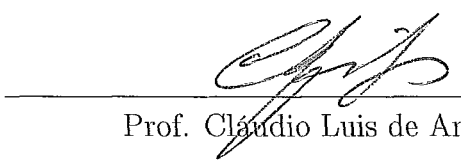
INTEGRAÇÃO DE BANCO DE DADOS EM AMBIENTES DE GRID

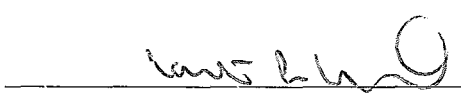
João Victor Pap Almeida

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

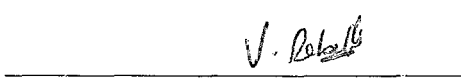
Aprovada por:



Prof. Cláudio Luis de Amorim, Ph.D.



Profª. Marta Lima de Queirós Mattoso, D.Sc.



Prof. Eugene Francis Vinod Rebello, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2008

ALMEIDA, JOÃO VICTOR PAP

INTEGRAÇÃO DE BANCOS DE DADOS EM AMBIENTES DE GRID [Rio de Janeiro] 2008

XII, 104 p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2008)

Dissertação – Universidade Federal do Rio de Janeiro, COPPE

1 - Integração de bancos de dados

2 - grids de dados

3 - tabelas virtuais

I. COPPE/UFRJ II. Título (série)

À minha grande orientadora e amiga, Inês Dutra, que me apoiou em todo o momento nessa difícil jornada do Mestrado. Ao grande amigo Paulo Motta, que sempre me ajudou nos mais dificeis projetos, incentivando-me a continuar.

À Carlos Augusto (Guto), Vera Prudência e Adriano Caminha, grandes professores na graduaçao que me deram o pontapé inicial para o Mestrado. À Meri Toleidano, Mauro Staretz e Solange Scolatempore, da MI Montreal Informática, que confiaram em meu pontencial liberando-me para o Mestrado. À Denise Mattos, Marcelo Caux, Alexandre Amorim, Kyle Malone, Sergio Santos, Omar, Marcia, da Electronic Data System, que confiaram em meu pontencial me liberando para o Mestrado. À galera da Compera nTime, que me apoiou sempre no Mestrado Aos amigos que fiz na COPPE. À meus pais e irmão, por todo o apoio.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc)

INTEGRAÇÃO DE SGBDS EM AMBIENTES DE GRID

João Victor Pap Almeida

SETEMBRO/2008

Orientadores: Cláudio Luis de Amorim

Inês de Castro Dutra

Programa: Engenharia de Sistemas e Computação

Sistemas de Grid podem ser classificados em dois grandes grupos: grids computacionais (Computational Grids) e grids de dados (DataGrids). Muitos sistemas têm sido desenvolvidos voltados para grids computacionais, porém muito poucos têm estado voltados para grids de dados. Algumas soluções vêm sendo desenvolvidas para orquestração de sistemas de arquivos e poucos focam na orquestração de bases de dados. Sistemas tais como AMGA, GREIC e OGSA-DAI, este último desenvolvido no contexto do Open Grid Forum (OGF), têm procurado oferecer soluções para a disponibilização de dados locais em ambientes de grid e integração de bancos de dados heterogêneos e dispersos geograficamente. Normalmente, estas soluções exigem: (1) que o usuário conheça a localização física das tabelas locais a cada nó do grid ou conheça detalhes sobre o sistema gerenciador de bancos de dados; ou (2) que os dados sejam importados das bases de dados, localizadas em cada nó, para um servidor. Neste trabalho, propomos a utilização de tabelas virtuais definidas pelos administradores dos nós do grid para tornar transparente o acesso aos dados pelos usuários. Cada nó mantém sua individualidade e pode disponibilizar os dados no grid através de tabelas virtuais. Este esquema traz algumas vantagens: permite que o administrador defina permissões e políticas de acesso aos diversos usuários de cada sítio; permite que dados mantenham-se confidenciais; oferece ao usuário uma única visão dos dados; "esconde" do usuário a organização física das tabelas, o que pode reforçar a segurança dos dados; disponibiliza SGBDs heterogêneos em ambientes de grid; pode ser utilizado para armazenar maiores quantidades de dados, visto que as tabelas físicas estão distribuídas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

INTEGRATION OF DATABASES ON GRID ENVIRONMENTS

João Victor Pap Almeida

SEPTEMBER/2008

Advisors: Cláudio Luis de Amorim

Inês de Castro Dutra

Department: Computing and Systems Engineering

Grid systems have been recently utilized by researchers worldwide. Several hardware and software infrastructures make it possible to develop the so-called e-Science. These systems can be classified in two big groups: computational grids and datagrids. Many systems have been developed whose focus is on computational grids, however very few have been dedicated to datagrids. Some solutions have been developed for the orchestration of file systems and few concentrate on the orchestration of databases. Systems such as AMGA, GREIC and OGSA-DAI (the latter one developed in the context of the Open Grid Forum – OGF), offer solutions that make local data available in grid environments, or integrate heterogeneous and distributed databases. Most often, these solutions require that the user knows the physycal location of the table or the kind of database being utilised in the remote grid node. In this work we offer another solution that makes the access to different databases transparent to the user. We use virtual tables defined and configured by grid site administrators to "virtually" represent the physical tables located at different grid sites. Each grid site can maintain its own view of the data, but there is one virtual table that connects all physical tables. This scheme can bring some advantages: it allows the definition of user permissions and access policies for each grid site; it allows data to be kept confidential; it offers to the grid user one single view of the data; it hides from the grid user the physical organization and location of data tables what can reinforce data security and confidentality; integrate heterogeneous databases; it can be utilized to store larger tables, since physical tables are distributed.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Sistemas de Computação em Grid são uma realidade atualmente e têm sido utilizados de forma bem sucedida em várias áreas de aplicação. Um grid, segundo Foster e outros autores [14, 17], pode ser definido como um conjunto de recursos dispersos geograficamente, onde estes recursos estão organizados com políticas bem definidas de utilização e oferecem poder computacional ou requisitos específicos (conjuntos de dados, instrumentos, dentre outros) para uma ou mais organizações virtuais. Uma organização virtual consiste em um grupo de pessoas com objetivos semelhantes que necessita dos recursos para processamento em sua área de domínio. Stockinger [28] classifica grids em dois grandes grupos: grids computacionais e grids de dados. Em grids computacionais o foco está na utilização dos milhares de recursos dispersos geograficamente para execução de aplicações paramétricas, bag-of-tasks (BoT) ou aplicações que requerem utilização de recursos específicos tais como instrumentos de precisão, microscópios, sensores, dentre outros. Grids de dados têm seu foco na utilização dos recursos de grid para armazenamento e recuperação de dados.

Atualmente, existem vários grids espalhados pelo mundo inteiro rodando aplicações de natureza diversa. Muitas destas aplicações, principalmente na área de negócios, requer acessos a bancos de dados, que estão dispersos geograficamente.

Muitas soluções têm surgido para amenizar o problema de localização de dados dispersos geograficamente, mas que possuem a mesma natureza funcional. O Open Grid Forum (OGF) tem um grupo dedicado apenas ao tema de integração e interoperação de dados (Database Access and Integration Services Working Group – DAIS-WG) [9]. No contexto do DAIS-WG foi definido o "Open Grid Services

Architecture - Data Access and Integration" (OGSA-DAI), modelo de integração de dados que suporta vários sistemas gerenciadores de bancos de dados, incluindo bancos de dados distribuídos. Uma outra iniciativa é o sistema AMGA, que também dá suporte a consultas a alguns sistemas gerenciadores de bancos de dados, podendo utilizar uma linguagem baseada em SQL. Ainda um outro exemplo é o GREIC, cujo modelo baseado em "peer-to-peer", permite processamento distribuído de consultas utilizando os "peers" locais. Outras soluções se dedicam à modelagem de dados baseada em ontologias, onde os dados são representados de forma abstrata, podendo ser transformados e representados em versões locais compatíveis com os sistemas gerenciadores de bancos de dados locais.

Estas soluções buscam tornar os bancos de dados disponíveis no Grid para consultas futuras. No contexto destas soluções, o usuário deve conhecer a localização das tabelas de forma explícita. Na área de Banco de Dados existem ferramentas que provêm a integração de banco de dados heterogêneos. Trabalhos como SIMS [10], Turkwilla [16] e Infomaster [15] fazem a integração de dados de forma transparente para o usuário. Atualmente não existe nenhum trabalho que faça a integração de bancos de dados em ambientes de Grid, de forma transparente para o usuário.

Por exemplo, se uma empresa e suas filiais possuem tabelas $T_1, T_2, ..., T_n$, que devem ter a mesma funcionalidade, e um usuário, em qualquer das filiais, necessitar fazer uma consulta envolvendo um dos campos destas tabelas (que podem inclusive possuir alguns campos diferentes e números diferentes de atributos), não seria necessário saber a localização das tabelas e nem saber se estas tabelas estão dispersas geograficamente. Uma solução como esta seria relativamente segura, visto que os usuários de cada filial podem ter permissões limitadas a determinados campos, registros ou a determinadas tabelas. Além disso, esta solução isola os usuários da parte física de armazenamento das tabelas.

Segundo Pacitti e outros [23], um dos principais desafios em ambientes de grids de dados está relacionado ao gerenciamento de esquemas (Schema management). Usuários deveriam poder fazer consultas de alto nível usando seu próprio esquema sem precisar conhecer um esquema global. O problema então seria permitir o mapeamento descentralizado de esquemas de um nó para o outro.

Neste trabalho, apresentamos uma solução para integração de bancos de dados em ambientes de grid, onde os usuários não precisam saber a localização física das

tabelas de dados. Cada nó possui localmente seu esquema de dados e usuários neste nó podem continuar acessando os dados utilizando este esquema e sua linguagem e sistema de bancos de dados favoritos. Porém, consultas a dados remotos também podem ser realizadas, desde que as várias tabelas remotas para aquelas consultas tenham funcionalidades semelhantes.

Para alcançar o objetivo de ter consultas únicas a partir de qualquer sítio, com certo grau de confidencialidade e restrições de permissão, utilizamos o conceito de **tabela virtual**, que pode ser visto como uma "view" de todas as tabelas existentes no sistema. Com este modelo de tabela virtual, os administradores de cada sistema de bancos de dados fornecem a um mediador todas as características das tabelas locais (campos e esquemas locais), com classes de permissão e acesso. Este catálogo guarda, então, informações acerca das máquinas que contêm as tabelas, e um mapeamento de dados virtuais em concretos e vice-versa.

Este esquema virtual tem várias vantagens:

- permite que o administrador defina permissões e políticas de acesso aos diversos usuários de cada sítio;
- permite que dados mantenham-se confidenciais;
- oferece ao usuário uma única visão dos dados;
- "esconde" do usuário a organização física das tabelas, o que pode reforçar a segurança dos dados;
- integra e disponibiliza bancos de dados heterogêneos em ambiente de grid;
- pode ser utilizado para armazenar maiores quantidades de dados, visto que as tabelas físicas estão distribuídas.

Este esquema foi implementado e testado com tabelas dispersas em 3 diferentes sítios que utilizavam 3 diferentes sistemas gerenciadores de bancos de dados (inclusive com tipos de dados diferentes). A solução proposta utiliza o modelo OGSA-DAI para fazer a comunicação nos nós do Grid. Na atual implementação utilizamos JDBC como meio de comunicação entre os 3 sítios. Os testes mostram a viabilidade da utilização de tabelas virtuais. Apesar de termos desenhado a

solução utilizando o modelo OGSA-DAI, este esquema pode ser construído em cima de qualquer das soluções propostas por OGSA-DAI, AMGA ou GREIC, que se concentram mais na parte operacional de integração com middlewares de grid e suporte a vários sistemas de bancos de dados.

Este trabalho está organizado da seguinte forma. No Capítulo 2 discutimos aspectos de grids computacionais, com sua definição e arquitetura. No Capítulo 3 discutimos os principais aspectos de DataGrids e desafios ainda por serem resolvidos. Apresentamos também as principais soluções para integração e acesso a bancos de dados em grids. No Capítulo 4 apresentamos nosso esquema de tabelas virtuais como solução para integração e disponibilização de bancos de dados em grids. No Capítulo 5 detalhes da implementação e testes de viabilidade são apresentados. No Capítulo 6 conclusões e perspectivas de trabalho futuro são apresentados.

Capítulo 2

Computação em Grids

A grande visão de Computação em Grids é apresentada como uma analogia às grades de energia onde os usuários obtêm acesso à energia elétrica através da tomada sem se preocupar de onde vem ou como essa energia é gerada [13]. Na concepção de computação, seria como se, ao necessitarmos de uma grande quantidade de poder computacional (por exemplo, efetuar um cálculo muito complexo), só precisássemos "plugar" o nosso computador pessoal em um grid que nos ofereceria os recursos computacionais necessários para receber, deste computador pessoal, todos os dados necessários para o processamento ser completado e executá-lo de forma eficiente.

Segundo Foster e outros [14], o termo Computação em Grid foi estabelecido em meados da década de 90 para denotar "uma infra-estrutura computacional distribuída para engenharia e ciências avançadas". Baker e outros [11] consideram que a popularização da Internet e a disponibilidade de computadores com alto poder computacional e redes de alta velocidade a baixo custo fornecem a oportunidade tecnológica de se usar as redes de computadores como um recurso computacional unificado. Desta forma, as tecnologias complementariam ao invés de competir com as tecnologias de computação distribuídas existentes [14, 21]. Uma definição um pouco mais precisa de "grid" é apresentada por Krauter e outros [17]: "um grid é um sistema computacional de rede escalável podendo atingir o tamanho da Internet com máquinas distribuídas através de múltiplas organizações e domínios administrativos. Neste contexto, um sistema computacional de rede distribuído é um computador virtual formado por um conjunto de máquinas heterogêneas interligadas por uma rede, onde seus elementos concordam em compartilhar seus recursos locais com os outros." Podemos dizer, então, que a infra-estrutura de grid deve prover, de forma

global e transparente, os recursos requisitados por aplicações de grande demanda computacional e/ou de dados.

Existem três principais aspectos que caracterizam sistemas de computação em grid:

- heterogeneidade (heterogeneity): um grid envolve uma multiplicidade de recursos que são heterogêneos por natureza e que podem estar dispersos por vários domínios administrativos através de grandes distâncias geográficas;
- escalabilidade (scalability): um grid pode crescer de poucos recursos para milhões. Isto levanta o problema da potencial degradação do desempenho à medida que o tamanho de um grid aumenta. Conseqüentemente, aplicações que requerem um grande número de recursos dispersos geograficamente devem ser projetadas para serem extremamente tolerantes a latência;
- dinamicidade ou adaptabilidade (dynamicity or adaptability): em um grid, a falha de um recurso é a regra, e não a exceção. De fato, com tantos recursos, a probabilidade de que algum recurso falhe é naturalmente alta. Os gerenciadores de recursos ou aplicações devem adaptar o seu comportamento dinamicamente a fim de extrair o máximo de desempenho a partir dos recursos e serviços disponíveis. Um ambiente de grid ideal irá prover acesso aos recursos disponíveis de forma homogênea de tal modo que descontinuidades físicas tais como diferenças entre plataformas, protocolos de rede e barreiras administrativas se tornem completamente transparentes.

