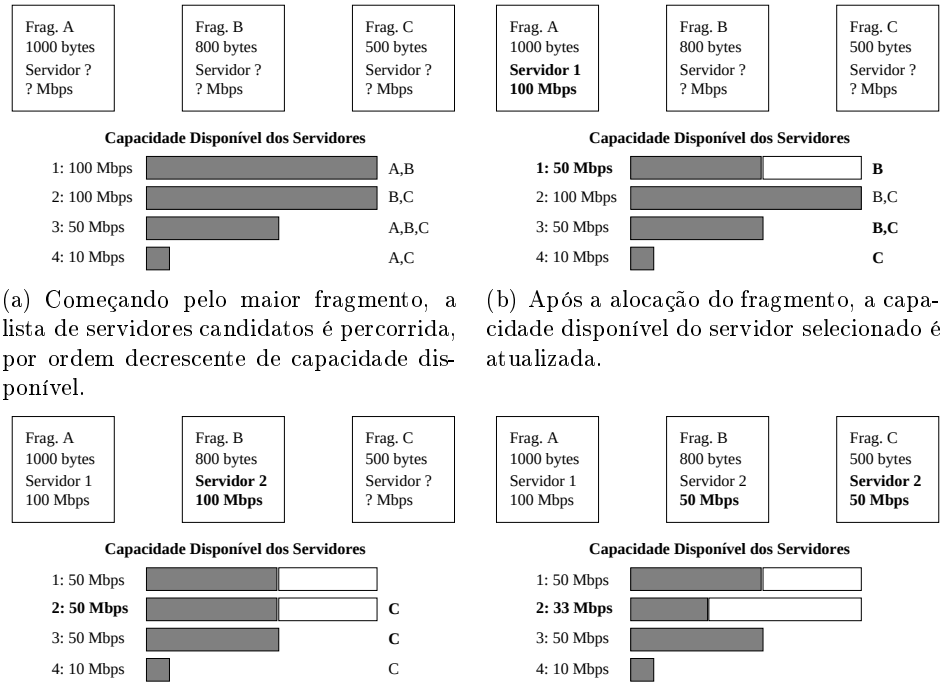
    d) O fragmento da consulta é escalonado e removido da lista.

O objetivo do algoritmo é fazer com que o maior fragmento da consulta seja escalonado para o mais rápido entre os servidores candidatos. O raciocínio por trás desse objetivo é que todos os fragmentos de consulta são servidos simultaneamente e a consulta só é atendida por completo quando o último byte do último fragmento pendente é servido. Quanto maior o fragmento da consulta, maior a chance desse ser o último fragmento pendente e portanto, melhor seria se esse fragmento fosse escalonado para ao melhor servidor candidato, em termos de capacidade. O mesmo raciocínio se aplica ao segundo maior fragmento da consulta e assim sucessivamente.

A Figura 3.7 ilustra as decisões tomadas pelo algoritmo de escalonamento para um repositório hipotético. Os dados de entrada para o algoritmo podem ser vistos na Figura 3.7(a). Nesse repositório hipotético, a consulta foi dividida em 3 fragmentos com tamanhos diferentes. Os servidores 1, 3 e 4 são candidatos para prover os dados requisitados no fragmento A da consulta, enquanto para o fragmento B, os servidores candidatos são 1, 2 e 3 e para o fragmento C, servidores 2, 3 e 4. Antes de qualquer fragmento da consulta ser escalonado, a capacidade disponível dos servidores eram: servidores 1 e 2, 100 Mbps; servidor 3, 50 Mbps e servidor 4, 10 Mbps. Supõe-se que os servidores não têm fragmentos de consultas pendentes no estado inicial.

Começando pelo fragmento A, maior fragmento da consulta, o algoritmo busca pelo servidor candidato com maior capacidade disponível para servir o fragmento. A Figura 3.7(b) mostra que o fragmento A seria então escalonado para o servidor 1. Após a seleção do servidor, a informação de capacidade disponível do servidor 1 é atualizada. É importante reforçar que até este instante, o fragmento A seria servido pelo servidor 1 a 100 Mbps, a capacidade anteriormente disponível. O novo valor de capacidade disponível se refere a qual seria a taxa de serviço, por fragmento, caso um novo fragmento da consulta seja alocado para o servidor 1.

Em seguida (Figura 3.7(c)), o algoritmo segue para o segundo maior fragmento, o fragmento B, para o qual a melhor opção de escalonamento é o servidor 2. Uma vez escalonado, a informação de capacidade disponível do servidor 2 também é atualizada.



(a) Começando pelo maior fragmento, a lista de servidores candidatos é percorrida, por ordem decrescente de capacidade disponível.

(b) Após a alocação do fragmento, a capacidade disponível do servidor selecionado é atualizada.

(c) O servidor 2 foi a melhor opção (100 Mbps) para a alocação do fragmento B.

(d) Os servidores 2 e 3 apresentavam a mesma capacidade disponível para servir o fragmento C. O primeiro servidor da lista foi selecionado.

Figura 3.7: Ilustração das decisões locais do algoritmo de escalonamento guloso.

A Figura 3.7(d) mostra que o algoritmo aloca o último fragmento, C, também para o servidor 2. Com esta alocação, a capacidade disponível do servidor é atualizada, assim como a taxa com a qual o fragmento B seria servido.

Com esta alocação de servidores, o fragmento A da consulta seria atendido em 80 microsegundos (a 100 Mbps, desconsiderando atrasos de rede). O fragmento B seria servido a 50 Mbps juntamente com o fragmento C, até que o menor entre estes terminasse. Neste caso, o fragmento C é servido em 80 microsegundos. Após o fim do fragmento C, o restante do fragmento B (300 bytes) seria servido à taxa de 100 Mbps, em 24 microsegundos. Neste caso, o tempo de atendimento do fragmento B define o tempo de resposta da consulta, 104 microsegundos.

A abordagem gulosa toma decisões baseadas em informações imediatas, buscando sempre uma boa solução local. Não há uma visão global da solução. No repositório hipotético, se o fragmento C fosse alocado ao servidor 3, o tempo de atendimento da consulta seria de 80 microsegundos, uma vez que o fragmento B passaria a ser atendido com exclusividade pelo servidor 2 (a 100 Mbps), sendo servido em 64 microsegundos. Isto faria com que o fragmento A determinasse o tempo de resposta total da consulta, que seria de 80 microsegundos. A escolha do melhor servidor para um fragmento pode comprometer a qualidade da escolha feita para o fragmento anterior.

No entanto, uma vantagem do algoritmo baseado na abordagem gulosa é o seu baixo custo computacional. O tempo de execução do algoritmo segue $O(mn)$, com m igual ao número

de fragmentos da consulta e n igual ao número de servidores candidatos a servir a consulta. Com isso, a cobertura da replicação dos dados (número de servidores nos quais os dados estão replicados) tem um impacto linear no tempo de execução do algoritmo de escalonamento.

3.3.2 Escalonamento Baseado em Recozimento Simulado

Em contraste com a abordagem heurística explorada pelo algoritmo guloso descrito na Seção 3.3.1, buscou-se uma solução baseada nos princípios de otimização com objetivo de encontrar o escalonamento ótimo para os fragmentos das consultas. A técnica inicialmente explorada foi a programação linear, que define um modelo matemático de otimização baseado em um sistema de equações lineares. Apesar de computacionalmente mais cara, esperava-se que a qualidade da solução obtida por essa técnica compensasse o tempo adicional de computação da solução.

As seguintes variáveis foram definidas para o modelo linear, além das apresentadas na Equação 3.1:

F: o conjunto de fragmentos da consulta;

S: o conjunto de servidores do sistema;

S_f: o conjunto de servidores candidatos a servir o fragmento f da consulta, subconjunto de S;

Tam_f: o tamanho do fragmento f da consulta;

x_{if}: assume 0 ou 1, indica se o fragmento f da consulta será requisitado ao servidor i;

s_i: assume 0 ou 1, indica se o servidor i participa do conjunto de servidores solução.

A equação a seguir define a função do tempo de resposta total do sistema (após o provimento de todas as consultas):

$$T(F,S) = \sum_{f \in F} \sum_{i \in S_f} \frac{x_{if} \times Tam_f}{c(i,t)} \tag{3.2}$$

A equação a seguir define a função que quantifica o paralelismo do sistema, ou seja, a quantidade de servidores participantes no escalonamento dos fragmentos das consultas:

$$P(S) = \sum_{i \in S} s_i \tag{3.3}$$

Finalmente, a função objetivo do modelo busca um menor tempo de resposta no provimento das consultas (minimizando o resultado da Equação 3.2) e uma maior distribuição do escalonamento, para evitar a concentração da carga nos servidores mais capazes (maximizando o resultado da Equação 3.3):

$$O(F,S) = minimizar\,(T(F,S) - P(S)) \tag{3.4}$$

Uma análise aprofundada da função objetivo 3.4, no entanto, revelou que à medida em que os fragmentos de uma consulta eram escalonados para algum servidor, a capacidade disponível desse servidor era alterada, o que afetaria a decisão de escalonamento dos fragmentos restantes da consulta. De fato, a capacidade disponível (Equação 3.1) de um servidor só pode ser definida em um determinado instante. Essa característica classifica o problema de escalonamento de fragmentos de consultas no sistema de virtualização explorado como não linear, o que inviabiliza a simples utilização da técnica de programação linear. Com isto, optou-se por utilizar uma técnica de otimização que utiliza uma abordagem heurística no intuito de escapar de mínimos locais.