Deste modo, deseja-se que um grid middleware torne um ambiente radicalmente heterogêneo em um ambiente virtualmente homogêneo.

Stockinger [28] classifica grids em dois grandes grupos: Grid Computacional e DataGrid. No primeiro caso, temos de certa forma uma extensão natural da tecnologia de cluster, onde grandes tarefas computacionais devem ser computadas em recursos computacionais distribuídos. Já um DataGrid trata do gerenciamento, localização e replicação eficiente de quantidades grandes de dados. Pode-se identificar pelo menos três comunidades que precisam de acesso a fontes de dados distribuídas, não considerando apenas as aplicações de data grid: (1) bibliotecas digitais (e coleções de dados distribuídos): possuem serviços para manipulação,

procura e visualização de dados; (2) ambientes de grid para processamento de dados distribuídos: permitem a execução de diversos tipos de aplicações tais como visualização distribuída e descoberta de conhecimento; e (3) armazenamentos persistentes: disponibilizam dados independentes da tecnologia de armazenamento.

O ideal é que embora com objetivos diferentes todas pudessem manipular seus dados em fontes distribuídas através de uma interface (API, Application Programming Interface) comum. Esta API deveria ser igual seja qual for a forma de armazenamento: objetos em Bancos de Dados Orientados a Objetos (BDOO), BLOBs (Binary Large Object) em Banco de Dados objeto-relacional, ou como arquivo. Mesmo dentro de uma mesma comunidade ou classe de aplicação haverá características peculiares [21].

As primeiras implementações de Computação em Grid tendiam a ser internas a uma organização ou a uma empresa em particular. Entretanto, grids "cross-organizacionais" também vêm sendo implementados e são uma parte importante da Computação em Grids e Otimização de Negócios no Futuro.

2.1 O surgimento das organizações virtuais

Considere os seguintes cenários [13]:

1. Uma empresa que necessita decidir sobre a localização de uma nova fábrica, requer um modelo sofisticado de previsão financeira de um ASP (Application Software Provider), que forneça dados proprietários históricos de uma base de dados corporativa em sistemas de armazenamento operados por um SSP (Storage Service Provider). Durante a reunião de decisão, as sinopses estão sendo executadas de forma interativa e colaborativa, mesmo que os líderes das divisões que participem da decisão estejam localizados em cidades diferentes.

2. Um consórcio industrial formado para desenvolver um estudo de viabilidade para a próxima geração de aeronaves supersônicas, responsabiliza-se por uma simulação multidisciplinar de alta precisão para toda a aeronave. Esta simulação integra componentes de software proprietário desenvolvidos por diversos participantes, onde cada um deles opera no computador de seu participante e acessa bases de dados apropriadas e outros dados disponibilizados pelo consórcio por seus membros.

3. Uma administração em crise responde a um derramamento químico utilizando o estado atmosférico local e os tipos de solo para estimar o crescimento do derramamento, determinando o impacto baseado na localização da população assim como nas características geográficas tais como rios e abastecimento de água, criando um plano de mitigação termal (talvez baseado nos modelos de reação química) e sobrecarregando a responsabilidade de emergência pessoal planejando e coordenando a evacuação, notificando hospitais, e assim por diante.

4. Milhares de físicos de centenas de laboratórios e universidades ao longo do mundo se uniram para desenvolver, criar, operar e analisar os produtos do maior detector de partículas do laboratório de energia física europeu no CERN. Durante a fase de análise, eles uniram sua computação, armazenamento e recursos de rede para criar um "DataGrid", capaz de analisar petabytes de dados.

Esses quatro exemplos diferem em alguns aspectos: o número e o tipo de participantes, tipo de atividades, duração e escala da interação e recursos que são compartilhados, mas eles têm algo em comum. Em cada caso, o número de participantes receosos com vários graus de relacionamento prévio (ou talvez nenhum) querendo compartilhar recursos para executar alguma tarefa. Além disso, um compartilhamento é mais do que uma troca de documentos: isto pode envolver acesso direto ao software remoto, computadores, dados, sensores e outros recursos. Por exemplo, membros de um consórcio podem prover acesso a seus recursos computacionais.

As organizações virtuais nascem justamente do interesse comum compartilhado por vários membros de uma comunidade. Estas organizações, então, estabelecem uma série de protocolos para utilização e compartilhamento dos recursos.

2.2 As primeiras atividades de Grid

No passado, existiu muito interesse computacional no mundo da Computação em Grids, mas também podemos notar um número de derivados da mesma, incluindo: compute grids, DataGrids, science grids, access grids, knowlegde grids, cluster grids, tera grids e commodity grids.

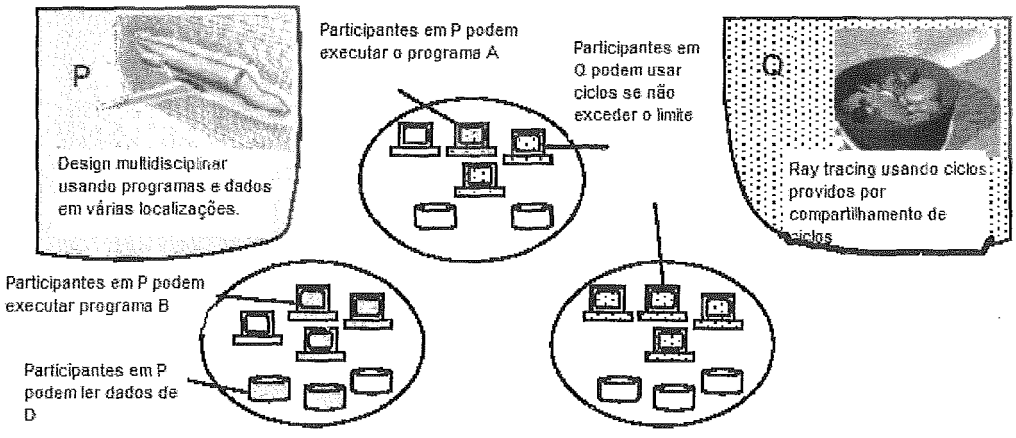


Figura 2.1: *Uma organização real pode participar em uma ou mais organizações virtuais, compartilhando alguns de todos os seus recursos. Acima, três organizações reais (círculos) e duas VOs: P, que une participantes em um consórcio de desenvolvimento aeroespacial e Q, que une colegas que concordam em dividir ciclos de computação, como por exemplo, executar computações em fila. A organização da esquerda participa em P. O da direita participa em Q e o terceiro é um membro de P e Q. As políticas que governam o acesso aos recursos variam de acordo com as organizações reais, recursos e Vos envolvidas. Figura adaptada de [13]*

O valor chave para um Grid, é baseado em méritos de negócio e na satisfação do usuário. A satisfação do usuário é uma métrica baseada pela Qualidade do Serviço provido pelo grid, como disponibilidade, performance, simplicidade de acesso, aspectos gerenciais, valores de negócio, e flexibilidade em fixar o preço. Os méritos de negócio frequentemente relatam e indicam o problema sendo resolvido pelo grid. Por exemplo: execução de jobs, aspectos gerenciais, workflows de simulação e outras tecnologias chaves - baseada em fundamentos.

Os primeiros esforços de Computação em Grids eram alinhados com as áreas funcionais de sobreposição de dados, computação e seus respectivos mecanismos de acesso. Vamos explorar abaixo essas áreas para um melhor entendimento de sua utilização e seus requerimentos funcionais.

2.2.1 Dados

Os aspectos de dados de uma Computação em Grids deve ser capaz de efetivamente gerenciar todos os aspectos de dados, incluindo alocação de dados, transferência de dados, e aspectos críticos de segurança. O núcleo funcional dos requisitos para

aplicações de Computação em Grids são [13] :

- A habilidade para integrar diversas origens de dados distribuídas, heterogêneas e gerenciadas de forma independente.
- A habilidade de prover mecanismos eficientes de transferência de dados, para prover os dados onde a computação será feita, em busca de uma melhor escalabilidade e eficiência.
- A habilidade de prover cache de dados e/ou mecanismos de replicação para minimizar o tráfego na rede.
- A habilidade para prover mecanismos de descoberta de dados, no qual irá permitir para o usuário procurar dados baseados em características do dado.
- A capacidade de implementar criptografia de dados e verificações de integridade, para garantir que o dado é transmitido na rede de uma forma segura.
- A habilidade de prover mecanismos de backup/restore e políticas, necessárias para prevenir perda de dados e minimizar o tempo que o grid fica indisponível.

2.2.2 Computação

O núcleo funcional dos requisitos computacionais para aplicações em Grids são [13]:

- A habilidade para seguir por um gerenciamento independente de recursos computacionais.
- A habilidade para prover mecanismos que podem selecionar recursos computacionais capazes de executar jobs de usuários de forma transparente e inteligente.
- O entendimento de disponibilidade de recursos, configurações de recursos dinâmica e provisionamento.
- Mecanismos de "failover" e detecção de falha.
- Garantir mecanismos de segurança apropriados para gerenciamento de recursos seguros, acesso e integridade.

Em 1998, foi dito que a "Computação em Grids" é uma infra-estrutura de hardware e software que provê acesso seguro, consistente para capacidades de alto poder computacional. Essa definição foi centrada inicialmente nos aspectos computacionais de grids. Posteriormente essa definição foi ampliada com um foco maior no compartilhamento de recursos coordenados e resolução de problemas em organizações virtuais multi-institucionais.

A qualidade e quantidade de requisitos para alguns setores de negócio, relacionados a aplicações computacionais, também vêm se tornando cada vez mais complexos. O mercado atualmente está se dando conta de que existe uma necessidade de pesquisa, e estão conduzindo vários experimentos científicos e cenários de modelagem complexos, como o processo do genoma, pesquisa astronômica, uma enorme variedade de simulações, cenários de modelagem científica. Atualmente esses requisitos podem exceder as demandas e a disponibilidade do poder computacional instalado em uma organização.

Essas aplicações de necessidade de alto poder computacional são certamente análogas ao sistema elétrico do início de 1900, tal que para prover tal disponibilidade de energia elétrica, cada usuário tinha que ter e operar um gerador. Então, quando uma grade de energia elétrica tornou-se realidade, isso mudou o conceito de fornecimento de energia elétrica. Em um ambiente similar, os grids computacionais mudaram a percepção da utilidade e disponibilidade do poder computacional. Assim, o ambiente de Grid Computacional tornou-se realidade podendo prover poder computacional confiável, poderoso e barato para seus consumidores.

Por exemplo, em um grid intensivo em dados, o foco é o gerenciamento de dados, os quais estão sendo acessados por diversos de meios de armazenamento e em localidades dispersas geograficamente. Essas origens de dados podem ser banco de dados, sistemas de arquivos e outros dispositivos de armazenamento. Os sistemas de grids devem também ser capazes de prover serviços de visualização provendo transparência para o acesso a dados, integração e processamento.

Os requisitos de dados das primeiras soluções em grids eram:

- A habilidade de descobrir dados.
- O acesso a banco de dados, usando meta-dados ou outros atributos de dados.

- A capacidade de suportar acesso a dados de forma flexível e com capacidade de filtro de dados.

As tendências atuais da Computação em Grids são arquiteturas baseadas em serviços para ambientes em grids. Essa arquitetura é construída por interoperabilidade e baseada em padrões de protocolos aberto.

2.3 As atividades atuais em Grids

Inicialmente, o foco das atividades da Computação em Grid estava nas áreas de poder computacional, acesso a dados e recursos de armazenamento [13].

A definição do compartilhamento de recursos em Computação de Grids mudou baseado em experiências anteriores, com um foco maior sendo aplicado para uma forma sofisticada de compartilhamento de recursos coordenados distribuídos através de participantes em uma organização virtual. O conceito de compartilhamento de recursos coordenados inclui algum recurso disponível em uma organização virtual, incluindo poder computacional, dado, hardware, software e aplicações, serviços de rede, e alguma outra forma de realização de recurso computacional [13].

2.4 As áreas de negócio de Grids

Um dos mais valiosos aspectos da Computação em Grid é que ela tem atraído muito os negócios. Podemos ver atualmente, o Oracle 10 G que já possui uma característica de estar trabalhando com Computação em Grids.

Em termos gerais, a utilização de Computação em Grids em ambientes de negócios provê diversos benefícios. Esses benefícios incluem [13]:

- Aceleração da implementação de horizonte temporal tendo em vista a intersecção com os resultados finais antecipados do negócio.
- Aumento de produtividade e colaboração de organizações virtuais e recursos de dados e recursos computacionais.
- Permitir departamentos dispersos de forma ampla e as empresas criarem organizações virtuais para compartilhar dados e recursos.

- Prover acesso instantâneo a recursos de dados e recursos computacionais
- Impulsionar gastos com investimentos de capital, e gastos com investimentos operacionais, os quais com foco em ajudar a ter certeza da utilização dos custos da capacidade computacional.
- Evitar armadilhas comuns de superprovisionar e arcar com custos excessivos.
- Flexibilidade robusta e infra-estruturas operacionais resiliente.

Muitas organizações começaram identificar as maiores áreas de negócio para aplicações de negócio em Computação de Grid. Algums exemplos das maiores áreas de negócio incluem:

- Ciências, por processar strings de informações biológicas e químicas.
- Serviços financeiros, por processar grandes e complexos modelos financeiros.
- Colaboração para pesquisa, por permitir procura intensiva avançada de computação e dados.
- Serviços de engenharia, incluindo a parte automotiva e de aeroespaço, pelo design colaborativo e dados - teste intensivo.
- Governo, por permitir colaboração e agilidade nos departamentos civis, militares e outras agências.
- Jogos colaborativos, por substituir os servidores em tempo real de jogos simples com paralelismo mais alto.

2.4.1 Ciências

Esse setor tem tido diversos avanços. Diversas mudanças no caminho de tratamento de medicamentos e esforços de descoberta de novos medicamentos estão sendo conduzidos. Além disso, temos o Projeto Genoma, que talvez tenha sido um dos maiores projetos que tem exigido esforços de Computação em Grids no setor de Ciências. [13]

Os esforços de Computação em Grids descobriram que os desafios nessa área incluem extensos montantes de análise de dados, movimentação de dados, cache de dados e mineração de dados.

Os Sistemas de Computação em Grid podem prover uma infra-estrutura comum para acesso de dados e ao mesmo tempo, prover mecanismos seguros de acesso aos dados. Atualmente, essa área utiliza a Computação em Grid para executar algoritmos de comparação seqüenciais e habilitar modelagem molecular, usando os dados coletados. [13]

2.4.2 Serviços Financeiros

A tecnologia e avanços no ramo dos negócios são mais notados nas áreas de tecnologia da informação. A emergência de um mercado competitivo força a satisfação do consumidor e redução do risco, como nas mais competitivas áreas financeiras. Esses objetivos agora são arquivados para os dados de mercado atual, dados históricos, complexas modelagens financeiras baseadas nesses dados e tempo de resposta curto para consulta de usuários. [13]

É nesse ponto que entra em cena a Computação em Grid, que provê a análise financeira e serviços de setores de indústria, pois o fato é que muitas das soluções para essa área são dependentes de inúmeros acessos a milhões de dados. Para isso ser feito com sucesso, essas instituições financeiras tendem a formar organizações virtuais com participação de departamentos diferentes e de outras organizações externas. Na adição do uso de recursos existentes, um sistema em grid pode prover de forma mais eficiente os resultados, pela fácil adaptação de alterar rapidamente os algoritmos pertencentes a análises financeiras. [13]

2.4.3 Colaboração para Pesquisa

Organizações voltadas para pesquisa e universidades estão praticamente voltadas à área de colaboração de pesquisas que requer análises de um grande montante de dados. Alguns exemplos desses projetos são experimentos físicos, análises da seqüência do genoma humano, entre outros.

A Computação em Grid provê mecanismos de compartilhamento de recursos por formar uma ou mais organizações virtuais provendo capacidades de compartilhamento específicas. A formação de organizações virtuais dinâmicas provê

capacidades para dinamicamente adicionar e excluir participantes de organizações virtuais, gerenciar o compartilhamento sob demanda, provisionado de um framework comum e integrar de forma segura para acessos de dados.

2.4.4 Engenharia

A Engenharia precisa de recursos para capturar dados, velocidade para análise destes e prover resposta rápida para a necessidade de mercado. Mas como sabemos, a complexidade desse setor é bastante alta.

Algumas das complexidades que podem ser vistas são:

- A análise de dados em tempo real para localizar um padrão específico em um problema.
- Os estudos parametrizados para verificar aspectos diferentes dos sistemas.
- Os experimentos de modelagem para criar novos projetos.
- As atividades de simulação para verificar nos modelos existentes a exatidão.

2.4.5 Jogos Colaborativos

Existem tipos colaborativos de disciplinas de Computação em Grid que estão envolvidas em tecnologias emergentes para suportar jogos online, enquanto utilizam provisionamento sob-demanda de recursos de computação intensiva, como computadores e redes de armazenamento. Esses recursos são selecionados baseados em requisitos, frequentemente envolvendo aspectos como volume, tráfego e número de jogadores, funcionando melhor do que com recursos centralizados.

Ambientes de jogos usando Computação em Grid são capazes de suportar cada ambiente "virtualizado" por habilitar jogos colaborativos.

2.4.6 Governo

No setor governamental, a Computação em Grid tem foco em prover acesso coordenado para montantes de dados presos às diversas agências governamentais. Isso provê um acesso rápido para resolver problemas críticos, como situações de emergência, e outras atividades normais. Esses ambientes chaves fornecem uma eficiência maior de decisão com um menor tempo. [13]

A Computação em Grid habilita a criação de organizações virtuais, incluindo muitos participantes de várias agências governamentais (estadual e federal). A formação de organizações virtuais, e dos respectivos elementos de segurança é mais desafiadora por causa dos altos níveis de segurança no governo e dos complexos requisitos.

2.5 Aplicações em Grid

Em relação a aplicações em Grids, podemos agrupá-las com as necessidades que estas têm em comum:

- Particionamento da aplicação que envolve a divisão do problema em partes.
- Descoberta e escalonamento de tarefas e workflow.
- Comunicação de dados, distribuindo os dados do problema onde e quando ele é requisitado.
- Provisionando e distribuindo códigos da aplicação para os nós de sistema específicos.
- Resultados gerenciados, auxiliando nos processos de decisão do ambiente.
- Características autônomas, como auto-configuração, auto-otimização, auto-recuperação e auto-gerenciamento.

2.5.1 Escalonadores

Escalonadores são tipos de aplicações responsáveis pelo gerenciamento dos jobs, como alocação de recursos necessários para algum job específico, particionamento de jobs para escalonar execução paralela de tarefas, gerenciamento de dados, correlação de eventos. Esses escalonadores formam uma estrutura hierárquica, com meta-escalonadores que formam a raiz e outros escalonadores de nível mais baixo. Esses escalonadores devem ser construídos com aproximação de uma implementação local de escalonador, de algum outro meta-escalonador ou um escalonador de cluster para execuções paralelas. A figura 2.2 mostra o conceito.

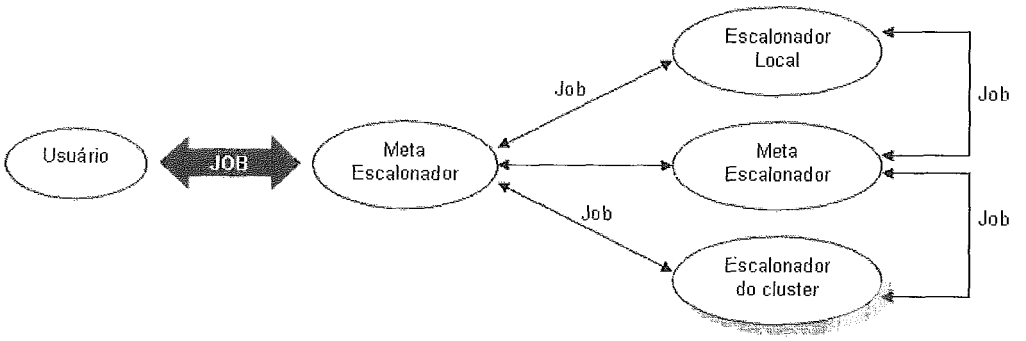


Figura 2.2: *A hierarquia dos escalonadores inclui local, meta-level e cluster. Figura adaptada de [13]*

Os jobs submetidos aos escalonadores da Computação em Grid são avaliados baseados nos seus requisitos em nível de serviço, e então alocados a seus respectivos recursos de execução.

Isso irá envolver um gerenciamento complexo de fluxo de trabalho e atividades de movimentação de dados para ocorrer em uma base regular. Existem escalonadores que devem prover capacidades para áreas como:

- Reserva de recursos avançada.
- Validação de acordo com o nível de serviço e execução.
- Gerenciamento de política de "jobs" e recursos e execução para melhores tempos de execução com restrições de orçamentos acessíveis.
- Monitoração de execução de "jobs" e status.
- Re-escalonamento e ações corretivas de possíveis situações de falha.

2.5.2 "Resource Broker"

O "resource broker" provê serviços de "emparelhamento" entre o requisitante do serviço e o provedor do serviço. O emparelhamento habilita a seleção dos melhores recursos de um provedor de serviços para a execução de uma determinada tarefa. Esses "resource brokers" coletam informações como: disponibilidade, modelos de uso, capacidade e informações de custo.

Os processos de emparelhamento em um resource broker envolvem alocação e funções de suporte como:

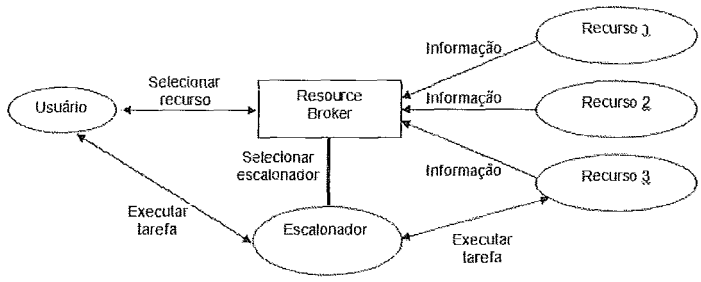


Figura 2.3: *O resource broker coleta informações dos respectivos recursos, e usa a origem da informação no processo de emparelhamento. Figura adaptada de [13]*

- Alocação do recurso apropriado ou uma combinação de recursos para a execução de uma tarefa.
- Suportar deadline de usuários e restrições de orçamento para otimizações de escalonamento.

2.5.3 Balanceamento de Carga

A característica de balanceamento de carga deve estar sempre integrada em um sistema de Computação em Grid, para evitar atrasos de processamento e má utilização de recursos. Esses tipos de aplicações devem ser feitos junto com os escalonadores e gerenciadores de recursos.

Existem casos onde reservas de recurso podem ser requisitadas, como por exemplo, a execução de vários jobs em paralelo.

Uma outra característica que é do interesse do balanceamento de carga é o suporte à detecção de falha e gerenciamento. Esses distribuidores de carga podem redistribuir os jobs para outros recursos, se necessário.

2.5.4 Portais de Grid

Os portais de Grid são similares aos portais Web, no sentido de prover acesso uniforme aos recursos de grid. Esses tipos de portais ajudam a aliviar a complexidade do gerenciamento de tarefas através de interfaces gráficas customizadas e personalizadas para os usuários.

Alguns exemplos das capacidades de um Portal de Grids seguem:

- Consulta a banco de dados ou servidores LDAP (Lightweight Directory Access Protocol) para informações específicas de recursos.
- Facilidade de transferência de arquivos como upload, download, integração com software personalizado etc.
- Gerenciamento de jobs através de resposta de status.
- Alocação de recursos para a execução de tarefas específicas.
- Gerenciamento de segurança.
- Prover soluções personalizadas.

Um exemplo prático de um Portal de Grids é o GridSphere [1], que é um framework com código-aberto.

Um dos elementos chave no Gridsphere é a habilidade para os administradores do sítio e usuários individuais configurarem o conteúdo dinamicamente.

2.6 A Infra-estrutura de Grids

A infra-estrutura de Grids forma o núcleo para que aplicações em Grid tenham sucesso. Essa infra-estrutura é uma combinação complexa de um número de capacidades e recursos identificados por algum problema específico e do ambiente endereçado.

Os provedores de serviço devem considerar as seguintes questões, a fim de identificar o núcleo de suporte de infra-estrutura requisitado pelo ambiente:

- Quais problemas tentaremos resolver para o usuário?
- Quão difícil é usar a ferramenta de Grid?
- Quais são os padrões abertos, ambientes e regulamentos de provedores de serviços de grid que devem ser abordados?

Nos primeiros estágios da computação em Grids, diversas soluções específicas e middlewares foram desenvolvidos para resolver problemas em Grids. Atualmente, com o surgimento e a convergência de tecnologias de grid orientadas a serviços, incluindo soluções baseadas em XML, e provedores industriais oferecendo soluções de grid reutilizáveis, torna-se menos complicado implementar tais soluções.

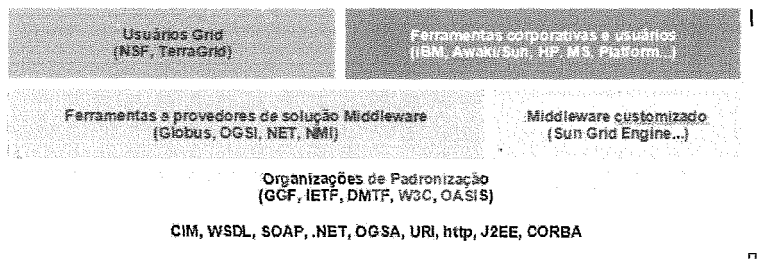


Figura 2.4: *A classificação básica das organizações de Computação em Grid. Figura adaptada de [13]*

2.6.1 A Organização da Computação em Grid

A organização da Computação em Grids e suas atribuições podem ser classificadas em quatro categorias, baseadas em suas funções na Computação em Grid. Essas funções são:

- Organizações desenvolvendo padrões de Grids e guias de boas práticas.
- Organizações desenvolvendo toolkits, frameworks e soluções de middleware.
- Organizações usando soluções baseadas em grids, para resolver seus problemas computacionais, dados e requisitos de rede.
- Organizações trabalhando para adotar conceitos de grid em produtos comerciais, através de computação utilitária e "Business On Demand computing" (comércio sob demanda).

Existem muitas organizações no mundo aprimorando e inovando ambientes de Computação em Grid. A IBM Corporation é a líder em ambientes de comércio sob demanda e a própria corporação tem habilitado um ambiente deste tipo. A IBM atualmente está trabalhando com um grande número de clientes globais em um mesmo esforço.

Organizações desenvolvendo padrões de Grid e guias de boas práticas são responsáveis por refinar o processo de padronização de grids e definir um guia com boas práticas para o uso de grids, tanto no meio científico quanto no meio comercial.

2.6.2 Open Grid Forum (OGF)

O OGF (antigo GGF, Global Grid Forum) foi estabelecido como uma comunidade pública para a discussão de tecnologia de Grid. O OGF também fornece um meio de coordenar os esforços da tecnologia de Computação em Grid, trazendo a tona o reuso e interoperabilidade e compartilhando resultados. Atualmente, existem mais de 400 organizações envolvidas com o OGF no mundo. Isso inclui instituições de pesquisa científica, universidades e organizações comerciais.

O objetivo principal do OGF é promover e prover suporte ao desenvolvimento, implementação de tecnologias de grid e aplicações pela criação de documentações contendo especificações de boas práticas, casos de uso, arquitetura e guias de implementação.

Os objetivos do OGF são:

- Criar um processo aberto para o desenvolvimento de especificações e acordos em relação à Grids.
- Criar especificações de grid, documentos de arquitetura e guias de boas práticas.
- Gerenciar e controlar versões de documentos e especificações.
- Obedecer a propriedades políticas.
- Prover um fórum para armazenamento de informações e colaboração.
- Melhorar a colaboração entre várias pessoas envolvidas em pesquisas relacionadas em Grid, framework, instalação e usuários.
- Criar guias de boas práticas obtidos a partir da experiência das tecnologias associadas com Computação em Grid.
- Educar nos avanços de tecnologias de grid e compartilhar experiências a partir do interesse das pessoas.

Além dos objetivos do OGF, existem áreas de trabalho dentro do mesmo:

- Ambientes de aplicação e programação.

- Arquitetura.
- Dados.
- Sistemas de Informação e Performance.
- Peer-to-peer: Desktop Grids.
- Escalonamento e gerenciamento de recursos.
- Segurança.

Atualmente, uma das maiores atividades no OGF, que está atraindo a comunidade de Grids, é o modelo de arquitetura baseado em padrões abertos inspirado em serviços web, chamado de Open Grid Service Architecture (OGSA). Com padrões abertos, como a fundação e a integração do software, OGSA tem crescido como o núcleo da tecnologia de Grid para o compartilhamento de recursos futuros, principalmente com as novas dimensões comerciais que estão aderindo a soluções em Grid.

Existem organizações que estão desenvolvendo Toolkits e Frameworks relacionados a Grid, como podemos destacar o Globus, Legion, dentre outros.

Por outro lado, existem organizações comerciais usando soluções baseadas em Grid, onde, todos os recursos computacionais incluindo clusters, servidores, sistemas operacionais e aplicações são vistos como utilidades. Os avanços da Computação em Grid, através dos princípios de tecnologia aberta, integrações baseadas em padrão e maturidade na tecnologia de hardware e software estão embutidos sob esse conceito de "utilidades".

As áreas de estratégia de aplicabilidade em grid no mundo comercial são: computação utilitária, visualização de recurso e computação sob-demanda [13]. Algumas das tecnologias prominentes que vêm ajudando as organizações comerciais são:

- Avanços de arquiteturas orientadas a serviço, em particular, os Web Services, permitindo organizações para começar a trabalhar em soluções de software interoperáveis.
- Capacidades de visualizações de hardware incluindo clusters, etc.

- Capacidades de software em gerenciamento de recursos e provisionamento incluindo arquiteturas de "policy-driven" para encontrar a qualidade do serviço, uso de métricas, entre outros.

- Princípios de computação autônoma permitem a disponibilidade de recursos.

Existem algumas organizações participantes desse meio atualmente, dentre as quais podemos destacar a IBM, Avaki, Platform, Oracle, entre outras.

Capítulo 3

DataGrids

DataGrids têm sido adotados e classificados como a próxima geração de plataformas de execução e armazenamento por muitas comunidades científicas que precisam compartilhar, acessar, transportar, processar e gerenciar grandes coleções de dados distribuídas pelo mundo [32].