O princípio do recozimento simulado (do inglês, *simulated annealing*), é uma analogia entre um processo de fundição e a solução de um problema de otimização combinatória, proposto por [Kirkpatrick et alia (1983)]. O termo *annealing* refere-se a um processo térmico que começa pela liquidificação de um metal a uma alta temperatura, seguido pela lenta e gradativa diminuição de sua temperatura, até que o ponto de solidificação seja atingido, quando o sistema atinge um estado de energia mínima. O resfriamento rápido conduz a produtos meta-estáveis, de maior energia interna, enquanto o resfriamento lento conduz a produtos mais estáveis, estruturalmente mais fortes por terem menor energia interna.

Na analogia, os estados possíveis de um metal correspondem a soluções do espaço de busca, a energia em cada estado corresponde ao valor da função objetivo e a energia mínima corresponde ao valor da solução ótima. A cada iteração do método, um novo estado é gerado a partir do estado corrente, através de uma alteração aleatória neste. Se o novo estado possui energia menor que o estado corrente, o novo estado passa a ser o estado corrente. Caso o novo estado possua uma energia maior que o estado corrente, com uma diferença de Δ unidades, a probabilidade do novo estado se tornar o estado corrente segue a função de aceitação $P = e^{\Delta/kT}$; sendo k a constante de Boltzmann e T a temperatura corrente do sistema. Este procedimento é repetido até se atingir o equilíbrio térmico. Quando o equilíbrio térmico é atingido a uma dada temperatura, essa é diminuída e uma nova iteração do método é iniciada. O processo como um todo termina quando a temperatura se aproxima de zero.

Dessa maneira, quanto maior a temperatura do sistema, maior a probabilidade de aceitação para uma transição de maior energia e, a altas temperaturas, cada estado tem praticamente a mesma chance de se tornar o estado corrente. A possibilidade de uma transição que piore a solução permite que o processo escape de ótimos locais. Como a cada iteração a temperatura é diminuída, no final do processo, praticamente não se aceita transições de piora.

A aplicação desse método em um problema de otimização demanda a configuração de um conjunto de parâmetros de controle, que dependem sensivelmente do problema e só podem ser determinados empiricamente. Os principais parâmetros são: temperatura inicial, que determina o quão aceitável é uma transição de piora da solução no início do processo; e o tamanho do decréscimo da temperatura a cada iteração, que determina a insistência do processo e quantos estados estáveis serão alcançados até que se determine qual o estado de energia mínima.

Para o problema de escalonamento tratado neste trabalho, um estado do sistema corres-

ponde a uma opção de escalonamento dos fragmentos da consulta a um sub-conjunto dos servidores candidatos. Uma transição entre estados corresponde à troca de escalonamento de um fragmento da consulta. A energia de um estado, grandeza que se busca minimizar, é determinada pela função apresentada na Equação 3.5.

$$E(d) = \frac{\sum T(f)}{n} \tag{3.5}$$

A função energia E, recebe um estado d como parâmetro. O estado contém informações sobre o tamanho dos fragmentos da consulta, o servidor escalonado para servir cada fragmento e a capacidade disponível em cada servidor. A função T retorna o tempo de resposta previsto para o fragmento f conforme seu escalonamento corrente e a capacidade disponível no servidor para o qual está escalonado. n corresponde ao número de servidores diferentes que participam do escalonamento no estado d do sistema. O esforço de minimização da função energia favorece um escalonamento mais distribuído entre os servidores do sistema e um menor tempo de resposta para servir os fragmentos.

Antes da primeira iteração do método de recozimento simulado (RS), é necessário definir um estado inicial para o sistema. Para tal, é realizado um escalonamento aleatório dos fragmentos da consulta. Para calcular a energia de um determinado estado do sistema, soma-se o tempo de resposta esperado para cada fragmento da consulta, devido ao escalonamento atual e à capacidade disponível de cada servidor envolvido no escalonamento. Desta maneira, a solução ótima é a que implica no menor tempo de resposta possível para cada um dos fragmentos da consulta.

Para uma transição entre estados do sistema, um fragmento da consulta é escolhido de maneira aleatória e, dentre os servidores candidatos, é selecionado outro servidor que não o atual responsável por atender ao fragmento da consulta. Como consequência da troca de escalonamento, as capacidades disponíveis dos dois servidores envolvidos na operação são atualizadas.

A abordagem baseada em recozimento simulado é uma estratégia de otimização de simples implementação e, dependendo da qualidade de configuração dos parâmetros, atinge resultados satisfatórios. No entanto, por se tratar de um mecanismo que depende de assumir diversos estados na busca de um potencial ótimo global, apresenta um alto custo computacional que, em alguns casos, pode ser desvantajoso ainda que a solução ótima seja encontrada.

Capítulo 4

Implementação e Configuração das Simulações

Este capítulo discute algumas decisões de implementação do modelo de indexação de meta-dados e fragmentação de consultas e dos algoritmos de seleção de servidores. Os resultados deste trabalho foram obtidos através de simulações e portanto buscou-se gerar, ainda que sinteticamente, configurações genuínas de servidores, cenários de dados e cargas de trabalho para as simulações. Este capítulo também discute a configuração do simulador de acordo com a caracterização da carga de trabalho realizada.

4.1 Detalhes de Implementação

O indexador de meta-dados e o processador de consultas foram implementados na linguagem C. Para tal, algumas adaptações foram realizadas sobre a biblioteca desenvolvida por [Murta (2005)], que disponibiliza um tipo abstrado de dados para polígonos assim como os algoritmos geométricos mais básicos, como intersecção de polígonos. O algoritmo de decomposição de polígonos em retângulos foi incorporado a essa biblioteca.

Em sua versão original, essa biblioteca não provê recursos para identificação única de polígonos, e portanto o tipo abstrado de dados que representa os polígonos foi alterado. Um mapa de bits foi associado a cada instância de um polígono, contendo o seu identificador único em todo o sistema. Como o número de polígonos em todo o sistema pode ser alto, foi necessário implementar um mapa de bits com suporte a um tamanho arbitrário, ainda que eficiente em operações como intersecção de mapas. Para tanto, uma biblioteca [Blustein (1995)] foi adaptada.

Com a utilização desta implementação de mapa de bits, foi possível identificar a partir de que conjunto de polígonos um outro determinado polígono foi gerado, a qualquer instante durante a execução dos algoritmos. Este recurso permitiu, por exemplo, que se evitasse calcular a intersecção de um par de polígonos múltiplas vezes durante os processos de indexação de meta-dados e processamento de consultas.

Os algoritmos de escalonamento de consultas foram desenvolvidos como módulos de um

simulador baseado em eventos, implementado na linguagem C++. Para a implementação dos algoritmos baseados na abstração de recozimento simulado, foi utilizada uma biblioteca científica [Galassi M. et al. (2005)].

O algoritmo de decomposição de polígonos acrescentado à biblioteca de operações geométricas [Murta (2005)] limita o simulador ao processamento de consultas por intervalos com representação através de polígonos côncavos com ângulos retos entre vértices adjacentes. No entanto, a implementação suporta que os polígonos tenham buracos, desde que a delimitação interna do polígono siga as mesmas características. É importante observar, no entanto, que esta restrição se aplica apenas à implementação do simulador utilizado para obter os resultados aqui apresentados e não ao modelo de indexação de meta-dados e processamento de consultas propostos. Existem algoritmos na literatura [Bose et alia (2000)] que suportam poliedros (não necessariamente retos ou bi-dimensionais) de maneira geral.

4.2 Parâmetros de Simulação

A cada execução do simulador, está associado um conjunto de parâmetros que descreve, dentre outras coisas, as características das consultas a serem simuladas e a configuração do cenário de dados sobre o qual as consultas serão feitas. A Tabela 4.1 apresenta os parâmetros de simulação utilizados considerando como aplicação o Microscópio Virtual [Ferreira et alia (1997)], aplicação na qual a geração da carga de trabalho foi baseada. Esta aplicação armazena dados que representam uma imagem de resolução extremamente alta. Através do Microscópio Virtual, usuários são capazes de especificar uma região da imagem para ser visualizada, assim como especificar o nível de *zoom* a ser utilizado para exibir a região requisitada. Para tal, os usuários submetem consultas por intervalos específicos de dados, limitados por coordenadas espaciais na imagem.

Parâmetro	**Unidade**	**Valores**
Tamanho do repositório de dados	*pixels* ou células	100k x 100k (∼37,2 GB)
Tamanho da célula	bytes	4 (código RGB)
Número de servidores	-	4, 16
Fator de heterogeneidade dos servidores	% dos servidores	50%
Tamanho da replicação (Fator x)	% do repositório de dados	0%, 20%, 40%, 80%, 100%
Cobertura da replicação (Fator y)	% dos servidores	1, 20%, 40%, 80%, 100%
Tamanho da consulta	resolução da tela *pixels*	1280x1024, 2048x1536

Tabela 4.1: Parâmetros de simulação para o Microscópio Virtual

O tamanho do repositório de dados representa a quantidade total de dados passíveis de serem consultados no sistema. De maneira geral, o tamanho do repositório pode ser dado

em número de células, menor unidade endereçável. Para o Microscópio Virtual, a célula é uma abstração para um *pixel* da imagem bi-dimensional de alta resolução, que por sua vez é representada pelo conjunto de células, ou seja, todo o repositório de dados. A imagem utilizada nas simulações possui 100 mil células em cada uma das dimensões. Consideramos cada *pixel* (ou célula) representado por 4 bytes para expressar a sua cor em código RGB. Assim, o repositório soma aproximadamente 37,2 GB.

Para o tamanho das consultas, foram considerados diferentes tamanhos de retângulos, que correspondem a diferentes resoluções de telas: 1280x1024 e 2048x1536. No entanto, a aplicação permite ao usuário selecionar entre diversos níveis de *zoom*, o que influencia no tamanho das consultas, como discutido na Seção 4.2.1.