Datagrids [12], no início, negociavam a provisão de serviços e uma infra-estrutura para aplicações distribuídas de dados intensivos que precisavam acessar, transferir e modificar grandes conjuntos de dados armazenados em recursos de armazenamento distribuídos. Para usuários aproveitarem ao máximo benefícios da estrutura, as seguintes capacidades eram necessárias:

- habilidade para procurar através de vários conjuntos de dados disponíveis e descobrir corretamente recursos de dados para acessar os dados.
- habilidade para transferir grandes conjuntos de dados entre recursos no menor tempo possível.
- habilidade para usuários poderem gerenciar várias cópias de um mesmo dados.
- habilidade para selecionar recursos computacionais adequados e processar dados sobre eles.
- habilidade para gerenciar permissões de acesso aos dados.

Um DataGrid provê serviços que ajudam usuários a descobrir, transferir e manipular grandes conjuntos de dados armazenados em repositórios distribuídos, e, além disso, criar e gerenciar cópias desses dados. No pior cenário, um DataGrid

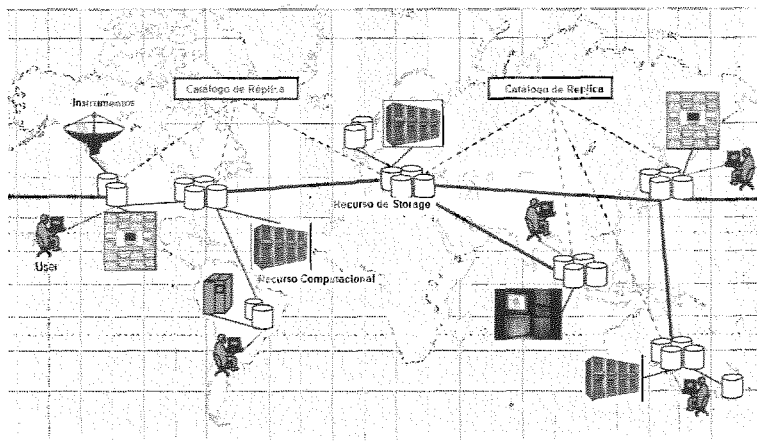


Figura 3.1: *Uma visão em alto nível de um DataGrid. Figura adaptada de [32]*

provê duas funcionalidades básicas: um mecanismo confiável de transferência de dados com alto desempenho, e, um mecanismo para gerenciamento de réplicas [12]. Todas as operações em um DataGrid são medidas por uma camada de segurança que negocia a autenticação entre entidades e garante a execução de apenas operações autorizadas.

Um DataGrid, portanto, provê uma plataforma pela qual usuários podem acessar recursos de armazenamento e recursos de rede, com o objetivo de executar aplicações intensivas em dados, sobre dados remotos, promovendo, então, um rico ambiente para usuários analisarem dados, compartilharem os resultados com seus colaboradores e manter informações de estado sobre os dados, ultrapassando barreiras institucionais e geográficas. Na figura 3.1 temos uma visão de um DataGrid.

Recursos em um Grid são heterogêneos em termos de ambientes operacionais, capacidade e disponibilidade e estão sob controle de seus próprios domínios locais administrativos. Sistemas de grid devem lidar com problemas como: compartilhamento de recursos, autenticação e autorização de entidades, e, gerenciamento e agendamento de recursos para uso de forma eficiente e efetiva dos recursos disponíveis. Naturalmente, DataGrids compartilham estes mesmos problemas, mas possuem suas próprias características e desafios:

Datasets Aplicações "data-intensive" são caracterizadas pela presença de grandes *datasets* da ordem de gigabytes. Gerenciamento de recursos em DataGrids tenta minimizar latências de transferências de grande volume de dados, criando

réplicas através de estratégias de replicação apropriadas e gerenciando recursos compartilhados.

Coleções de dados compartilhadas Compartilhamento de recursos no DataGrid também inclui compartilhamento de coleções de dados distribuídas.

"Namespace" Unificado O dado em um DataGrid compartilha o mesmo "namespace" lógico no qual todo elemento de dado tem um único arquivo lógico. O arquivo lógico é mapeado para um ou mais arquivos físicos em vários recursos de armazenamento no Grid.

Restrições de Acesso Usuários desejam confidencialidade sobre seus dados ou desejam restringir a distribuição para colaboradores próximos. Autenticação e autorização em DataGrids envolvem um controle sobre as coleções de dados compartilhadas.

Foster propôs uma arquitetura de Grid para compartilhamento de recursos sobre diferentes entidades baseados no conceito de Organizações Virtuais. Uma Organização Virtual é formada quando diferentes organizações e recursos colaboram para um objetivo comum.

3.1 Design

A existência de Organizações Virtuais impactam o projeto de arquitetura de DataGrids em várias formas. Por exemplo, uma Organização Virtual pode ser composta de uma hierarquia de Organizações Virtuais Regionais, Nacionais e Internacionais. Um Datagrid deve então, nesse caso, para compartilhamento de coleções de dados, ser guiado de acordo com o relacionamento existente entre as Organizações Virtuais que possuem cada coleção de dados.

3.1.1 Arquitetura

Os componentes de um DataGrid podem ser organizados em uma arquitetura em camadas. As camadas são:

- Construção do Grid: Consiste de recursos computacionais distribuídos (clusters, supercomputadores), recursos de armazenamento (RAID, fitas) e

instrumentos conectados por uma rede com alta largura de banda. Cada um dos recursos executa softwares como sistemas operacionais, submissão de jobs, sistemas de gerenciamento e sistemas de gerenciamento de banco de dados.

- Comunicação: Consiste de protocolos usados para consultar recursos na camada de Construção do Grid e para conduzir transferência de dados sobre eles. Esses protocolos são criados sob protocolos de comunicação como TCP/IP e protocolos de autenticação como o PKI (Public Key Infrastructure), senhas ou SSL (Secure Sockets Layer). Protocolos de transferência de arquivos como o GridFTP (Grid File Transfer Protocol), sob outros, que provêm serviços para transferência eficiente de dados entre dois recursos do DataGrid.

- Provê serviços para gerenciamento e processamento de dados no DataGrid. Os serviços principais como replicação, descobrimento de dados e submissão de jobs provê acesso transparente para dados distribuidos.

- Provê serviços específicos para usuários customizando-os para adequar ao domínio do úsuário (física, biologia, modelagem climática, etc).

Esses requisitos lideraram a comunidade de pesquisa em computação em Grid, através de fóruns como o Open Grid Forum (OGF), para adotar novas arquiteturas de serviços para Grid (OGSA), que é baseado no paradigma de Web Services. Web Services são componentes que usam mecanismos padrões para representação e troca de dados. OGSA foi construido em um Web Service usando XML (eXtensible Markup Language), e protocolo de comunicação SOAP (Simple Object Access Protocol) para criar Serviços de Grid. Um serviço de dados implementa um ou mais conjutos de interfaces básicas que descrevem o dado e provê operações para manipulá-lo [32].

3.1.2 Paradigmas

3.1.2.1 "Content Delivery Network"

O "Content Delivery Network" (Rede de Entrega de Conteúdo) consiste de uma coleção de servidores (não de origem) que tentam transferir trabalho dos servidores de origem entregando conteúdo sob seu nome [18]. Isto é, com uma "Content Delivery Network", requisições de clientes são realizadas por outros servidores distribuídos

pela Internet (também chamados de servidores na borda) que guardam em cache o conteúdo originalmente armazenado na origem (servidor original). Uma solicitação de um cliente é transferida do servidor principal para o servidor distribuído mais próximo ao cliente.

Comparando com DataGrids, no caso do "Content Delivery Network", a rede é gerenciada por uma simples entidade que tem a autoridade para adicionar ou remover nós e, eles possuem configuração estáveis. DataGrids são criados por instituições formando Organizações Virtuais com algum propósito em comum.

A ogranização de um "Content Delivery Network" é hierárquico. Já o DataGrid pode ser: monádico, hierárquico, federado ou uma combinação híbrida destes.

3.1.2.2 "Peer-to-Peer"

"Peer-to-peer" [22] são formados por agregações ad hoc de recursos para formar um sistema não centralizado, no qual cada par é autônomo e depende de outros pares para recursos, informações, e transferência de requisições. O principal objetivo de uma rede P2P é: garantir escalabilidade e credibilidade por remover a centralização de autoridade, para garantir redundância, para compartilhar recursos e assegurar anonimato. Uma entidade em uma rede P2P pode se juntar ou se retirar a qualquer momento e, por isso, estratégias e algorítimos devem ser desenhados tendo em mente a instabilidade e requisitos para escalabilidade e confiabilidade.

Grids e redes P2P geralmente não provêm forte garantia de consistência por causa do overhead de manter locks em grandes volumes de dados e da natureza da rede, respectivamente.

Redes P2P e DataGrids atualmente não possuem suporte para "recovery" e "rollback" [32]. Entretanto, os esforços são em prover suporte transacional para Datagrids, para prover tolerância a falhas para transações distribuídas. (Transaction Management Research Group (OGF)).

3.1.2.3 Banco de dados distribuídos

Um banco de dados distribuído é uma coleção de dados organizados e armazenados em diferentes sítios de uma rede de computadores. Cada sítio tem um grau de autonomia, é capaz de executar uma aplicação local e também participar da execução da aplicação global. Um banco de dados distribuído pode ser formado ainda

pela divisão de um simples banco de dados por diferentes sítios. Essa tecnologia é muito robusta. Ela provê processamento de transação distribuída, otimização de consulta distribuída, e gerenciamento eficiente de recursos. Entretanto, esses sistemas não podem ser empregados na escala corrente envisionada por DataGrids, pelas fortes propriedades que definem um Banco de Dados Distribuído (Atomicidade, Consistência, Isolamento e Durabilidade), para garantir que o estado do banco de dados permanece consistente e determinístico.

Os bancos de dados distribuídos são organizados usando o mesmo paradigma de esquema relacional como bancos de dados não distribuídos, e, o dado pode ser buscado usando SQL (Structured Query Language). Dados no DataGrid são organizados em catálogos que mapeiam a descrição lógica do dado para a representação física do mesmo.

No que diz respeito à desempenho em bancos de dados distribuídos, replicação e caching são usados para otimizar o processamento de consultas [32].

3.2 Elementos

Essa seção irá falar sobre vários aspectos de DataGrids. Um DataGrid consiste de vários elementos. O primeiro é a organização do DataGrid, que classifica os esforços no mundo em DataGrid. O próximo, detalha as tecnologias de transporte usadas em DataGrid. Um mecanismo robusto de replicação é crucial para uma boa operação de DataGrids. O último elemento caracteriza a alocação de recursos e agendamento. Todos esses elementos serão detalhados pelo ponto de vista de requisitos específicos para ambientes de DataGrids.

3.2.1 Organizacionais

A figura 3.2 mostra várias características organizacionais de projetos em DataGrids.

3.2.1.1 Modelo

O modelo é a maneira no qual as origens dos dados são organizados no sistema. Uma variedade de modelos existe para a operação de um Datagrid (Monádico, Hierárquico, Federado e Híbrido), e, serão discutidos abaixo:

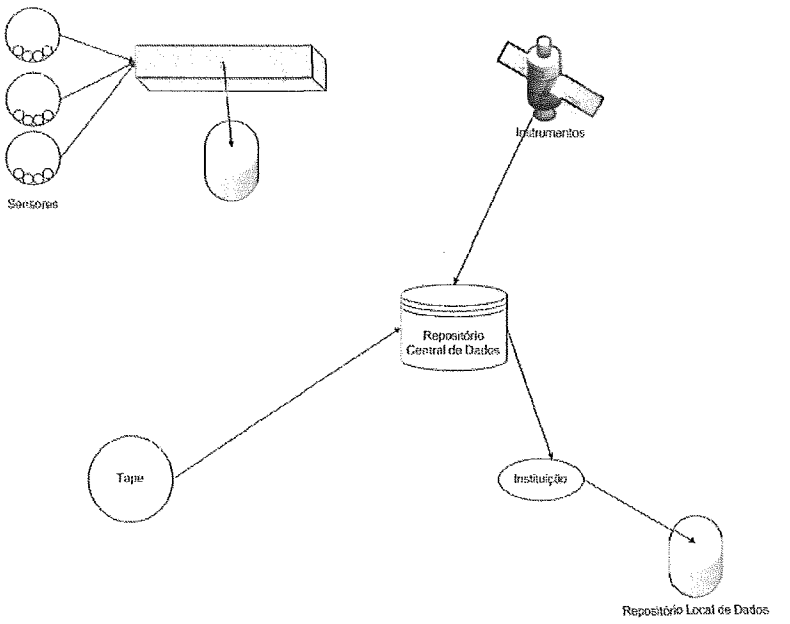


Figura 3.2: *Organização de um DataGrid de forma Monadic.Figura adaptada de [32]*

Monádico Essa é a forma geral de um DataGrid no qual todo dado é requisitado a um repositório central que responde às consultas de usuários e provê os dados. O dado pode ser de várias origens distribuídas pelo Grid, e está disponível através de acesso centralizado, como uma interface web, no qual o usuário se autentica. Esse modelo foi aplicado no projeto NEESgrid, nos Estados Unidos. [32]

A grande diferença entre esse modelo e os demais modelos é que existe somente um único ponto que irá "procurar" o dado. Esse modelo é adequado para situações onde o overhead da replicação não compensa pela eficiência ganha. Por exemplo, em situações que os dados ficam concentrados em uma determinada região.

Hierárquico Esta topologia é utilizada quando se tem uma fonte central de dados, os quais devem ser distribuídos através de colaboração pela rede.

Federado Esta topologia prevalece em DataGrids criados que querem compartilhar informações presentes em bases de dados já existentes.