Além do número de servidores, o simulador também permite que se especifique o grau de heterogeneidade dos servidores, em termos da capacidade para servir consultas. A capacidade dos servidores pode ser expressa em qualquer métrica que indique uma taxa limite de serviço. Para as simulações realizadas neste trabalho, consideramos a banda passante da interface de rede dos servidores como o limitador de desempenho: 10 Mbps ou 100 Mbps. O fator de heterogeneidade é o parâmetro que corresponde à porcentagem dos servidores com maior capacidade de serviço, no caso, os servidores com interfaces de redes mais rápidas. Para os resultados apresentados neste trabalho, a rede foi considerada o gargalo do sistema. No entanto, é trivial considerar outro recurso do sistema como o limitador de desempenho, como por exemplo taxa de leitura de discos rígidos ou a capacidade do processador dos servidores para atender a cada consulta.

A geração dos cenários de dados e os parâmetros de simulação relacionados são discutidos na Seção 4.2.2.

Para obter os resultados apresentados nesse trabalho, foram geradas 50 mil consultas (através do estudo descrito na Seção 4.2.1). Foram executadas várias simulações com diferentes conjuntos de parâmetros. No entanto, por questões de objetividade, os resultados aqui apresentados correspondem aos seguintes parâmetros: 16 servidores, 8 (os ímpares) com interfaces de rede de 100Mbps e outros 8 (os pares) com interfaces de 10 Mbps; consultas com resolução 2048x1536; os demais parâmetros conforme mostra a Tabela 4.1.

Para a execução do algoritmo de recozimento simulado, várias combinações de parâmetros foram utilizadas para configurar a biblioteca utilizada [Galassi M. et al. (2005)]. O conjunto de parâmetros do algoritmo se mostrou ser muito sensível à configuração do cenário de dados. No entanto, a Tabela 4.2 mostra os valores utilizados para os resultados apresentados neste capítulo. Estes valores foram selecionados por apresentarem um bom compromisso entre tempo de execução e qualidade da solução.

4.2.1 Geração da Carga de Trabalho

A geração da carga de trabalho foi baseada em uma aplicação real fortemente dependente de dados, o Microscópio Virtual. Para gerar uma carga de trabalho realista, foram analisados históricos de consultas submetidas por usuários reais enquanto utilizavam a aplicação. O histórico de consultas dessa aplicação proveu informações quanto aos polígonos que representam

Parâmetro	**Valor**	**Descrição**
Temperatura inicial	1500 K	Determina quantos passos o algoritmo tomará e quão aceitável é tomar uma decisão de piora no início.
Constante de Boltzmann (k)	1,0	Constante que multiplica a temperatura para tomar decisão de piora.
μ_T	1,002	Taxa com a qual a temperatura diminui a cada passo do algoritmo.
Temperatura mínima	10^{-5}	Temperatura de parada do algoritmo.
Iterações para T fixo	10	Número de iterações a serem realizadas sem alterar T.
Número de tentativas	200	Número de tentativas a cada passo.

Tabela 4.2: Parâmetros para recozimento simulado utilizados na biblioteca *GNU Scientific Library*.

a região da imagem que usuários pretendiam visualizar, assim como o nível de *zoom* utilizado, que permite à aplicação obter a quantidade de dados necessária do repositório para processar a consulta. Na prática, o nível de *zoom* atua como um multiplicador da área do polígono requisitado.

Sumarizando, foi realizada uma caracterização da popularidade dos níveis de *zoom*, posicionamento e tamanhos de polígonos para o Microscópio Virtual. O gráfico da Figura 4.1 apresenta a distribuição cumulativa do tamanho das consultas geradas com base nos históricos de consultas analisados. É possível observar 5 diferentes níveis de *zoom* entre as consultas geradas.

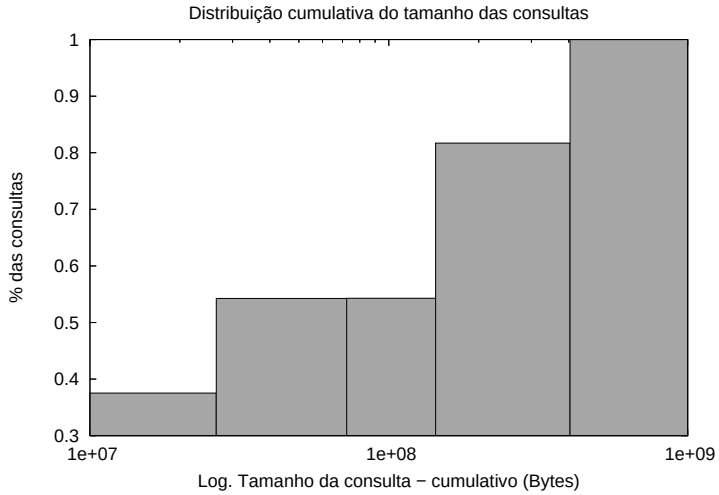


Figura 4.1: Distribuição cumulativa do tamanho das consultas, em escala logarítmica.

As consultas menores, 13631488 bytes e *zoom* de 1x, são as mais populares na coleção de consultas geradas, representando aproximadamente 37%. Uma discussão quantitativa do tamanho das consultas em bytes será apresentada adiante nesta dissertação. Os outros níveis de *zoom* presentes na coleção são: 4x, aproximadamente 17%; 8x, abaixo de 0,001%; 16x,

aproximadamente 27%; e 64x, aproximadamente 18%.

Para que fosse possível avaliar o comportamento dos algoritmos de escalonamento sob diferentes demandas, foram geradas diferentes taxas de chegada de consultas ao repositório de dados. Como as simulações lidam com um cenário de distribuição irregular de dados e um conjunto de servidores heterogêneos, o limite máximo da taxa de chegada de consultas utilizada nas simulações foi determinado empiricamente através de várias simulações. O limite empírico encontrado foi de 305 consultas por hora para a mesma configuração de cenário de dados e conjunto de servidores utilizada para gerar os resultados apresentados neste trabalho (16 servidores, consultas com resolução 2048x1536). As simulações foram realizadas com saltos de 10% entre os valores de 20% e 90% do limite empírico para a taxa de chegada de consultas. Distribuições exponenciais foram utilizadas para calcular o tempo de chegada entre consultas (IAT).

4.2.2 Geração da Configuração dos Cenários de Dados

Apesar dos dados serem irregularmente distribuídos e replicados no repositório, alguns cuidados foram tomados para que houvesse uma coerência entre os diferentes cenários de dados, permitindo uma comparação justa entre os resultados das diferentes simulações realizadas.

Os parâmetros responsáveis por produzir a replicação dos dados são: tamanho da replicação (x) e cobertura da replicação (y). O tamanho da replicação corresponde à porcentagem do repositório que será replicada entre os servidores enquanto a cobertura da replicação é a porcentagem dos servidores que receberão as frações de dados replicados durante a geração de um cenário de dados. Como um exemplo, um cenário sem replicação de dados seria representado pelo parâmetro $x = 0\%$ (y não se aplica nesta situação). Em contrapartida, um cenário com dados completamente replicados pode ser gerado com os valores de x e y iguais a 100%.

Para simular um cenário no qual os dados são irregularmente distribuídos e replicados, foi projetado um algoritmo que distribui dados aleatoriamente entre os servidores. Este algoritmo é capaz de controlar o nível de replicação de dados presente em um cenário de dados, além de garantir que todas as células do repositório estejam presentes em pelo menos um dos servidores que compõem o sistema. O algoritmo segue os seguintes passos:

1. Respeitando o tamanho do repositório de dados (especificado através do parâmetro), é gerado um determinado número de poliedros retos de tamanho e posicionamento aleatórios. São permitidas intersecções entre os poliedros;

2. Operações de decomposição são executadas sobre todos os poliedros gerados no passo anterior, juntamente com um poliedro representando todo o repositório de dados. O resultado desse passo é um conjunto de poliedros retos maximais disjuntos, de maneira que sua união corresponda ao poliedro que representa o repositório por completo;

3. Aleatoriamente, são selecionados servidores para serem responsáveis por cada poliedro. O algoritmo garante que cada servidor receba pelo menos um poliedro;

4. O repositório de dados é dividido em diversos poliedros de mesmo tamanho;

5. Aleatoriamente, é selecionado um sub-conjunto de poliedros de maneira que a soma de suas áreas corresponda a $x\%$ da área de todo o repositório;

6. Baseado em y, o algoritmo calcula quantos servidores devem receber cada um dos poliedros selecionados no passo anterior. Ou seja, para cada poliedro selecionado, são escolhidos aleatoriamente um ou mais servidores para serem responsáveis por prover a fração dos dados representada por aquele poliedro.

Após a execução dos passos listados, um cenário de dados irregularmente distribuídos e replicados foi gerado de maneira que cada servidor hospeda pelo menos uma pequena fração de todo o repositório de dados. Os passos de (1) a (3) garantem a propriedade de cobertura, ou seja, toda fração do repositório é alocada a pelo menos um servidor, e também gera a irregularidade na distribuição dos dados. Uma vez que a propriedade de cobertura é atendida, os passos de (4) a (6) geram a replicação de dados em si. O passo (4) produz vários poliedros de mesmo tamanho para que se possa haver o controle sobre quanto dado é replicado no cenário (fator x). Em seguida, os passos (5) e (6) são responsáveis por distribuir aleatoriamente, com base no fator y, um número selecionado dos poliedros de mesmo tamanho.

Para que o impacto do tamanho e cobertura da replicação dos dados fosse analisada, foram gerados vários cenários de dados com diferentes valores para os fatores x e y. Como o algoritmo projetado toma várias decisões de maneira aleatória, os cenários de dados seriam potencialmente gerados com configurações que tornassem injustas comparações entre os resultados das simulações. Portanto, foi implementado um processo de geração incremental das configurações dos cenários de dados.

A Tabela 4.1 mostra os valores utilizados para os fatores x e y para as simulações que geraram os resultados apresentados neste capítulo. Para um valor de $x = 0\%$, não há replicação de dados, portanto cada porção dos dados é hospedada por exatamente um servidor. Para todos os cenários gerados com os demais valores de x, a distribuição original das frações de dados é mantida. Desta maneira, a replicação dos dados para os diferentes valores de x respeita a distribuição original dos dados, resultante da configuração do cenário de dados com $x = 0\%$. Incrementalmente, o mesmo acontece entre os cenários de dados com $x = 20\%$ e $x = 40\%$ e assim por diante. Mais frações de dados são replicadas à medida em que o parâmetro de tamanho da replicação aumenta, porém sempre respeitando a distribuição dos dados realizada para o valor anterior do parâmetro.

Analogamente ao processo descrito para o tamanho da replicação, a cobertura da replicação também é gerada incrementalmente. Em um cenário de dados com o mínimo de cobertura de replicação ($y = 1$), as frações replicadas do repositório são hospedadas em exatamente um servidor, além do servidor que recebeu aquela fração dos dados no cenário gerado com $x = 0\%$. Para uma dada fração replicada do repositório de dados, à medida que o fator y cresce, a cobertura de replicação é incrementada e novos servidores receberão uma cópia da fração de dados até que o fator de cobertura seja alcançado.

Capítulo 5

Resultados das Simulações

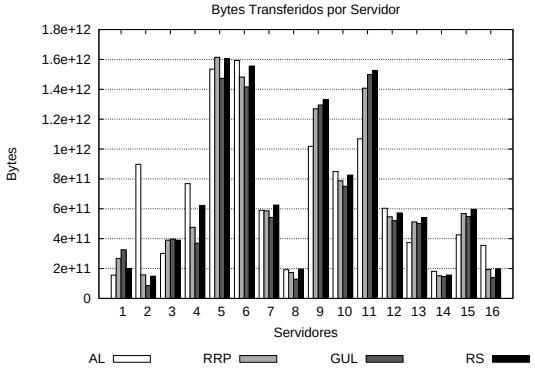
Este capítulo discute os resultados de simulação, avaliando o desempenho do sistema sob diferentes configurações de replicação e distribuição de dados, comparando os algoritmos de escalonamento de consultas propostos e analisando os aspectos de balanceamento de carga no sistema.

5.1 Análise de Balanceamento de Carga

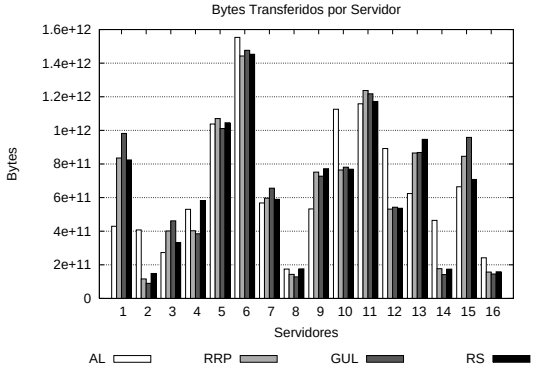
Uma vez que os dados estão potencialmente distribuídos em aglomerados de servidores ou até mesmo em grades computacionais, algoritmos ideais de seleção de servidores devem considerar a heterogeneidade da capacidade computacional dos servidores, assim como a carga imposta a eles a cada instante. Por exemplo, como os servidores podem estar conectados aos clientes através de conexões WAN/LAN de diferentes capacidades, selecionar um servidor conectado aos clientes através de uma conexão lenta para servir um fragmento muito grande de uma consulta pode tornar o tempo de resposta do serviço inaceitável. Em contrapartida, selecionar sempre os servidores com as melhores conexões com os clientes pode sobrecarregar as conexões ao ponto de resultar em um serviço de baixa qualidade.

Como os servidores em grades computacionais normalmente apresentam diferentes capacidades computacionais, o balanceamento de carga não deve implicar na requisição de uma quantidade similar de bytes para cada um dos servidores. Um balanceamento de carga ideal para servidores heterogêneos deve resultar em uma relação entre a capacidade de serviço de cada servidor e a carga de trabalho imposta a ele. Os gráficos da Figura 5.1 mostram a quantidade de bytes requisitados a cada servidor em simulações sobre diferentes configurações de repositórios e utilizando diferentes algoritmos de escalonamento de consultas. Para a geração desses gráficos, foram submetidas ao sistema 274,5 consultas por hora (90% do limite máximo).

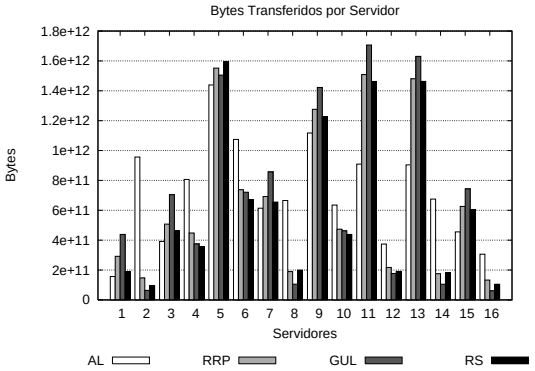
No repositório simulado, os servidores representados por números ímpares apresentavam conexões de 100 Mbps, enquanto os servidores pares apresentavam conexões mais lentas (10 Mbps). Desta maneira, um bom balanceamento de carga resultaria em uma maior concentração dos bytes requisitados aos servidores ímpares.



(a) 40% do repositório de dados replicado, com cobertura de 20% ($x = 40\%$, $y = 20\%$).



(b) 40% do repositório de dados replicado, com cobertura de 60% ($x = 40\%$, $y = 60\%$).



(c) 80% do repostitório de dados replicado, com cobertura de 20% ($x = 80\%$, $y = 20\%$).

Figura 5.1: Bytes transferidos por cada servidor em repositório submetido a uma taxa de chegada de 274,5 consultas por hora.

Em todos os gráficos da Figura 5.1 pode-se observar que o algoritmo aleatório, AL, requisita aproximadamente a mesma quantidade de bytes para as duas categorias de servidores, independente de terem maior ou menor capacidade de serviço. O motivo pelo qual a distribuição dos bytes entre cada um dos servidores não é similar é que os intervalos de dados estão distribuídos e replicados irregularmente entre os servidores e no entanto, têm diferentes popularidades. Por isto, os servidores 5 e 6, no gráfico da Figura 5.1(a) por exemplo, serviram mais bytes que os demais. Alguns intervalos populares dos dados não estavam replicados no repositório e com isto esses servidores foram os únicos candidatos para serví-los.

Os demais algoritmos no entanto, concentram uma maior parte de suas requisições nos servidores de maior capacidade. No repositório correspondente ao gráfico da Figura 5.1(a), RRP, GUL e RS escalonaram, respectivamente, 63%, 65% e 61% dos bytes para serem requisitados para os servidores com melhores conexões com os clientes, em contraste com os 50% escalonados por AL.

Para o gráfico da Figura 5.1(b), cada porção replicada do repostiório foi replicada em adicionais 40% dos servidores participantes do sistema (fator y de 20% para 60%). O aumento da cobertura dos dados replicados oferece mais opções aos algoritmos, permitindo um melhor

escalonamento. Nesse repositório, RRP, GUL e RS escalonaram, respectivamente, 64%, 65%, e 62% dos bytes para serem requisitados para os servidores com maior capacidade. No caso, o impacto do aumento do fator y na divisão dos bytes entre as categorias de servidores foi marginal uma vez que os dados já replicados não se tratavam dos mais populares. É possível observar, no entanto, um melhor balanceamento de carga entre cada um dos servidores. Por exemplo, houve um melhor balanceamento entre a quantidade de bytes requisitados ao servidores 1, 9, 13 e 15.

O gráfico da Figura 5.1(c) exibe a distribuição da requisição por bytes em um repositório com 40% mais dados replicados(fator x de 40% para 80%), com a mesma cobertura de 20% dos servidores. Enquanto o aumento na replicação não afetou a distribuição dos bytes feita pelo algoritmo aleatório, RRP, GUL e RS passaram a requisitar, respectivamente, 76%, 81% e 77% dos dados para os servidores mais capazes. Quanto maior o tamanho do repositório replicada, maior a probabilidade dos intervalos populares de dados serem replicados e maior a chance desses estarem presentes em servidores de alta capacidade, o que favorece significativamente um bom escalonamento de consultas. No gráfico, o escalonamento de bytes para o servidor 6, um servidor com conexão de baixa velocidade, caiu em aproximadamente 50%.

O gráfico da Figura 5.2 exibe a concentração da transferência dos bytes requisitados em um repositório com tamanho e cobertura de replicação máximos. Esse repositório fornece aos algoritmos todas as opções possíveis de escalonamento e portanto, todos os algoritmos, com exceção do aleatório, concentram mais de 90% dos bytes requisitados nos servidores de maior capacidade de serviço (os ímpares). O motivo pelo qual nem todos os bytes são requisitados aos servidores de maior capacidade é que existe um compromisso entre a quantidade de fragmentos de consultas pendentes em um servidor e o seu tempo de resposta e como os dois algoritmos propostos neste trabalho utilizam o conceito de capacidade disponível, em alguns casos os fragmentos de consulta foram escalonados para os servidores de menor capacidade de serviço, com a expectativa de um melhor tempo de resposta.