Nesta estrutura, pesquisadores de uma determinada instituição podem

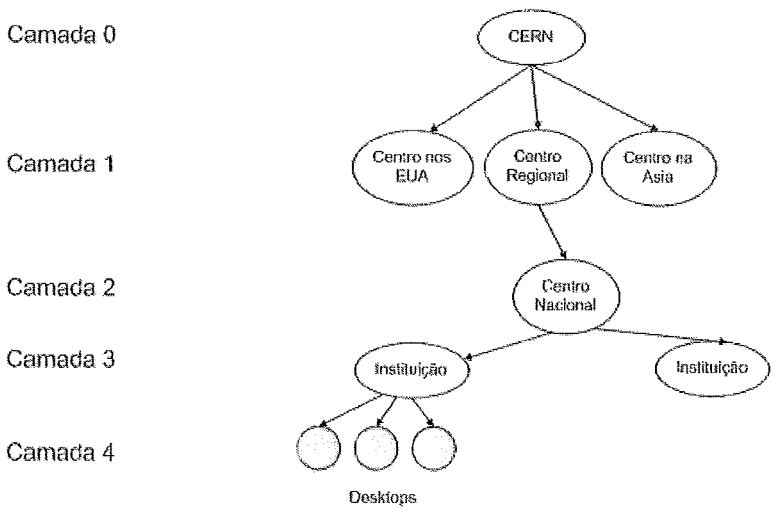


Figura 3.3: *Organização de um DataGrid de forma Hierárquico.Figura adaptada de [32]*

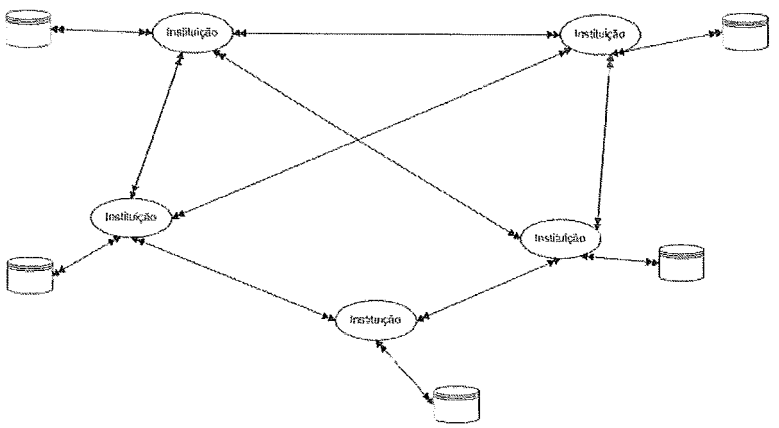


Figura 3.4: *Organização de um DataGrid de forma Federado.Figura adaptada de [32]*

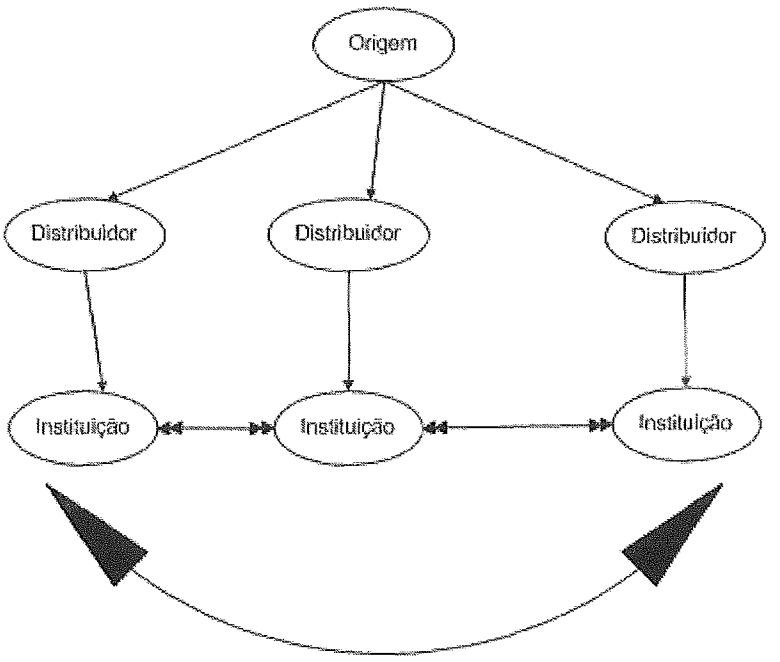


Figura 3.5: *Organização de um DataGrid de forma Híbrido.Figura adaptada de [32]*

requisitar informações de outras da rede, e as recebem, desde que tenham permissão para tal (via autenticação).

Híbrido Esta topologia combina as topologias explicadas anteriormente, de modo a adaptar os modelos de acordo com as características específicas do Grid em questão. Isso vem da necessidade dos pesquisadores conseguirem compartilhar os dados e/ou resultados de suas pesquisas.

3.2.1.2 Escopo

O escopo de um DataGrid pode variar dependendo se ele é restrito a um simples domínio (intradomain) ou se a infra-estrutura é comum para várias áreas científicas (interdomain). No primeiro caso, a infra-estrutura é adaptada para as necessidades particulares do domínio. No segundo caso, a infra-estrutura provida será genérica [32].

3.2.1.3 Organizações Virtuais

DataGrids são formados por Organizações Virtuais e, portanto, o design das Organizações Virtuais refletem na organização social do DataGrid. Uma

Organização Virtual é *colaborativa* se ela é criada por entidades que irão juntas compartilhar recursos e colaborar em um único objetivo. Existe então, um acordo implícito entre os participantes no que diz respeito ao uso de recursos. Uma Organização Virtual *regulada* será controlada por uma única organização, a qual estabelece regras para acesso e compartilhamento destes recursos. Numa Organização Virtual *baseada em economia*, provedores de recurso entram em colaborações com consumidores através da motivação do lucro. Uma Organização Virtual *baseada em reputação* será criada por entidades convidadas para unir-se em colaboração baseada no nível de serviços que eles conhecem para prover [32].

3.2.1.4 Origens de Dados

Origens de dados em um DataGrid deve ser *temporário* ou *permanente*. Um exemplo para um DataGrid temporário seria um satélite que faz broadcast de dados somente em alguns horários do dia. Muitas vezes, as aplicações precisam saber do curto tempo de vida dos dados. Muitas das implementações correntes de DataGrids usam fontes de dados permanentes, como bancos de dados de produção.

3.2.1.5 Gerenciamento

O gerenciamento de um DataGrid pode ser *independente* ou *gerenciado*. Atualmente, Datagrids requerem grande esforço de intervenção humana para tarefas como monitoramento de recursos, autorização de usuários e réplica de dados. Entretanto, pesquisas têm sido voltadas para tornar independente o gerenciamento em um DataGrid [32].

3.2.2 Transporte de dados

O mecanismo de transporte de dados é uma das tecnologias fundamentais sobre um DataGrid. O transporte de dados envolve não apenas transferência de bits entre os recursos, mas também outros aspectos de acesso a dados como segurança, controle de acesso e gerenciamento de transferência de dados [32].

3.2.2.1 Funções

Transporte de dados em Grids podem ser modelados com uma estrutura de três camadas que é similar ao modelo de referência OSI. Na camada inferior, temos o

procoloco de transferência que especifica uma linguagem comum para que dois nós em uma rede possam iniciar e controlar transferência de dados. Essa camada cuida do simples movimento de bits entre dois hosts. O GRidFTP [7] é um exemplo de protocolo para essa camada. A segunda camada é uma rede opcional de *revestimento*, que cuida da rota do dado. Essa rede provê suas próprias semânticas sobre o protocolo de Internet para satisfazer um propósito em particular. Em redes P2P, revestimentos baseados em *hash tables* distribuídas provêem um modo mais eficiente de localizar e transferir arquivos. [32]. A camada mais alta provê funções específicas da aplicação como entrada e saída de arquivos (E/S). Um mecanismo de E/S de arquivo permite uma aplicação acessar remotamente arquivos como se eles estivessem localmente disponíveis. Esse mecanismo apresenta para a aplicação uma interface transparente através de APIs que escondem a complexidade e falta de robustez de uma rede.

3.2.2.2 Segurança

Segurança é um requisito importante durante o acesso ou transferência de arquivos e deve assegurar a autenticação apropriada de usuários, integridade de arquivos e confidencialidade [32]. A segurança no transporte de dados pode ser dividida em três categorias principais:*autenticação* e *autorização* de usuários e *encriptação* de transferência de dados. Autenticação pode ser baseada em senhas ou chaves públicas simétricas ou assimétricas de protocolos como Kerberos. No que diz respeito à movimentação de dados, autorização de usuários é reforçada por mecanismos como controle de acesso ao dado que será transferido. Formas de autorização *coarse-grained* usam métodos tradicionais como permissão de arquivos UNIX para restringir o número de arquivos ou coleções que serão acessadas pelo usuário [32]. Mas, com a expansão dos DataGrids, os requisitos para uma autorização *fine grained* têm tido um maior foco [32]. Requisitos incluem restringir o número de acessos mesmo para usuários autorizados, delegar direitos de leitura e escrita para arquivos particulares ou coleções e posse flexível do dado.

3.2.2.3 Tolerância à falha

Tolerância à falha é também uma característica importante requerida no ambiente de um DataGrid, especialmente quando ocorrem transferências de arquivos de dados

grandes. Tolerância à folha pode ser subdividida em re-iniciar ("restarting over"), continuar a partir da interrupção ("resuming from interruption"), e prover caching. [32].

Re-iniciar a transferência toda ("Restarting over") significa que o mecanismo de transporte de dados não provê nenhuma tolerância à falha. Entretanto, todo o dado em trânsito poderia ser perdido e existe um sinal de overhead por estar realizando novamente uma conexão. Protocolos como o GridFTP [7] permitem a continuação da transferência a partir do momento em que esta foi interrompida ("resuming from interruption") a partir do último byte reconhecido.

Overlay networks provê o caching de transferências a partir de protocolos "store-and-foward". Nesse caso, o receptor nunca tem que esperar até que as conexões sejam re-estabelecidas. Entretanto, caching reduz o desempenho de toda a transferência de dados e o montante de dados que pode ser feito caching depende das políticas de armazenamento entre os pontos de rede intermediários [32].

3.2.2.4 Modo de transferência

Essa subseção irá descrever os modos de transferência suportados pelo DataGrid. Modos de transferência de *Bloco, Stream e Compactado* já estão disponíveis em transmissões de dados tradicionais, em protocolos como o FTP. Mas, o FTP possui limitações, pois não foi projetado para banda larga. Algumas idéias foram sugeridas para aumentar o desempenho de transferência de dados em ambientes de Grid reduzindo a latência e, consequentemente, aumentando a taxa de transferência [32]. Essas idéias são listadas abaixo:

- Transferência de dados paralela - é a habilidade para usar vários streams de dados sobre o mesmo canal para transferir um arquivo.

- Transferência de dados listrada (*striped*) - é a habilidade para usar vários streams de dados para acesso simultâneo de diferentes blocos de um arquivo que está particionado em vários nós de armazenamento (também chamados de *striping*. Essa característica distribui a carga de acesso sobre os nós e também aumenta a utilização da banda.)

- Auto-redimensionamento de buffers - é a habilidade para automaticamente fazer redimensionamento da janela TCP do transmissor e receptor e do

tamanho dos buffers, para que a banda disponível possa ser utilizada de forma mais eficiente.

- Operações de container - é a habilidade de agregar vários arquivos em um grande *dataset* que pode ser transferido ou armazenado de forma mais eficiente. Esse ganho de eficiência se traduz na redução do número de conexões requeridas para transferir o dado e também, na redução da latência inicial [32].

3.2.3 Replicação de Dados

Um Datagrid é uma colaboração distribuída geograficamente no qual todos os membros requerem acesso para os *datasets* produzidos na colaboração. Replicação de *datasets* é um requisito chave para garantir a escalabilidade da colaboração, credibilidade de acesso a dados e para preservar a largura da banda [32]. A replicação é delimitada pelo tamanho de armazenamento disponível nos diferentes sítios do DataGrid e a largura de banda entre esses sítios. Um sistema de gerenciamento de réplica garante acesso para o dado solicitado enquanto gerenciamento o armazenamento.

Um sistema de gerenciamento de réplica, consiste de nós de armazenamento os quais são ligados entre si via protocolos de transferência de dados de alto desempenho. O gerente da réplica direciona a criação e gerenciamento de réplicas de acordo com as demandas de usuários e dos elementos de armazenamento disponíveis, e mantém histórico de todas as réplicas e suas respectivas localizações.

Os elementos importantes de um mecanismo de replicação são a arquitetura do sistema e a estratégia adotada para a replicação. A arquitetura do mecanismo de replicação pode ser subdividida em categorias que serão descritas abaixo:

3.2.3.1 Modelo e Topologia

O modelo seguido pelo sistema determina o caminho no qual os nós serão organizados e o método de replicação. Um sistema *centralizado* pode ter uma réplica mestre a qual é atualizada e as atualizações são propagadas para os outros nós. Um sistema *decentralizado* ou mecanismo *peer-to-peer* pode ter várias cópias, todas elas devem ser sincronizadas com elas mesmas.

Nós em um sistema de gerenciamento de réplicas pode ser organizado em uma

variedade de topologias a quais podem ser agrupadas em três tipos: *Hierárquico, Flat e Híbrido.* A topologia hierárquica possui uma estrutura a qual atualizações são propagadas através de caminhos definitivos. Topologias flat são existentes em sistemas P2P e a progressão das atualizações são dependentes nos arranjos entre os pares. Topologias híbridas podem ser usadas quando precia-se mesclar as duas topologias [32].

3.2.3.2 Integração de Dispositivos de Armazenamento

A relação entre replicação e armazenamento é muito importante e determina a escalabilidade, robustez e adaptabilidade de um mecanismo de replicação. O sistema de replicação controla o sistema de arquivos e o mecanismo de E/S de um disco local. A replicação é conduzida ao nível de processos e é frequentemente engatilhada por uma requisição de leitura ou escrita para um arquivo localizado em uma localização remota por um programa. Um exemplo seria o NFS (Network File System), que é um sistema de arquivos distribuído, cujo propósito é prover acesso transparecente a arquivos remotos para aplicações [32].

A replicação é iniciada e gerenciada por aplicações e usuários. Como mecanismos interagem com os sistemas de "storage" através de protocolos de transferência de arquivos em um nível mais alto.

3.2.3.3 Protocolos de Transferência

Protocolos de transporte de dados usados nos sistemas de gerenciamento de réplicas são características diferenciais. *Protocolos free* para movimentação de dados como o GridFTP permite o cliente transferir dados independente do sistema de gerenciamento de réplica. *Protocolos closed* restringem acesso as réplicas para suas bibliotecas cliente. Exemplos de protocolos de transferência são: RLS (Replica Location Service), GDMP (Grid Data Mirroring Pilot), que usam o GridFTP para ser o mecanismo de transporte primário.

3.2.3.4 Metadado

Para usuários, pode ser difícil, quase impossível identificar *datasets* específicos dentre milhares que estarão presentes em uma coleção grande distribuída. A partir dessa perspectiva, existindo propriedades de metadados definidas, a partir do dado

replicado, usuários podem consultar *datasets* baseados em atributos que são mais fáceis para os mesmos. Metadados podem ter dois tipos de atributos: *dependente de sistema*, o qual consiste de atributos do arquivo como data de criação, espaço em disco, localização física, e, *definido por usuário*, o qual consiste de propriedades que são dependentes do experimento ou Organização Virtual que o usuário está associado [32]. Por exemplo, em um experimento de Física, o metadado poderia descrever atributos como data do experimento, modelo de produção e tipo de evento. O metadado pode ser ativamente atualizado pelo sistema de gerenciamento de réplicas ou então atualizado passivamente por usuários quando eles criarem novas réplicas modificando as existentes ou adicionando um novo arquivo para o catálogo.

3.2.3.5 Propagaçao de Atualização

Falando em DataGrids, geralmente quando um dado é atualizado em um sítio, as atualizações são então, propagadas para o restante das réplicas. Essa atualização pode ser *síncrona* ou *assíncrona*. Enquanto o modo síncrono é muito utilizado em banco de dados, não é praticado em DataGrids devido ao custo dos protocolos de *locking*, e à frequente movimentação de grande massa de dados requisitada.

3.2.3.6 Organização de Catálogo

Um catálogo de réplica pode ser distinguido na base de sua organização. Um catálogo pode ser organizado como uma *árvore* no caso do LDAP (Lightweight Directory Access Protocol) ou baseado em catálogos como o Globus Replica Catalog.

Estratégias de replicação determinam quando e onde criar uma réplica do dado [32]. Essas estratégias são guiadas por fatores como demanda do dado, condições de rede e custo de transferência. As estratégias de replicação podem ser categorizadas em:

Método Essa classificação é baseada em duas estratégias: *estática* ou *dinâmica*. Estratégias dinâmicas adaptam as mudanças em função da demanda, largura de banda e disponibilidade de recursos, mas induzem overhead por causa do grande número de operações que eles têm que executar em intervalos regulares ou irregulares.

Granularidade Essa classificação é relacionada ao nível da subdivisão do dado

que a estratégia trabalha. Estratégias de replicação que lidam com vários arquivos ao mesmo tempo na granularidade de *datasets*. O próximo nível de granularidade é quando existem arquivos individuais, e o último nível de granularidade é em subdivisões menores de arquivos, chamadas de fragmentos.

Função objetivo A terceira classificação é relacionada com a função objetivo da estratégia de replicação. São eles: Localização, Popularidade, Custo de Atualização, Econômico, Preservação e Publicação.

3.3 Integração de Banco de dados

3.3.1 Integração de Dados em Banco de Dados

Integração de dados em Banco de Dados é o problema de combinar dados que residem em diferentes origens, e, prover ao usuário uma visão unificada desses dados. O problema de desenhar sistemas de integração de dados é importante nas aplicações mundiais recentes, e, é caracterizado por um número de problemas que são de interesse de um ponto de vista teórico [19].

Existem diversas ferramentas de Integração de Banco de Dados Heterogêneos, acadêmicas e comerciais, que cuidam da "virtualização" para o usuário na integração de dados. Abaixo iremos falar sucintamente sobre três ferramentas: O Infomaster [15] de Stanford, O Tukwilla [16] e o SIMS [10].

O Infomaster provê acesso integrado para origens de informações heterogêneas distribuídas, permitindo ao usuário uma ilusão de um sistema de informação homogêneo e integrado. Efetivamente, o infomaster cria um *data warehouse* virtual de suas origens. O usuário não precisa ser um especialista em acessar vários bancos de dados e o dado não precisa ser copiado para uma localização centralizada. O dado, entretanto, precisa ser guardado em *cache* por razões de desempenho [15].

O centro do Infomaster é um facilitador que determina quais origens contêm a informação necessária para responder a consulta de forma eficiente, desenha uma estratégia para responder a consulta, e, faz traduções para converter a informação original para uma forma comum ou na forma requisitada pelo usuário [15].

O Infomaster é usado desde 1995 para aluguel de casas na área de São Francisco, e, desde 1996, para alocação de salas em Stanford. Ele é a base para o projeto SIN

(*Stanford Information Network*) que está integrando várias origens de informações estruturadas no campus de Stanford. O Infomaster é também base do *Housewares Virtual Catalog*, uma prova de conceito com participantes de Corning, Mirro, Regal, Sears, GE Information Services, National Housewares Manufacturers Association, National Retail Federation, Stanford University, e Epistemics [15].

O Projeto Tukwila na Universidade de Washington é um sistema integrador de dados, o qual tenta responder consultas feitas através de origens heterogêneas e autônomas. Todas essas origens de dados são mapeadas em um esquema mediador comum. O sistema de integração de dados tenta reformular a consulta em uma série de consultas sobre as origens de dados, e, então, combinar o dado em um resultado comum [16].

O objetivo do Tukwilla é suportar de forma eficiente processamento de consultas de dados XML usando técnicas de processamento de consultas adaptativas, incluindo visualização de resultados incrementais e o compartilhamento dos sub-resultados através de consultas *tukwilla.*

O SIMS é um mediador de informação que provê acesso e integra diversas origens de informação. Consultas são expressas em uma linguagem uniforme, independente da distribuição da informação sobre as origens, das várias linguagens de consulta, da localização de origens, etc. O SIMS determina qual origem de dados usar, como obter a informação desejada, como e onde armazenar temporariamente a informação e manipular dados, e, também determina como eficientemente retornar a informação [10].

3.3.2 Integração de Dados em Grids

Um serviço de acesso a dados é tido como um serviço web que implementa uma ou mais interfaces especificadas pelo WS-DAI para prover acesso a recursos de dados. Um serviço de acesso a dados é um tipo de serviço de dados, onde o último deve adicionalmente incluir serviços para gerenciamento de recurso de dados ou movimentação de dados. As especificações provêm um *web service* baseado em uma *framework* de acesso a dados, expondo técnicas existentes de acessos a dados já existentes [6].

Existem diversos problemas em prover serviços de bancos de dados em Grid. Abaixo citaremos alguns deles: [24]:

- Sistemas de bancos de dados externos devem permitir conexões do Grid. Os problemas técnicos associados podem ser ultrapassados, mas o desafio político de persuadir os donos dos dados para permitir acesso a um grande número de usuários remotos pelo grid pode inibir o desenvolvimento;

- Quando um serviço de banco de dados de Grid é usado como um serviço transiente, a instalação se transforma em um problema. Sistemas atuais requerem administradores de sistemas e administradores de banco de dados que configurem serviços transientes, estabelecendo parâmetros operacionais.

- Um sistema de banco de dados com um Serviço de Acesso a Dados em Grid é baseado podendo ser vulnerável a uma sobrecarga deliberada.

Nas próximas subseções listamos os principais projetos de DataGrids existentes na literatura e mais relacionados com este trabalho. O primeiro projeto é o AMGA, o qual foi desenvolvido como parte do projeto europeu EGEE (Enabling Grids for E-science), e faz parte da distribuição do "middleware" gLite. A seguir, apresentamos o GREIC [8], e por último, o OGSA-DAI.

3.3.3 AMGA

DataGrids frequentemente possuem milhões de arquivos dispersos sobre diversos sítios de armazenamentos. Para localizar os arquivos de interesse, usuários e aplicações precisam de um mecanismo eficiente para descobrir e consultar informação sobre o conteúdo dos mesmos. Esse meio é provido por associar atributos descritivos (metadados) para arquivos e por expor a informação em catálogos, os quais podem ser consultados para localizar arquivos baseado nos valores dos atributos [26].

Um serviço de metadados para uso no ambiente de grids deve satisfazer alguns requisitos específicos:

- Deve expor uma interface completa, porém simples, para que usuários não técnicos possam utilizá-la.

- Deve ser flexível e suportar esquemas dinâmicos.

- O serviço deve também permitir que o metadado seja estruturado como uma hierarquia de coleções lógicas, então metadados relacionados podem ser agrupados juntos e isolados de outros metadados.

- Deve ser desenhado para ser escalável.
- Segurança é requerida para prover diferentes níveis de acesso para diferentes usuários.

O AMGA é uma interface para serviços de Metadados que foi desenvolvido baseado na experiência com teste de um número de serviços de metadados usados por vários colaboradores no CERN. Esses serviços possuem metas similares e foram feitos em conceitos similares, mas, possuem interfaces incompatíveis e esquemas estáticos desenhados para um domínio de aplicação específica, limitando o reuso em outros domínios. A interface do AMGA generaliza a funcionalidade desses serviços de Metadados em uma interface genérica e coerente, adaptado para vários domínios de aplicação.

Os conceitos básicos da interface de metadados são: entradas, atributos e esquemas. A *entrada* é o nome do ítem de dado ou recurso sendo descrito. Um *atributo* é o par (chave, valor) com informação de tipo, e um *esquema* é um grupo lógico de atributos. Entradas são associadas com um ou mais esquemas e herdam os atributos definidos em seus esquemas. Esse é a única forma de associar atributos a uma entrada, não é possível ter atributos associados a uma entrada diretamente. A interface define operações para adicionar ou remover entradas de um esquema, e para listar os esquemas aos quais uma entrada pertence. Além disso, entradas não podem existir por elas próprias, elas devem ser criadas com ao menos um esquema associado.

Esquemas são definidos dinamicamente pelo usuário. Existem operações para criar e excluir esquemas, como também, para criar e remover atributos de um esquema. Uma vez que os esquemas são dinâmicos, o AMGA provê métodos para descobrir os atributos de um esquema em tempo de execução. Esquemas são os blocos básicos para estruturar e organizar metadados como um grupo lógico.

Para fazer operações de consulta, é utilizada a SQL como linguagem, porque muitas das implementações usam bancos de dados relacionais como back-end, e, muitos dos usuários conhecem bem SQL. A interface de metadado não restringe o tipo de back-end de armazenamento.

Um problema é como tratar o tamanho dos *resultsets*. Uma implementação ingênua irá ler do back-end todos os resultados de uma consulta em uma operação

simples e enviar para o cliente uma mensagem. Isso não escala o tamanho do *resultset* por requisitos de memória. Para endereçar esse problema, a interface usa "interactors" para devolver respostas em pequenos pedaços [26].

A implementação do AMGA usa um modelo de sistema de arquivos para estruturar o metadado. Esquemas funcionam como se fossem diretórios: contêm entradas e outros esquemas, permitindo usuários criarem uma estrutura hierárquica.

O Controle de Acesso é feito baseado em diretórios, com todas as entradas em um diretório compartilhando a mesma lista de ACL.A implementação também suporta grupos de usuários. Outras características de segurança são autenticação baseada em certificados, certificados de grid ou senhas, e, conexões seguras usando SSL [26].

O AMGA foi desenvolvido para usar bancos de dados relacionais como meio de armazenamento. Cada diretório é uma tabela, entradas são linhas e atributos são colunas. Atributos são adicionados ou removidos de diretórios por adicionar ou remover colunas da tabela de diretórios. Uma tabela mestre mantém o índice de todos os diretórios juntos com alguma propriedade (por exemplo, ACLs). A estrutura é flexível e eficiente [26].

O protótipo foi implementado como um servidor multi-thread C++. O back-end é modular, suportando vários sistemas de armazenamento de uma forma modular. Muitos dos módulos de armazenamentos desenvolvidos foram para banco de dados relacionais, incluindo PostgreSQL, Oracle, MySQL e SQLLite [26].

O front-end suporta dois protocolos de acesso: SOAP e TCL Streaming.

Muitas aplicações estão usando ou já usaram a implementação de AMGA, seja para avaliação, ou seja para produção. A colaboração LHCb tem avaliado a implementação de AMGA usando sua informação de bookkeeping (20 milhões de entradas, 15 GB).

3.3.4 GREIC Data Gather Service

O GREIC Data Gather Service (DGS) [8] foi desenvolvido com o Grid Relational Catalog GREIC Project, o Centro de Tecnologias Computacionais Avançadas (CACT) da Universidade de Lecce.

A arquitetura do DGS foca em integrar de forma transparente nós de um grid distribuídos geograficamentes (através de nós DGS conectados de forma P2P), tentando desconectar aspectos de rota de acesso a dados e então, definindo uma

arquitetura mais geral e robusta.

Os desafios que o DGS teve são:

- *Modelo de arquitetura*: A solução proposta foi baseada em Peer-to-peer. Essa alternativa ajuda evitando gargalos e promove: escalabilidade, autonomia, expandabilidade, gerenciabilidade e confidencialidade.

- *Interoperabilidade*: Um ambiente de grid é heterogêneo em termos de hardware e software. Programas, serviços e protocolos implementados por diversos desenvolvedores não podem se comunicar sem existir padrões em comum.

- *Segurança*: O sistema proposto tem que prover acesso seguro para recursos. Autenticação mútua entre atores do sistema, autorização de processos baseados em Lista de Acesso de Controle (ACL), mecanismo de delegação de dados, gerenciamento de credenciais de proxy e criptografia de dados.

- *Desconectando recursos/roteamento*: Um problema importante é desassociar aspectos de roteamento de acesso. Dessa forma o mesmo overlay da arquitetura de grid pode ser aplicado com pequenas alterações (relacionados à drivers específicos de recursos de dados) em diferentes cenários envolvendo recursos diferentes de dados. Essa direção pode ser atingida por: introduzir o conceito de origem virtual e definindo interfaces (operações) conectadas).

- *Desempenho*: Todos os módulos foram escritos em C para endereçar desempenho.

- *Pesquisa distribuída*: O DGS permite o usuário submeter uma consulta com *targets* diferentes, *deadline* e modo (síncrono/assíncrono).

A arquitetura do DGS é composta pelos seguintes componentes:

1. DGS - atua como um par - nó intermediário pertencendo a uma Organização Virtual específica - na arquitetura P2P.

2. Origens de dados - arquivos texto, componentes de armazenamento, bancos de dados, etc.

3. Client Applications - interfaces gráficas e de linha dc comando. Permite interatividade com a arquitetura DGS P2P para: submeter consultas, retornar resultados, gerenciar usuários e grupos.

O GREIC Data Service representa o componente chave para a arquitetura proposta em P2P. Ele provê um ponto de entrada para submissão de consultas e interage com outros pares DGS para fazer ações de routing e controle. O serviço é composto dos seguintes componentes:

- *Server Front-End*: fica escutando as requisições de cliente as despachando para o executor próprio. Depende do protocolo GSI e provê segurança garantida pelo certificado X509v3 e Lista de Controle de Acesso (ACL).

- *Gerenciamento de Meta-dado*: provê funcionalidades básicas de acesso e gerenciamento do Catálogo de Metadado relacional do DGS, que contém toda a informação sobre usuários, grupos, organizações virtuais, consultas, origem de dados locais, projectos, controle de acesso, vizinhos, listas, auditoria, etc. Atualmente ele roda em PostgreSQL, MySQL, SQLite, Unix ODBC ou Oracle DBMS, usando a GREIC Standard Database Access Interface, que provê acesso transaprente e uniforme para base de dados relacionais.

- *Gerenciamento de InterGather*: cuida de roteamento e ações de controle. Roteamo significa: rotear mensagem para outros DGS, e, reunir resultados parciais vindos de várias origens de dados e outros vizinhos DGS. Ações de controle significam: troca de informação entre DGSs para verificar quando um nó está vivo ou não, e, enviar mensagens de gerenciamento de membership para registrar um par no sistema, e, gerenciar a sua lista de vizinhos.

- *Gerenciamento de Consulta*: Realiza atividades de acesso a dados. O DGS tem a responsabilidade de interagir com origens locais, e, interage com diversas origens de dados dependendo da biblioteca SRAI, o qual provê acesso transparente e acesso uniforme para origens de dados heterogeneos. Provê binding dinâmico das seguintes origens de dados e serviços grids: Bancos de Dados XML, GREIC Data Access (serviço GREIC para acesso a bancos de dados relacionais), GREIC Data Storage (serviço grid para gerenciar storage de

workspaces) e arquivos de configuração GREIC Data Gather (para operações de monitoramento).

O GREIC DGS é um web-service desenvolvido explorando o gSOAP Toolkit [8]. Para garantir um canal de comunicação seguro, foi implementado o suporte a Grid Security Infrastructure (GSI), disponível no plug-in gSOAP.

Trabalhos futuros do GREIC DGS estão voltados para melhorar routing e tolerância a falha.

Atualmente ele é usado com a GILDA t-Infrastructure para atividades de treinamento, também usando no Centro EuroMediterrâneo para Mudança Climática, e, atualmente ele faz parte da release GILDA.

3.3.5 OGSA-DAI

O projeto Open Grid Services Architecture - Data Access and Integration (OGSA-DAI) construiu um middleware composto de implementação de interfaces e serviços para acessar e controlar data sources e sincronizá-las. Na atual fase do projeto, está restrito para bancos de dados relacionais e XML. A *framework*, foi projetada para permitir outras origens de dados como sistema de arquivos para ser acessado através da mesma interface. [9]

Os serviços e interfaces definidos no OGSA-DAI herdam das especificações definidas na especificação do Open Grid Services Infrastructure (OGSI). Os serviços do OGSA-DAI provêm as primitivas básicas para construir serviços de alto nível sofisticados, que permitem federação de dados e consultas distribuídas em uma Organização Virtual. [9].