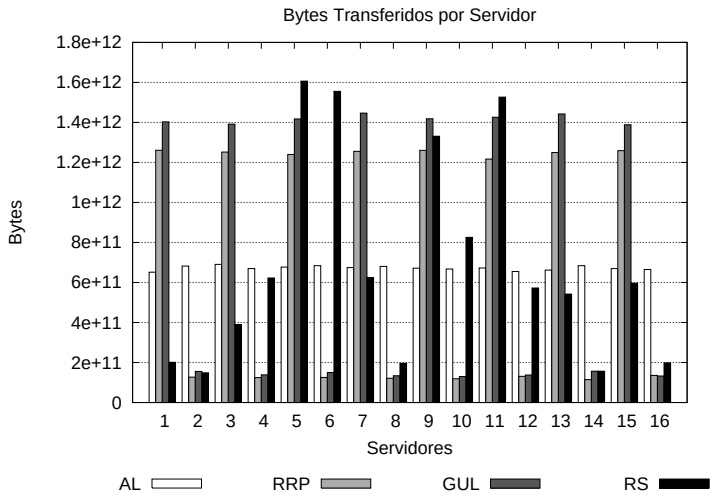
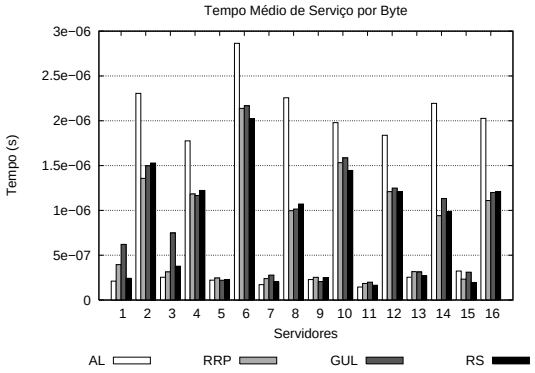
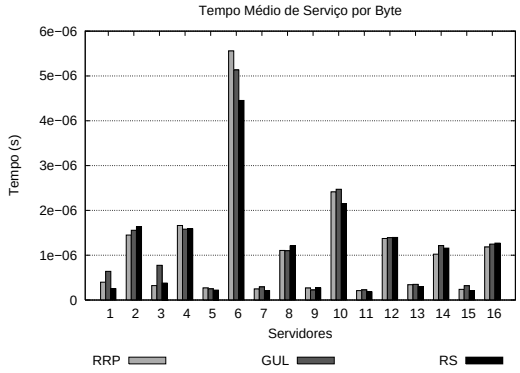


Figura 5.2: Bytes transferidos por cada servidor em repositório completamente replicado (x e $y = 100\%$) a uma taxa de chegada de 274,5 consultas por hora.

A heterogeneidade dos servidores no sistema simulado pode ser observada através dos gráficos da Figura 5.3. Os servidores ímpares possuem uma maior capacidade de serviço e portanto apresentaram um menor tempo de resposta por byte servido. O gráfico apresenta o tempo médio necessário por cada servidor para servir um byte de dados. O valor médio foi utilizado por melhor resumir o comportamento do tempo de serviço ao longo da submissão de toda a carga de trabalho aos servidores. Como a quantidade de fragmentos sendo servidos ao mesmo tempo por um servidor influencia no tempo de serviço de cada byte, apresentar valores como tempo máximo ou mínimo de serviço, retrataria momentos de pico ou vale da carga de trabalho e não mostraria com clareza a heterogeneidade dos servidores.



(a) 61 consultas por hora (20% do limite máximo suportado pelo sistema).



(b) 213,5 consultas por hora (70% do limite máximo suportado pelo sistema). AL apresentou um tempo de resposta muito maior que os demais e não foi exibido por questões de escala.

Figura 5.3: Tempo médio necessário para servir cada byte por cada servidor em um repositório com 80% dos dados replicados, com cobertura de 20% ($x = 80\%$ e $y = 20\%$).

No gráfico da Figura 5.3(a), apesar do algoritmo baseado em recozimento simulado ter resultado em um menor tempo de resposta por byte servido pelo servidor 6, em relação aos demais algoritmos, ainda é possível observar que o tempo de resposta obtido por este servidor se destacou dos demais. É contra intuitivo observar que, para a maioria dos servidores com maior capacidade de serviço (os ímpares), o algoritmo aleatório apresentou menores tempos de serviço do que os demais algoritmos. Isto ocorre por este algoritmo distribuir uniformemente requisições entre todos os servidores, enquanto os demais tendem a concentrar mais requisições nos servidores mais capazes. Assim, o uso do algoritmo alteatório resulta em menores cargas nos servidores de maior capacidade e, por conseqüência, esses servidores conseguem servir a estas requisições mais rapidamente. Este fato, porém, não implica em melhor tempo de resposta das consultas como um todo, uma vez que os servidores de menor capacidade recebem aproximadamente metade de toda a carga imposta ao sistema. A relação entre o desempenho do sistema e os algoritmos utilizados será discutida em detalhes na Seção 5.2.

A diferença entre os tempos de resposta por byte transferido se agrava ainda mais à medida que a taxa de chegada de consultas aumenta. No gráfico da Figura 5.3(b), o algoritmo baseado em recozimento simulado se manteve como o algoritmo de escalonamento com o melhor resultado. Ainda assim, o tempo de resposta por byte transferido pelo servidor 6 é

maior que o dobro do tempo de resposta alcançado pelos demais servidores.

Como mencionado anteriormente, os intervalos de dados possuem diferentes tamanhos e diferentes níveis de popularidade entre as consultas submetidas. O gráfico da Figura 5.4 mostra que alguns intervalos de dados chegam a ser requisitados em mais de 40% das consultas submetidas ao sistema. A mesma análise utilizada para gerar este gráfico apontou que mais de 75% dos intervalos de dados presentes no índice global não foram requisitados por nenhuma das consultas submetidas ao sistema.

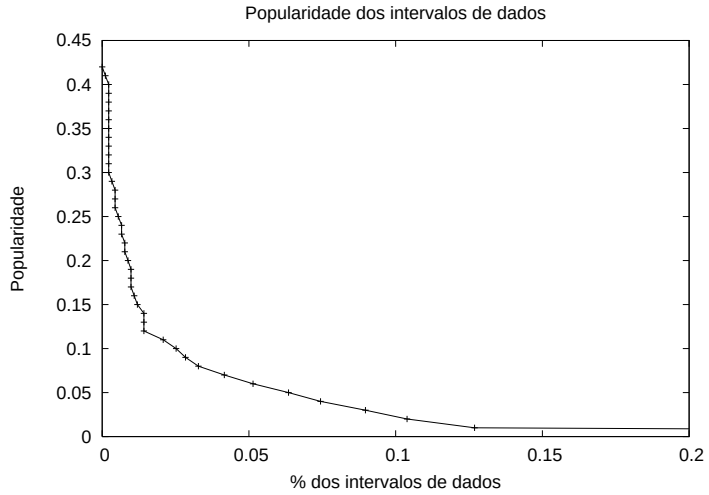


Figura 5.4: Enquanto mais de 75% dos intervalos de dados não são consultados mais de uma vez, alguns poucos intervalos de dados (menos de 0,001%) são requisitados em mais de 20% das consultas, chegando a um extremo de aproximadamente 42% das consultas.

Em algumas configurações de dados, é possível que alguns dos intervalos de dados mais populares não sejam replicados, e portanto, um único servidor fique responsável por servir esses dados. Com uma alta taxa de chegada de consultas, a capacidade de serviço disponível no servidor diminui e os intervalos de dados pendentes são servidos a taxas muito baixas. Esta situação de concentração de carga pode ainda ser agravada caso o servidor responsável possua baixa capacidade de serviço, como no caso do servidor 6, ilustrado pela Figura 5.3.

Ainda como uma análise das características de balanceamento de carga do sistema e dos algoritmos implementados, foi observado o coeficiente de variação (CV) entre os tempos de resposta dos fragmentos de uma consulta. O gráfico da Figura 5.5 mostra o valor médio do CV para os diferentes algoritmos implementados e sob diferentes taxas de chegada de consultas submetidas ao sistema.

O tempo de resposta dos fragmentos das consultas apresentam um valor médio do CV alto até mesmo sob baixas taxas de chegada de consultas. Por exemplo, o valor médio para uma taxa de 61 consultas por hora é aproximadamente 1 para todos os algoritmos de escalonamento, com exceção do algoritmo aleatório, cujo valor varia de maneira independente da taxa de chegada das consultas ou da configuração do repositório. O CV aumenta com a taxa de chegada de consultas. No caso ilustrado, o valor médio atingiu aproximadamente 2, com algumas consultas atingindo coeficientes extremamente altos, como 10.

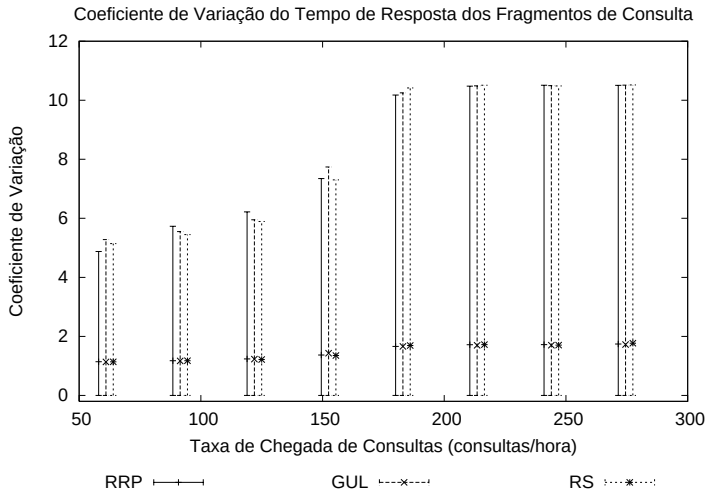


Figura 5.5: Variação entre tempos de resposta dos fragmentos de uma consulta aumentam com a taxa de chegada de consultas submetidas ao sistema. Repositório com 60% dos dados replicados, com cobertura de 60% (x e $y = 60\%$).