O OGSA-DAI está proximamente afiliado ao Open Grid Forum (OGF), no Grupo de Trabalho*Database Access and Integrafion Services.*

O projeto OGSA-DAI está desenvolvendo serviços Grid que representam recurso de dados, onde, um recurso de dados é uma entidade lógica que é capaz de originar ou sincronizar dados [9]. O Termo recurso de dados é usado para representar os aspectos e capacidades que são expostas ao Grid.

Os objetivos do OGSA-DAI são:

- Prover uma exposição controlada de um recurso de dados físico ao Grid.

- Suportar acesso a recurso de fontes de dados heterogêneos através de um estilo de interface comum enquanto trabalhando em mecanismos de consulta.
- Prover serviços base que permitem integração de dados a ser construído (Por exemplo, processamento distribuído de consulta, e, serviços de federação).
- Alavancar a infra-estrutura de Grid para segurança, gerenciamento, etc.
- Padronizar interfaces de acesso a dados através do Grupo de Trabalho do Global Grid Forum.
- Prover uma implementação com referência na especificação DAIS.

O acesso aos recursos de dados é feito através de serviços Grids.

Serviços Grids devem ser construídos com a capacidade de representar os diversos aspectos internos de um recurso de dados. Um serviço de Grid apresenta alguma visão de um recurso de dados. Um documento de consulta é submetido ao serviço Grid, e, avaliado para produzir um documento com resultado, que é geralmente retornado ao cliente.

O OGSA-DAI, estendendo as interfaces definidas em OGSA, introduziu os seguintes serviços:

Grid Data Service-GDS Representa uma sessão cliente com um recurso de dados físico. Um GDS é criado ou instanciado a partir de um GDSF.

Grid Data Service Factory-GDSF É definido para representar o ponto de presença de um recurso de dados no Grid. É através das instâncias do GDSF que um recurso de dado físico é exposto.

DAI Service Group Registry-DAISGR As instâncias e capacidades do GDSF devem ser localizadas no Grid através do uso de um DAISGR, o qual GDSF devem registrar para expor suas capacidades.

O OGSA-DAI somente trabalha com recursos de dados configurados estaticamente. Não existe mecanismo para dinamicamente expor recurso de dados uma vez que o container de serviços já esteja iniciado.

A instância de um GDSF expõe as capacidades de um recurso de dados no Grid. Algum cliente que desejar interagir com o recurso de dados deve instanciar um GDS. Um GDS atua como uma seção com o recurso de dados físico.

O OGSA-DAI não define nenhuma nova linguagem de consulta: o GDS está atuando através das linguagens de consultas existentes que deverão "rodar" diretamente no recurso físico.

O acesso a dados é o primeiro foco do OGSA-DAI. O escopo do projeto também envolve Integração de Dados. No contexto de Grid, integração de dados significa aplicar virtualização. [9].

As fases futuras do OGSA-DAI irão focar-se em:

- Estender a funcionalidade existente do OGSA-DAI em base de prioridades agregadas;
- Prover flexibilidade suficiente para habilitar substituição de componentes tecnológicos de uma forma plug-and-play;
- Melhorar a qualidade do software em termos de segurança, desempenho e escalabilidade;
- Estender suporte para mais plataformas de software;
- Composição de serviços de alto-nível para estabelecer plataformas de programação reutilizáveis e melhorar o gerenciamento;
- Alinhar cmo o DAIS uma vez que a padronização do processo tenha estabilizado.

3.4 Discussão

O projeto Enabling Grids for E-science (EGEE) atua em prover pesquisas na academia e na indústria com acesso a recursos computacionais, independente de sua localização geográfica. A infra-estrutura do grid do EGEE-II, é composta por PCs padrões interconectados através de links de alta desempenho sob a Internet, provendo uma infra-estrutura adequável para lidar com um grande número de tarefas computacionais. [29]. Vamos a seguir, ilustrar um comparativo feito em [29], o qual fez um comparativo entre o AMGA [26], GREIC [8] e OGSA-DAI.

Foi desenvolvido um plano de teste que direcionou em verificar diferentes aspectos relacionados à instalação e uso das ferramentas. Iniciou-se com operações simples de SELECT, INSERT, UPDATE e DELETE, aumentando a complexidade da operação, por exemplo, envolvendo operações complexas de JOIN.

Para prover um ambiente adequado para os testes, foi projetado um plano de teste envolvendo 5 sítios: INFN Bari, CNAF, INFN Catania, SPACI Lecce e Observatório Astronômico INAF de Trieste. No momento que os testes foram feitos, o middleware EGEE foi o gLite versão 3.0. Cada sítio é equipado com o "gLite User Interface". Bari, Lecce e Trieste proveram as versões 3 do OGSA-DAI servidor e cliente, o servidor e cliente do GREIC DAS (Versão 2.3.1). O AMGA (versão 1.3) está instalado em Bari, Catania e Trieste. Os clientes AMGA estão disponíveis na "gLite User Interface" [29].

O hardware envolvido durante o teste foi:

- Em Bari, a CPU utilizada foi um XEON 3.06 GHz, com 120 GB de HD, 2 GB de RAM, com o Middleware gLitle 3.0.
- Em Lecce, a CPU utilizada foi um Intel Pentium (R) 3.4 GHz, com 65 GB de HD, 1 GB de RAM, com o Middleware gLitle 3.0.
- Em Trieste, a CPU utilizada foi um Xeon 1GHz, com 80GB de HD, 2 GB de RAM, com o Middleware gLitle 3.0.

O ingrediente final do teste foram os bancos de dados. O banco de dados molecular é fornecido pela comunidade de Bioinformática, que é uma tabela de 500.000 tuplas em MySQL. O banco de dados 2MASS é um catálogo Astrônomico organizado em uma tabela de 500.000.000 tuplas, em PostGreSQL. O banco de dados exemplo sakila foi desenvolvido pelo time de documentação do MySQL, e, sua intenção foi prover um esquema padrão que pode ser usado por tutoriais. O banco de dados "world" provê um conjunto de tabelas que contém informações sobre os países e cidades do mundo e é útil para consultas básicas. Esse banco de dados está em MySQL. O banco de dados dellstore foi projetado para representar uma loja de DVDs. Esse banco de dados está em PostGreSQL. O banco de dados population é fornecido pelo projeto EGG. Foram utilizados dois Sistemas Gerenciadores de Banco de dados: MySQL e PostgreSQL. O PostGreSL e MySQL foram instalados em cada sítio, provendo serviços de acesso a grid [29].

Não ficou claro no trabalho de Taffoni *et al.* [29] se as tabelas estão armazenadas em todos os SGBDs, ou, caso contrário, em que nós do grid estão as tabelas. A única informação que foi explícita no artigo é que os SGBDs estão instalados em todos os sítios.

A comparação de desempenho entre as ferramentas foi feita usando o tempo de resposta da consulta relacionada com fatores como CPU e uso de Memória no lado do cliente e do servidor. A consulta usada para SELECT foi: SELECT * from molecule where id <= N, onde N é o número de linhas com um intervalo de 1 a 100.000.

As cláusulas de UPDATE, INSERT e DELETE correpondem a execução de uma única linha na operação de insert, delete e update. Esse teste foi feito sobre o banco de dados sakilla; a tabela usada possuia 16.049 linhas. Os resultados serão reportados na figura.

Em todas as ferramentas foi possível completar o teste, exceto pela AMGA, a qual não foi possível importar o banco de dados devido a um problema em particular sobre o dialeto SQL usado [29].

Olhando sob o ponto de vista de desempenho, o AMGA e o GREIC mostraram melhor desempenho que outras ferramentas para operações que envolveram pequenos e médios *datasets*, e o GREIC executa melhor no caso de largos *datasets*. Foi verificado que o AMGA não suporta padrões SQL e consultas de join complexas entre tabelas. [29]

Os resultados, em resumo são:

- AMGA é uma ótima ferramenta para gerenciar metadado. É muito rápido e eficiente e provê características de federação e replicação. Entretando, não é desenhado para acessar bancos de dados pré-existentes, os quais devem ser importados dentro do servidor AMGA.

- GREIC é fácil de ser instalado e configurado. É bem rápido no caso de pequenos/médios/grandes bancos de dados. É fácil de usar graças a sua interface de usuário, e pela interface de grid do portal (GREIC Portal).

- OGSA-DAI apresenta muitas características avançadas que as outras ferramentas não provêm: características de federação e replicação, uma

visualização poderosa de serviços de bancos de dados heterogêneos distribuídos. A documentação é precisa, mas para obter alguma desempenho é necessário aplicar algum "hacking".

Capítulo 4

PaP Meta-data Database System

Como mencionado no Capítulo 3, muitos projetos em DataGrids estão focados em prover gerenciamento de um grande montante de dados distribuídos através de nós de um grid. Um dos desafios em DataGrids é tornar esses dados disponíveis para o usuário de uma forma transparente. O dado pode estar distribuído em bancos de dados heterogêneos, como o Oracle, MySQL, SQL Server sob limites organizacionais.

Muitos sistemas como o AMGA [26], GREIC [8] e OGSA-DAI [9] estão voltados para a integração entre o SGBD e o Grid, de uma forma não transparente para o usuário. Quando falamos "não transparente", significa que o usuário tem que explicitar sempre a tabela física, e, em alguns deles, a consulta tem que ser passada no formato que o SGBD entenda.

Muitos sistemas de integração de banco de dados já resolvem o problema da heterogeneidade entre esquemas de bancos de dados. Entretanto, não estão voltados para grids.

O enfoque do *PaP Meta-data Database System* é propor a integração das soluções existentes de Sistemas de Bancos de dados Heterogêneos a ambientes de grid.

Neste capítulo descrevemos o *PaP Meta-data Database System*, cujo enfoque é prover acesso a dados de forma transparente para o usuário onde a informação do dado está armazenada em bancos de dados relacionais e heterogêneos, distribuídos geograficamente em diferentes localidades.

O *PaP Meta-data Database System* está focado em um primeiro momento em criar um *framework* que visa integrar tabelas heterogêneas, com um mesmo fim, localizadas em bancos de dados heterogêneos que podem estar localizados em diversos pontos do GRID. Essa integração é feita criando **visões**, que poderão ser

utilizadas por usuários de uma forma transparente para acesso a diversos nós e bancos de dados no Grid. Desta forma, o usuário conseguiria fazer consultas às tabelas virtuais, sem ter nenhum conhecimento sobre a localização do dado e sem ter conhecimento sobre o SGBD responsável pelo armazenamento.

O *PaP Meta-data Database System* precisa ser construído sobre um *framework* de Grids, visto que o JDBC não consegue se comunicar com os nós dos Grids.

O objetivo do *PaP Meta-data Database System* é usar o OGSA-DAI [9] como protocolo de comunicação com os nós do Grid. Mas, como a arquitetura foi feita de forma modular, em um primeiro momento, estamos usando JDBC para mostrar a viabilidade do projeto, pela complexidade do OGSA-DAI.

Alguns anos atrás, observamos uma ótima demanda para capacidade de armazenamento de vários setores da ciência, engenharia e até mesmo da indústria. Projetos como o LHC [2], entre outros, irão gerar milhares de Petabytes por ano. Esses dados serão armazenados em repositórios distribuídos e heterogêneos que pertencem a diferentes Organizações Virtuais. Muitos desses setores não utilizam SGBDs, eles usam arquivos, muitas vezes binários em formatos específicos.

Em capítulos anteriores, vimos que um DataGrid provê serviços que ajudam o usuário a descobrir, transferir e manipular grandes *datasets* armazenados em repositórios distribuídos ou em organizações virtuais. Esses *datasets* podem ter uma natureza diferente e estar armazenados em diferentes formatos, incluindo um fomato livre ou um formato estruturado (arquivos ou tabelas de bancos de dados). Muitos sistemas de DataGrids atualmente disponíveis são baseados em busca, localização, e transferência de dados (principalmente arquivos de dados), cuja localização é conhecida pelo usuário. Por exemplo, no caso de compatilhamento de arquivo, muitos sistemas requerem que o usuário explicite nomes dos arquivos e sua localização. Uma ótima exceção é o Condor, (Condor-G) [30], o qual é capaz de automaticamente transferir arquivos necessários do diretório local do usuário para sítios remotos, por aplicações interceptando chamadas de sistema relacionadas a operações de arquivos. RemoteFS [31] é uma outra contribuição considerável.

No que diz respeito a acesso a banco de dados, alguns trabalhos têm compartilhado o problema de consultar bancos de dados nos quais as tabelas estão geograficamente dispersas, de um modo uniforme e padrão.

Segundo a literatura, até o momento, a primeira tentativa para criar uma solução

para um sistema de gerenciamento de banco de dados para grids é o AMGA [26], parte do middleware do gLite. AMGA trabalha como o MySQL com a diferença que ele pode fazer consultas em tabelas, que estão remotamente localizadas. O usuário deve indicar a localização da tabela, usando seu Logical File Name (LFN), criado antes manualmente. Tabelas precisam estar no formato AMGA em todo nó do grid, e todos os nós do grid devem ter o serviço do AMGA instalado.

O GREIC [8] é uma outra iniciativa, a qual usa Data Gather Services (DGS), em uma arquitetura peer-to-peer (P2P), para integrar dados de bancos de dados localizados remotamente. GREIC, como o AMGA, requerem novos softwares instalados nos sítios remotos.

Nesse trabalho, apresentaremos uma nova forma de gerenciamento de consultas no contexto de grids, onde não existe a necessidade de instalação de novos softwares nos sítios remotos e onde o usuário final não precisa conhecer a localização das tabelas, tipos de dados ou qual sistema gerenciador de banco de dados é usado, e, ainda, se ele estará sendo usado remoto ou local.

O *PaP Meta-data Database System* é um *f*ramework desenvolvido em Java, que permite organizações virtuais compartilharem dados de uma forma organizada, criando Tabelas Virtuais. Ele também permite usuários fazer consultas desses dados de uma forma transparente através das Tabelas Virtuais.

Os usuários não têm conhecimento de Organizações Virtuais, ou como o dado está organizado através do Grid. Eles apenas precisam ter conhecimento das Tabelas Virtuais, as quais são configuradas pelo *PaP Meta-data Database System.*

Sobre todos os projetos em DataGrids propostos recentemente, o GREIC é o que mais se relaciona com o trabalho exposto nessa dissertação. GREIC atua de forma transparente e segura integrando fontes de dados em um grid distribuido de forma heterogênea (através de nós DGS conectados de foma P2P), tentando desacoplar aspectos de roteamento de acesso a dados e então definindo uma arquitetura mais genérica e robusta. Cada nó no grid tem um DGS e ele atua como um par. No *Pap Meta-data Database System*, as consultas podem ser feitas de alguma máquina que possua um navegador, não tendo necessidade de ter instalado nenhum software relacionado ao nosso sistema. Não existe a necessidade de instalação de novos softwares em sítios remotos e o usuário não necessita saber onde as tabelas estão localizadas e se existe um grid por trás de tudo.

4.1 Arquitetura

Em sua versão atual, o modelo do *PaP Meta-data Database System* suporta três Sistemas Gerenciadores de Bancos de dados:

- Oracle XE, que é a versão gratuita do famoso e poderoso banco de dados Oracle [4].
- SQL Server 2005 Express Edition, que é a versão gratuita do SQL Server 2005 [5], da Microsoft.
- MySQL [3], que falando de versões gratuitas, é um dos mais populares.