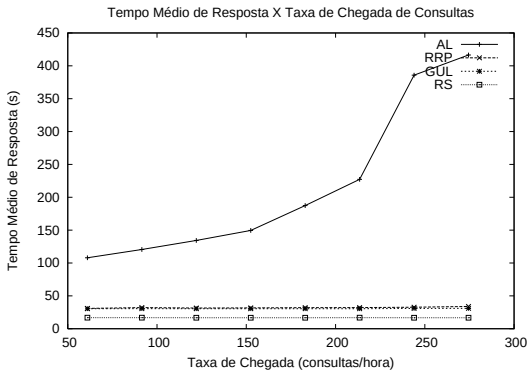
Os altos valores de CV indicam que, enquanto alguns fragmentos das consultas são servidos com um baixo tempo de resposta, outros fragmentos demoram para ser servidos, seja devido a um servidor com baixa capacidade disponível, baixa cobertura de replicação ou devido a um fragmento de consulta muito maior do que os demais. Uma vez que o maior tempo de resposta entre os fragmentos de uma consulta é considerado o tempo de resposta da consulta, o comportamento ilustrado na Figura 5.5 representa uma oportunidade de melhoria para o sistema. Dado que os algoritmos de escalonamento conseguem prever o tempo de resposta de cada fragmento de uma consulta, a quantidade mínima de recursos poderia ser alocada a cada um dos fragmentos que são servidos rapidamente, de maneira que todos os fragmentos sejam servidos com o mesmo tempo de resposta do fragmento mais lento. Esta melhoria não impactaria negativamente no tempo de resposta das consultas uma vez que este é definido pelo tempo de resposta do fragmento mais lento, que continuaria sendo servido tão rápido quanto possível, de acordo com a métrica utilizada pelo algoritmo de escalonamento. No entanto, os recursos dos servidores seriam utilizados de maneira mais inteligente, o que resultaria em uma melhoria global no tempo de resposta das consultas.

5.2 Análise de Desempenho

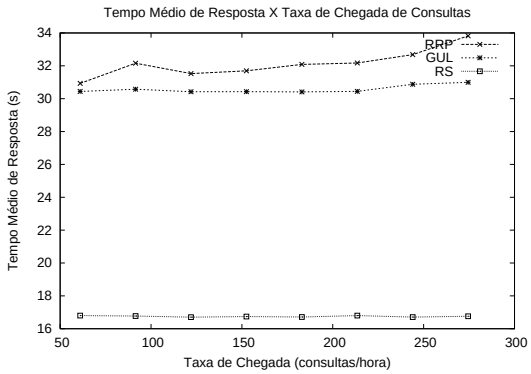
Como citado anteriormente, foi determinado empiricamente um intervalo de taxas de chegada de consultas suportado pelo sistema composto de servidores heterogêneos e uma configuração de dados irregularmente distribuídos e replicados. Com o objetivo de avaliar o desempenho do sistema utilizando diferentes algoritmos de seleção de servidores, foram realizadas simulações do sistema sob diferentes taxas de chegada de consultas dentro do intervalo suportado pelo sistema. A métrica de desempenho considerada para o sistema de virtualização de dados foi o tempo de resposta das consultas por ele atendidas. Ainda, o valor médio dos vários tempos de

resposta foi utilizado como uma medida global de desempenho. Como a carga atual imposta a um servidor afeta diretamente o seu tempo de serviço de dados, a utilização de valores como tempo máximo ou mínimo de resposta retratariam momentos específicos dos experimentos, nos quais a carga de trabalho apresentou picos ou vales de requisições.

A Figura 5.6 mostra o desempenho dos diferentes algoritmos de escalonamento de fragmentos de consultas em um repositório completamente replicado. Nesse repositório, todas as opções possíveis de escalonamento de fragmentos estão disponíveis para os algoritmos de seleção de servidores. Deste maneira, apenas a capacidade dos servidores e a carga instantânea imposta a eles limitam o escalonamento de um fragmento de consulta a um dos servidores do sistema.



(a) O desempenho do sistema com o algoritmo aleatório se deteriora sensivelmente com o aumento da taxa de chegada de consultas.



(b) Algoritmo baseado em recozimento simulado resultou em um ganho de aproximadamente 50% de desempenho.

Figura 5.6: Tempo médio de resposta em função da taxa de chegada de consultas em um repositório completamente replicado. (x e $y = 100\%$).

É possível observar no gráfico da Figura 5.6(a) que o algoritmo aleatório leve o sistema ao ponto de saturação muito antes dos demais algoritmos. Isto ocorre uma vez que não há qualquer esforço por parte do algoritmo para selecionar o servidor mais apropriado para servir cada um dos fragmentos das consultas. Para permitir um melhor contraste entre os demais algoritmos, as mesmas curvas foram apresentadas no gráfico da Figura 5.6(b) e a curva referente ao algoritmo aleatório foi removida por questões de escala.

A utilização do algoritmo *round robin* ponderado resultou em um ganho de desempenho entre 72,0% e 92,1% quando comparado com o sistema operando com o algoritmo aleatório. O esforço voltado para o balanceamento da carga dos servidores (escolhendo-os alternadamente para servidor fragmentos das consultas) e as tomadas de decisões ponderadas pela capacidade de serviço dos servidores são os responsáveis por tal ganho de desempenho. Ainda, a degradação do desempenho foi menos sensível ao aumento da taxa de chegada de consultas.

Ao considerar a carga instantânea imposta aos servidores do sistema e o tamanho dos fragmentos das consultas, o algoritmo baseado no paradigma guloso conquistou um desempenho ainda melhor sobre a mesma configuração de dados. Os ganhos de desempenho do algoritmo guloso sobre o algoritmo *round robin* ponderado variaram entre 1,6% e 9,0%.

O algoritmo baseado na abordagem de otimização alcançou um ganho de desempenho de aproximadamente 47% com relação ao algoritmo guloso. Ao contrário do algoritmo guloso, que busca o melhor escalonamento disponível para cada um dos fragmentos da consulta, o algoritmo baseado em recozimento simulado busca o melhor escalonamento com uma visão global do tempo de resposta da consulta. Desta maneira, os fragmentos de uma mesma consulta não disputam entre si pelo melhor tempo de resposta, favorecendo o tempo de resposta global.

Um repositório completamente replicado torna disponível para os algoritmos de seleção de servidores, todas as possíveis alternativas de escalonamento uma vez que todos os servidores são candidatos para atender a uma consulta por intervalos de dados. Esta configuração de dados nos permite observar uma característica interessante dos algoritmos de escalonamento propostos. Através do gráfico da Figura 5.6(b), é possível observar que, estando dentro do limite suportado pelo sistema, a taxa de chegada de consultas não impõe virtualmente qualquer influência sobre o tempo de resposta das consultas submetidas a um repositório completamente replicado. Esta característica indica que os algoritmos propostos fazem boa utilização das informações quanto a carga instantânea imposta e heterogeneidade dos servidores que compõem o sistema de virtualização de dados.

No entanto, replicação total dos dados não é a configuração mais comum nos repositórios para os quais o sistema de virtualização de dados explorado foi desenvolvido. Mais comumente, intervalos de dados são replicados irregularmente entre os servidores e em alguns casos, intervalos populares de dados estão disponíveis em uma única fonte de dados. O gráfico da Figura 5.7 mostra o desempenho do sistema sobre um repositório com baixa taxa de replicação (20%), ainda que com uma alta taxa de cobertura (60%). O tempo de resposta médio das consultas submetidas a este repositório foi ordens de grandeza maior do que o tempo de resposta obtido no repositório completamente replicado. Esta diferença no tempo de resposta se dá pela concentração de intervalos de dados populares em um único servidor, e em muitos casos, um servidor com baixa capacidade de serviço.

Com a existência de um ponto de contenção, o desempenho do sistema utilizando os algoritmos propostos passou a ser sensível à taxa de chegada de consultas, assim como com os algoritmos AL e RRP no repositório descrito anteriormente. Apesar dos algoritmos apresentarem desempenhos mais similares em situações de contenção, ainda é possível observar que o algoritmo baseado em recozimento simulado atingiu o ponto de saturação mais lentamente que os demais, e portanto, desempenhou melhor sob altas taxas de chegada.

Como os servidores que compõem o sistema são heterogêneos e os intervalos de dados apresentam diferentes níveis de popularidade, o sistema se beneficia muito mais de uma maior taxa de replicação do que de uma maior cobertura dos dados replicados. O impacto de se disponibilizar mais opções de escalonamento para fragmentos menores e menos populares (maior cobertura de replicação) é menor do que a existência de um mínimo de distribuição das porções mais populares do repositório de dados (maior taxa de replicação). O gráfico da Figura 5.8 permite observar que, ainda que com uma baixa cobertura dos dados replicados (20%, 3 servidores), a distribuição da maior parte do repostiório e diminuição do número de

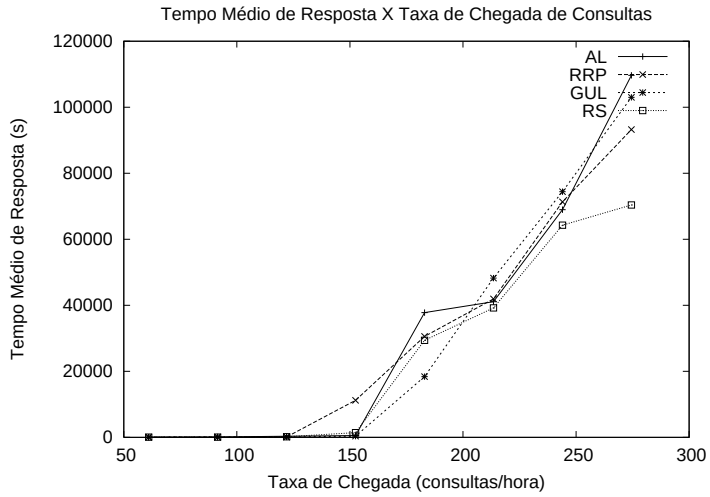


Figura 5.7: Tempo médio de resposta em função da taxa de chegada de consultas em um repositório com 20% dos dados replicados, com cobertura de 60%. ($x = 20\%$ e $y = 60\%$).

pontos de contenção teve um impacto fundamental no desempenho do sistema. Com uma taxa de replicação de 80%, o desempenho do sistema se aproximou do desempenho obtido em uma configuração de replicação total.

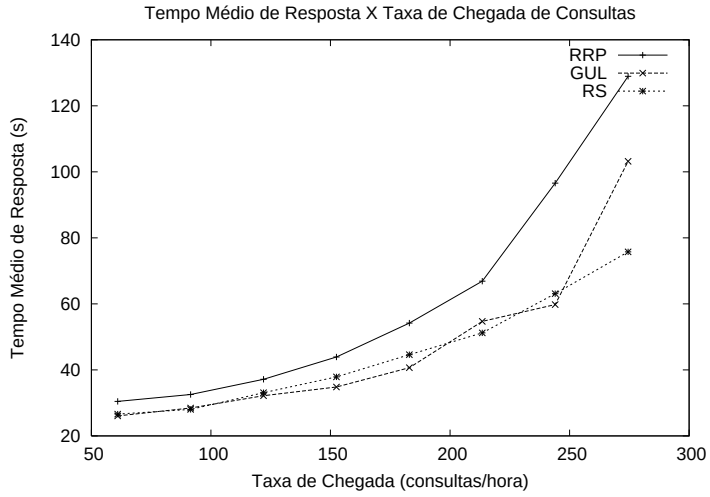
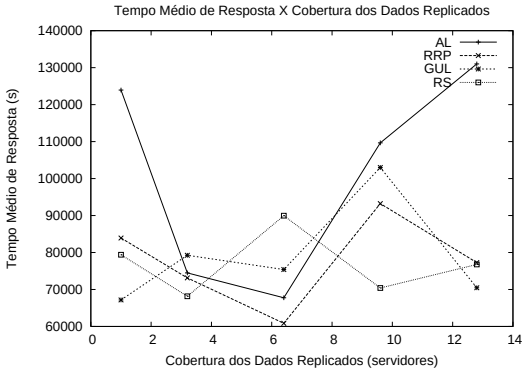


Figura 5.8: Tempo médio de resposta em função da taxa de chegada de consultas em um repositório com 80% dos dados replicados, com cobertura de 20%. ($x = 80\%$ e $y = 20\%$).

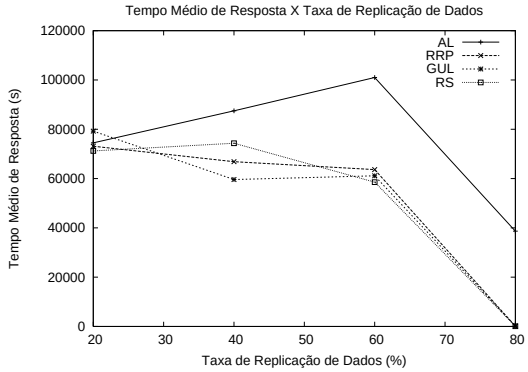
A existência de alguns poucos pontos de contenção fazem com que os algoritmos propostos sejam sensíveis à taxa de chegada das consultas, ainda que dentro do intervalo suportado pelo sistema. O desempenho do algoritmo aleatório não foi ilustrado no gráfico por questões de escala. Seu tempo de resposta médio variou entre 59,7 e mais de 38660 segundos.

Os gráficos da Figura 5.9 dão suporte a esta constatação. Na Figura 5.9(a), pode-se observar que um aumento na cobertura dos dados replicados não necessariamente exercem influência no desempenho do sistema. Como a maioria dos intervalos de dados apresentam

baixa popularidade (como mostrou o gráfico da Figura 5.4), a uma baixa taxa de replicação, as chances são grandes de serem replicados apenas intervalos de dados pouco populares e neste caso, uma maior cobertura da replicação destes intervalos de dados não traz benefícios significativos para o sistema em termos de tempo de resposta.



(a) Maior cobertura dos dados replicados não necessariamente melhora a escalabilidade do sistema com servidores heterogêneos, uma vez que dados replicados podem se concentrar em servidores com menor capacidade de serviço ($x = 20\%$).



(b) O desempenho do sistema é favorecido pela replicação de intervalos de dados populares, o que diminui os pontos de contenção ($y = 20\%$).

Figura 5.9: Tempo médio de resposta a uma taxa de chegada de 274,5 consultas por hora.

Através do gráfico da Figura 5.9(b) foi possível observar que um aumento na taxa de replicação do repostitório favorece diretamente o desempenho do sistema. Uma vez que os intervalos dados mais populares são replicados, uma maior taxa de cobertura desses intervalos tende a impactar diretamente no tempo de resposta das consultas, uma vez que aumentam-se os pontos de distribuição de carga.

Capítulo 6

Conclusão e Trabalhos Futuros

Este trabalho propôs um modelo de indexação de meta-dados e processamento de consultas que permite a criação de um sistema de virtualização de dados em repositórios irregularmente replicados e distribuídos. O modelo proposto se baseia em algoritmos geométricos para indexar os meta-dados referentes aos dados hospedados em cada fonte de dados no sistema. O modelo se estende também ao processamento de consultas por intervalos de dados para o sistema de virtualização. De maneira semelhante ao método de indexação de meta-dados, o processamento de consultas representa cada consulta como um conjunto de fragmentos através de uma seqüência de operações geométricas.

O modelo proposto é flexível e pode ser facilmente adaptado para suportar diversos tipos de aplicações sobre o sistema de virtualização de dados. A utilização do modelo junto a novas aplicações demanda apenas que se determine o número de dimensões desejadas para as representações geométricas e que seja definido um mapeamento entre os limites dos repositórios de dados ou parâmetros da aplicação e um espaço de coordenadas.

Dois algoritmos de escalonamento foram propostos para guiar a decisão de qual servidor deve servir cada um dos intervalos de dados que compõem uma consulta. O desempenho dos algoritmos desenvolvidos foi comparado com o de algoritmos mais básicos. Ao considerar a carga instantânea à qual os servidores do sistema estão submetidos, os algoritmos propostos apresentaram ganhos significativos em tempo de resposta, como fruto de um melhor escalonamento dos fragmentos das consultas e um balanceamento de carga mais justo entre as fontes de dados.

Os resultados das simulações apresentaram o algoritmo baseado em recozimento simulado como a melhor estratégia de escalonamento de fragmentos de consultas, resultando em melhorias de desempenho de até 47% quando comparado com o algoritmo baseado na abordagem gulosa. O desempenho do algoritmo baseado em recozimento simulado se destacou principalmente quando submetido a taxas de chegada de consultas próximas do limite suportado pelo sistema de virtualização, enquanto os outros algoritmos estudados atingem seu ponto de saturação com taxas de chegada mais baixas.

O trabalho apresentou um contraste do desempenho do sistema entre diferentes configurações de dados. A caracterização da carga de trabalho do Microscópio Virtual mostrou que

alguns intervalos de dados são extremamente populares, podendo ocorrer em mais de 40% das consultas. Por este motivo, configurações nas quais os intervalos de dados mais populares apresentam uma baixa taxa de replicação limitam significativamente o desempenho do sistema. O desempenho do sistema se mostrou muito pouco sensível à taxa de chegada de consultas em um repositório completamente replicado. A taxa de cobertura da replicação dos dados só passou a influenciar significativamente no desempenho do sistema uma vez que os intervalos mais populares foram replicados. Estas constatações permitiram apontar os intervalos de dados mais populares como os principais pontos de contenção do sistema, dado um limite máximo de taxa de chegada de consultas suportado. O limite citado é definido pela capacidade de serviço dos servidores que compõem o sistema.

6.1 Trabalhos Futuros

O algoritmo baseado em recozimento simulado necessita de configuração de um conjunto de parâmetros para cada aplicação alvo, o que justifica a adoção de um algoritmo mais genérico, ainda que seu desempenho não supere o obtido através da técnica de recozimento simulado. O algoritmo guloso segue uma abordagem simples para selecionar servidores para atender consultas por intervalos de dados, o que favorece seu tempo de execução. No entanto, outros passos baseados em diferentes heurísticas podem refinar as soluções obtidas pelo algoritmo guloso sem afetar significativamente o seu tempo de execução.

O estudo de coeficientes de variação indicou uma grande diferença entre os tempos de resposta dos fragmentos de uma consulta. Apesar do fragmento de consulta com maior tempo de serviço determinar o tempo de resposta da consulta como um todo, na versão atual do sistema, todos os fragmentos de uma consulta são servidos o mais rapidamente possível. Desta maneira, fragmentos críticos para o tempo de resposta de uma consulta consomem tantos recursos dos servidores quanto os demais fragmentos. O sistema deveria ser capaz de alocar recursos para o serviço de fragmentos de consultas de acordo com o quão decisivo é o fragmento para o tempo de resposta da consulta. Desta maneira, o serviço de fragmentos menores poderia ser adiado para favorecer o tempo de resposta de fragmentos maiores ou ainda, esses fragmentos poderiam ser alocados a servidores com menor capacidade de serviço. Medidas como essas poderiam diminuir o tempo de resposta das consultas.

Os resultados deste trabalho mostraram que a contenção de recursos em torno de porções de dados com alta popularidade é um fator decisivo no tempo de resposta das consultas submetidas ao sistema. Muitos dos sistemas de virtualização de dados existentes agem como replicadores de dados sob demanda. Existem oportunidades interessantes a serem exploradas com relação à replicação de dados em configurações irregulares como as endereçadas nesta dissertação. Como os repositórios abordadas são grandes, o compromisso entre desempenho e utilização de recursos de armazenamento se torna extremamente crítico. Como a qualidade do serviço provido ao cliente depende, dentre outras coisas, de sua proximidade na rede com relação ao servidor, é importante associar a necessidade de replicação de determinadas porções dos dados a um sub-conjunto dos clientes do sistema. Ainda, as porções de dados que devem

ser replicadas tendem a ser irregulares uma vez que apenas alguns fragmentos em comum entre as consultas seriam recorrentes, justificando a utilização de recursos para replicação.

Uma outra direção interessante seria a exploração da informação de popularidade das porções de dados nos algoritmos de escalonamento de consultas. Um exemplo de abordagem seria buscar diminuir a carga imposta sobre servidores que hospedam dados mais populares.

Um outro conceito que explora a popularidade de objetos de consulta é *caching*. Sistemas de *caching* com granularidade fina [Deshpande et alia (1998)], que permitem a consultas reutilizarem, parcialmente, os resultados obtidos por consultas anteriores, teriam uma aplicação interessante em repositórios irregulares como os descritos neste trabalho, uma vez que apenas pequenos fragmentos de uma consulta poderiam ser mantidos em memória principal. Alguns dos sistemas de *caching* providos pelos próprios mecanismos de armazenamento de dados não possuem granularidade menor do que uma consulta por inteiro, o que resultaria na manutenção de frações de dados pouco populares em *cache*, impactando no tempo de resposta do sistema.

Nos casos de aplicações que permitam a divisão do problema através de uma abordagem como "dividir para conquistar", a utilização de sistemas de *caching* semântico
[Godfrey e Gryz (1999)] resultariam em diminuições drásticas na transferência de dados através da rede (dados pré-processados seriam transmitidos). No entanto, compromissos devem ser avaliados uma vez que a carga de processamento de consultas nos servidores tende a aumentar devido ao processamento local realizado nos mesmos.

Referências Bibliográficas

Allcock, B.; Bester, J.; Bresnahan, J.; Chervenak, A. L.; Foster, I.; Kesselman, C.; Meder, S.; Nefedova, V.; Quesnel, D. e Tuecke, S. (2002). Data management and transfer in high-performance computational grid environments. *Parallel Computing*, 28:749–771.

Alpdemir, M. N.; Mukherjee, A.; Gounaris, A.; Paton, N. W.; Watson, P.; Fernandes, A. A. A. e Smith, J. (2003). OGSA-DQP: A service-based distributed query processor for the grid. In *Anais do Segundo e-Science All Hands Meeting*.

Amza, C.; Cox, A. L. e Zwaenepoel, W. (2003). Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In *Anais do Quarto ACM/IFIP/USENIX International Middleware Conference*, págs. 282–304, Rio de Janeiro, Brasil.

Baer, T. e Wyckoff, P. (2004). A parallel i/o mechanism for distributed systems. In *Anais do IEEE International Conference on Cluster Computing 2004*, págs. 63–69.

Bartal, Y.; Fiat, A. e Rabani, Y. (1992). Competitive algorithms for distributed data management. In *Anais do Vigésimo Quarto Annual ACM Symposium on Theory of Computing*, págs. 39–50.

Baumann, P.; Furtado, P.; Ritsch, R. e Widmann, N. (1997). The rasdaman approach to multidimensional database management. In *Selected Areas in Cryptography*, págs. 166–173.

Blustein, J. (1995). Implementing bit vectors in C. *Dr. Dobb's Journal*, 20(8):42, 44, 46.

Bose, P.; Czyzowicz, J.; Kranakis, E.; Krizanc, D. e Maheshwari, A. (2000). Polygon cutting: Revisited. *Lecture Notes in Computer Science*, págs. 81–89.

Chervenak, A.; Foster, I.; Kesselman, C.; Salisbury, C. e Tuecke, S. (1999). The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets.

Database Access and Integration Services Working Group (2005). Open grid services architecture - data access and integration – ogsa-dai. http://www.ogsadai.org.uk.

Deshpande, P. M.; Ramasamy, K.; Shukla, A. e Naughton, J. F. (1998). Caching multidimensional queries using chunks. In *SIGMOD '98: Anais do ACM SIGMOD international conference on Management of data*, págs. 259–270, New York, NY, USA. ACM Press.

DeWitt, D. e Gray, J. (1992). Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98.

Diniz, B.; Nogueira, D. L.; Cardoso, A.; Ferreira, R. A.; Guedes, D. e Meira Jr., W. (2006). Assessing data virtualization for irregularly replicated large datasets. In *Anais do Sexto IEEE International Symposium on Cluster Computing and the Grid*, págs. 505–512, Singapore.

Epstein, R.; Stonebraker, M. e Wong, E. (1978). Distributed query processing in a relational data base system. In *Anais do ACM SIGMOD international conference on management of data*, págs. 169–180.

Ferreira, R.; Moon, B.; Humphries, J.; Sussman, A.; Saltz, J.; Miller, R. e Demarzo, A. (1997). The Virtual Microscope. In *Anais do American Medical Informatics Association Annual Fall Symposium*, págs. 449–453, Nashville, TN.

Galassi M. et al. (2005). *GNU Scientific Library Reference Manual*, 2a edição.

Godfrey, P. e Gryz, J. (1999). Answering queries by semantic caches. In *Database and Expert Systems Applications*, págs. 485–498.

Gribble, S. D.; Brewer, E. A.; Hellerstein, J. M. e Culler, D. (2000). Scalable, distributed data structures for internet service construction. In *Anais do Quarto Symposium on Operating Systems Design and Implementation (OSDI 2000)*.

Haas, L. M.; Kossmann, D.; Wimmers, E. L. e Yang, J. (1997). Optimizing queries across diverse data sources. In *Anais do Vigésimo Terceiro International Conference on Very Large Databases*, págs. 276–285, Athens, Greece. VLDB Endowment, Saratoga, Calif.

Kemper, A. e Wiesner, C. (2001). Hyperqueries: Dynamic distributed query processing on the internet. In *The VLDB Journal*, págs. 551–560.

Kirkpatrick, S.; Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, Núm. 4598, 13 Maio 1983*, 220, 4598:671–680.

Kossman, D. (2000). The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469.

Luo, H.; Ye, F.; Cheng, J.; Lu, S. e Zhang, L. (2005). TTDD: Two-tier data dissemination in large-scale wireless sensor networks. *Wireless Networks*, 11(1-2):161–175.

Moore, R.; Baru, C.; Marciano, R.; Rajasekar, A. e Wan, M. (1999). *Data-Intensive Computing*, cap. 5, págs. 105–129. The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann.

Moore, R. W. e Baru, C. (2003). *Grid Computing: Virtualization Services for Data Grids*, cap. 16, págs. 409–435. Wiley Series in Communications Networking & Distributed Systems. John Wiley & Sons, Ltd.

Murta, A. (2005). A general polygon clipper library. http://www.cs.man.ac.uk/∼toby/alan/software.

Narayanan, S.; Catalyurek, U. V.; Kurc, T. M.; Zhang, X. M. e Saltz, J. H. (2003). Applying database support for large scale data driven science in distributed environments. In *Anais do Quarto International Workshop on Grid Computing (Grid 2003)*, págs. 141–148.

O'Rourke, J. (1998). *Computational Geometry in C.* Cambridge University Press.

Rahm, E. e Marek, R. (1995). Dynamic multi-resource load balancing in parallel database systems. In *Anais do Vigésimo Primeiro International Conference on Very Large Data Bases*, págs. 395–406.

Ranganathan, K. e Foster, I. (2002). Decoupling computation and data scheduling in distributed data-intensive applications. In *Anais do Décimo Primeiro IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC'02).*

Sarawagi e Stonebraker (1994). Efficient organization of large multidimensional arrays. In *ICDE: 10th International Conference on Data Engineering.* IEEE Computer Society Technical Committee on Data Engineering.

Soundararajan, G.; Manassiev, K.; Chen, J.; Goel, A. e Amza, C. (2005). Feedback-based scheduling for back-end databases in shared dynamic content server clusters. In *Anais do Segundo International Conference on Automatic Computing*, págs. 348 – 349, Washington, DC.

Stolte, C.; Tang, D. e Hanrahan, P. (2002a). Multiscale visualization using data cubes. In *Anais do Oitavo IEEE Symposium on Information Visualization.*

Stolte, C.; Tang, D. e Hanrahan, P. (2002b). Query, analysis, and visualization of hierarchically structured data using polaris. In *Anais do Oitavo ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

The Globus Alliance (2005). Globus toolkit. http://www.globus.org.

Watson, P. (2005). Databases in grid applications: Locality and distribution. In *Anais do Database: Enterprise, Skills and Innovation. 22nd British National Conference on Databases, BNCOD 22.*

Weng, L.; Agrawal, G.; Catalyurek, U. V.; Kurc, T. M.; Narayanan, S. e Saltz, J. H. (2004). An approach for automatic data virtualization. In *Anais do Décimo Terceiro IEEE International Symposium on High-Performance Distributed Computing (HPDC-13)*, págs. 24–33.

Yu, C. T. e Chang, C. C. (1984). Distributed query processing. *ACM Computing Surveys*, 16(4):399–433.

Özsu, M. T. e Valduriez, P. (1999). *Principles of distributed database systems.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, segunda edição.