Esses três bancos de dados foram escolhidos pela confiabilidade e respeito dos mesmos no mundo da Tecnologia da Informação, e pelo fato que muitas das Companhias de Tecnologia da Informação usam esses bancos de dados em seus negócios. Entretanto, qualquer outro Sistema Gerenciador de Banco de Dados pode ser facilmente adicionado ao *Pap Meta-data Database System.*

O modelo do *PaP Meta-data Database System* não foca somente o mundo científico, mas, também foca no mundo da tecnologia da informação. Não é nenhuma novidade que atualmente algumas companhias estão fazendo negócios com parcerias, e, até muitas vezes, processos de fusão. Por exemplo, poderia existir uma empresa A, que lida em um determinado mercado, e uma empresa B, que lida no mesmo mercado. Ambas podem estar localizadas em diferentes lugares, usando diferentes bancos de dados, e, podem decidir fazer um acordo, como construir uma empresa D, que seria a união da empresa A e B. Falando de banco de dados, ambos poderiam compartilhar entre si, as suas bases, que, mesmo estando em SGBDs distintos, e localizados em locais diferentes, poderiam ser compartilhados usando o *PaP Meta-data Database System.*

O mesmo funcionaria para o lado científico, onde, instituições trabalhando em um mesmo assunto, poderiam entre si, compartilhar informações, centralizando-as usando o *PaP Meta-data Database System*, de forma organizada.

Um exemplo de utilização do nosso esquema é mostrado na Figura 4.1, onde um usuário da aplicação, submete uma consulta usando o padrão SQL ANSI, para uma Tabela Virtual que está configurada no *PaP Meta-data Database System.* Internamente, essa Tabela Virtual é composta por uma ou mais tabelas

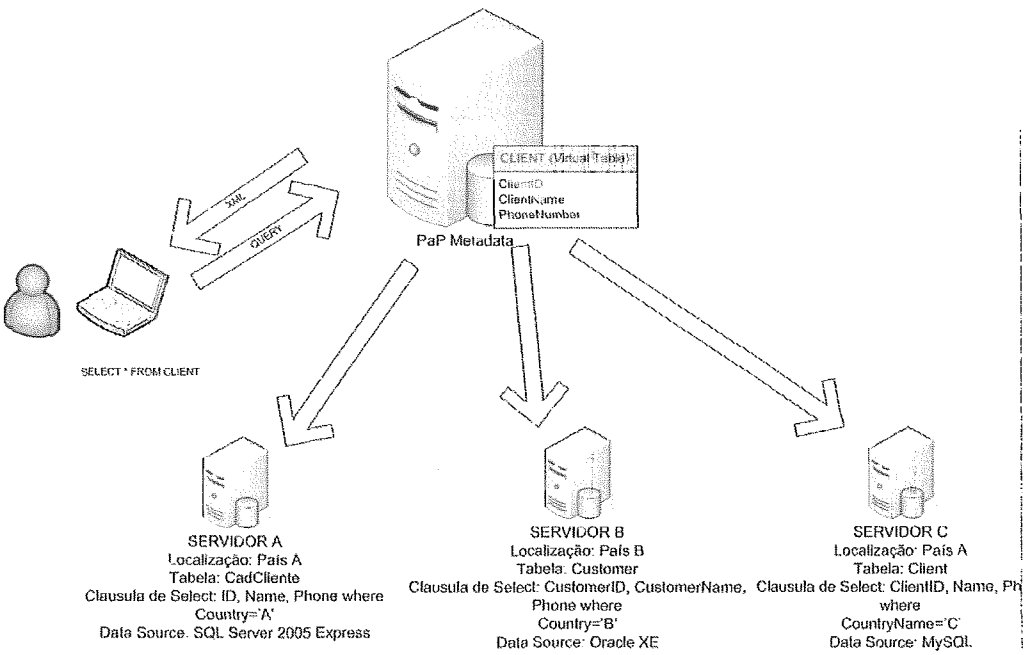


Figura 4.1: *O usuário submete uma consulta ao PaP Metadata. A consulta é feita através das Tabelas Virtuais. Após o processamento, os resultados são retornados ao usuário. Existe uma tabela virtual chamada Client, que é composta por três tabelas físicas no Grid: Uma partição da tabela virtual está localizada no Servidor A, localizado no País A, e o nome da tabela nessa localização é CadCliente, composta pelos campos: ID, Name e Phone. Somente participarão da tabela Virtual Client os registros onde o campo Country='A'. Uma outra partiçao da tabela virtual está localizada no Servidor B, localizado no País B, e o nome da tabela nessa localização é Customer, composta pelos campos CustomerId, CustomerName e Phone. Somente participarão da tabela Virtual Client os registros onde o campo Country='B'. Terminando a composição da tabela Virtual, Client é a partição localizada no Servidor C, localizada no país A, onde a tabela chama-se Client, e é composta pelos campos ClientId, Name e Phone.*

físicas localizadas em diferentes nós do Grid. Essas tabelas podem ser fisicamente armazenadas em três diferentes SGBDS (Oracle XE, SQL Server 2005 Express ou MySQL). O Front-end da aplicação irá retornar os resultados da Consulta.

No momento, a arquitetura do sistema ainda não distribui tabelas pelos nós do grid. Esse é um pré-requisito para o *PaP Metadata Database System*, que as tabelas já estejam distribuídas nos nós do grid, e particionadas nos nós do Grid. Na seção de projetos futuros, retornaremos ao ponto de um projeto futuro para desenvolver um *t*oolkit que faça a distribuição das tabelas no Grid, usando estatísticas gravadas atualmente na tabela de Histórico para ajudar a tomar decisões.

O projeto possui duas visões, e, logo, dois atores:

Visão do Administrador Nessa visão, o ator Administrador poderá ser capaz de adicionar novos nós, configurar tabelas virtuais, configurar as permissões dos usuários, e, também, fazer consultas.

Visão do Usuário Nessa visão, o ator Usuário será capaz de fazer consultas nas tabelas virtuais para as quais possui permissões, que foram dadas pelo ator Administrador na Interface Administrativa.

Uma vez que as tabelas estão distribuidas e particionadas através do Grid, o Administrador pode configurar uma Tabela Virtual. Uma tabela virtual é composta por uma ou mais tabelas físicas, que podem estar em um ou mais nós no Grid.

A configuração dos nós e as tabelas virtuais são armazenados no Banco de dados do *PaP Meta-data Database System.* Os dados que são necessários para os nós são: Host, Port, DBMS, User e Password. Se existir um *f*irewall no lado do nó, o Administrador da Rede precisa tornar disponível uma porta para acesso. A senha no sistema está usando a criptografia MD5. MD5 foi escolhida por causa das facilidades da implementação em Java dessa criptografia, e, porque ela é descrita como referência para fazer uma conexão segura ao Banco de dados. [25]

Uma vez que os nós estão configurados na aplicação por um Administrador, o Administrador pode também configurar Tabelas Virtuais, e, associar às tabelas físicas nós do Grid para compor a Tabela Virtual.

Na visão do Usuário, o mesmo não possui nenhum conhecimento sobre nós, ou alguma coisa a respeito do armazenamento físico que compõe tabelas virtuais. Na

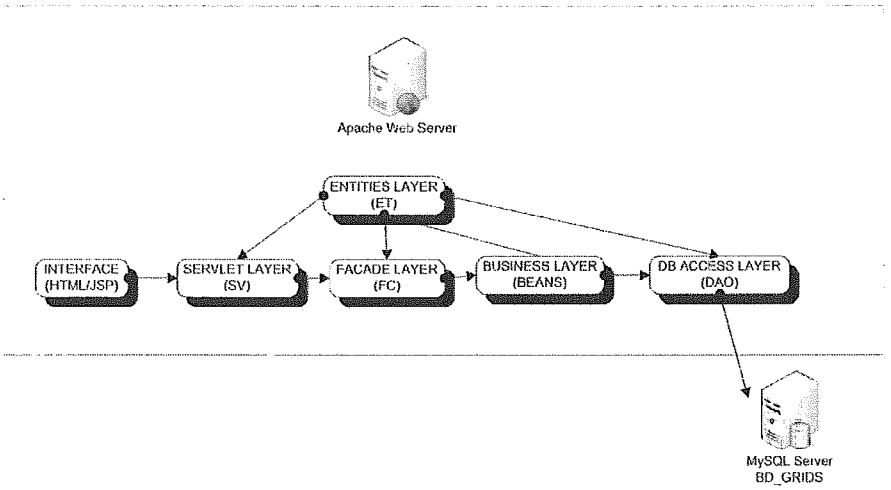


Figura 4.2: *A arquitetura da aplicação em Java*

visão do usuário, o mesmo só possui conhecimento de tabelas virtuais que o mesmo possui permissões para ver e consultar. Os nós do Grid são transparentes para o mesmo. Note bem que os acessos locais continuam sendo realizados normalmente através do SGBD local.

O usuário faz consultas através de uma interface web. A aplicação web foi desenvolvida em Linguagem Java. A escolha de uma aplicação web foi porque podemos ter uma versão centralizada, e não uma cópia local no lado do Cliente. O cliente somente precisa ter um browser como o Internet Explorer ou Mozilla Firefox instalado em sua máquina. Isso também implica em requisitos de hardware, pois, se tivéssemos que ter uma aplicação instalada em cada cliente, teríamos que garantir que todos os clientes estariam sempre usando a mesma versão, e, além disso, quando uma versão nova fosse disponibilizada, deveríamos ter uma forma de notificar todos os clientes, e garantir que todos estão usando a versão atual.

Java foi escolhido por sua popularidade no ramo das linguagens orientadas a objeto, e, pelo fato da mesma ser portátil, além disso, seu código pode estar sobre um Servidor de Aplicação Tomcat, que é gratuito, usando uma arquitetura multi-camadas, para, no futuro, termos benefícios do reuso.

MySQL foi escolhido como SGBD para gravar informações sobre a aplicação, pois, é seguro, portátil, e muito popular como um Banco de Dados gratuito.

As camadas da aplicação, mostradas na figura 4.2 são:

Interface Layer Essa camada é usada como uma interação final entre usuários e

as camadas de sistema. Essa camada foi desenvolvida usando HTML/JSP. Essa camada possui somente chamadas à servlets. Essa camada não possui nenhum contato com o banco de dados ou camada de negócio.

Entities Layer Essa camada possui uma cópia da estrutura de Banco de Dados (tabelas e atributos) usadas na aplicação. Essa camada é usada para ser passada através das outras camadas. Ela foi criada, para evitar passar por parâmetro ints, strings, bytes. Essa camada foi criada para diminuir o esforço, no caso de alterações na base de dados, visto que, uma vez tendo essa camada, a alteração seria somente nessa camada, e, não, ter que buscar todas as funções que fazem referência à tabela modificada, e modificar todas as chamadas de procedimentos.

Servlets Layer Nessa camada temos os Servlets. O servlet é um componente baseado em Java que provê um mecanismo simples e consistente para estender a funcionalidade de um web server e para acessar sistemas de negócios existentes. Servlets geram conteúdo dinâmico e interagem com os clientes web usando um paradigma de *r*equest-response. Eles têm acesso à inteira família de APIs Java. Uma vez que os servlets são servidores e independentes de plataforma, eles permitem ao desenvolvedor selecionar servidores, plataformas e ferramentas de escolha [33]. Essa camada possui instâncias das camadas de Facade e Interface. Essa camada não possui nenhum contato com o Banco de Dados.

Facade Layer Essa camada é usada para agrupar funcionalidades. Essa camada é chamada pelos Servlets, e nela existem instâncias da Business Layer. Essa camada não possui nenhum contato com o Banco de Dados.

Business Layer Essa camada possui todas as regras de negócio. Essa camada é chamada pela Facade Layer, e, nela existem instâncias da DAO Layer. Essa camada não possui nenhum contato com o Banco de Dados.

DAO Layer Essa camada é responsável pelo acesso ao banco de dados. A motivação da criação dessa camada, é, que, se, em um futuro, tenha que ser modificado o SGBD da aplicação, somente essa camada deve ser modificada,

permitindo então, deixar as outras camadas intactas, reduzindo então, o tempo de modificação.

4.1.1 Modelagem de dados

Para suportar a aplicação, como citado na seção anterior, um banco de dados em MySQL foi desenvolvido para possuir todo o armazenamento das informações sobre Tabelas Virtuais, Usuários, entre outros. Na Figura 4.3 mostramos a Modelagem de dados do sistema.

Abaixo descrevemos cada tabela no Modelo de dados e seus respectivos atributos:

DBSource Essa é uma tabela de domínio onde ficam armazenados todos os SGBDs suportados pela aplicação. Os atributos dessa tabela são: *DBSourceID*, que é o identificador da tabela, sendo chave primária, *Description*, que é o nome da origem de dados (Por exemplo, SQL SERVER 2005 Enterprise), e *Status*, que é o estado em que o DBSource se encontra. Os status possíveis são: *0 - Inativo e 1- Ativo.* Essa tabela inicialmente será preenchida com os SGBDs suportados pela aplicação. São eles: SQL SERVER 2005 Express Edition, Oracle XE e MySQL.

GlobalDataType Essa tabela é uma tabela de domínio onde ficam armazenados todos os tipos de dados suportados pela aplicação. Os atributos dessa tabela são: *GlobalDataTypeID*, que é o identificador da tabela, sendo chave primária, *Description*, que é o nome do tipo de dado (Por exemplo, int, varchar, double), e *Status*, que é o estado que o GlobalDataType se encontra. Os status possíveis são: *0 - Inativo e 1- Ativo.* Essa tabela inicialmente será preenchida com os tipos de dados suportados pela aplicação. São eles: int, varchar, double, float, long int e date time.

Action Essa tabela é composta pelos seguintes atributos: *ActionID*, que é o Identificador da Ação na Tabela, e chave primária , *Description*, que é a descrição da ação e *Status*, que é um inteiro que identifica o status da ação. Os status possíveis são: *0 - Inativo e 1- Ativo.* Essa tabela é uma tabela de domínio usada para armazenar os tipos de ação possível para um registro na tabela de histórico. Ações possíveis: Executar Consulta,

Figura 4.3: *A Modelagem de dados da aplicação*

Adicionar Usuário, Desabilitar Usuário, Adicionar Esquema, Alterar Esquema, Desabilitar Esquema, Adicionar Permissões, Remover Permissões, Adicionar Nó, Alterar Nó, Desabilitar Nó. Sua criação foi feita também para normalizar o modelo de dados, de forma, que, estatísticas sejam mais facilmente extraídas a partir da tabela de histórico.

Schema Nessa tabela ficam armazenadas informações sobre os esquemas. Um esquema foi uma forma adotada para agrupar Tabelas Virtuais. Dessa forma, um esquema pode possuir várias Tabelas Virtuais, e, uma Tabela Virtual pode pertencer a um e somente um esquema. Essa tabela é composta pelos seguintes atributos: *SchemaID*, que é o identificador do esquema e chave primária, *SchemaName*, que é o nome do esquema, *CreationDate*, que é a data de criação do esquema, *DisabledDate*, que é a data em que o esquema foi desabilitado, *XMLFilePath*, que é o caminho do arquivo XML usado para carregar o esquema, armazenado no servidor de aplicação, *Status*, que é o estado possível de um esquema. Os status posíveis são: *0 - Inativo e 1- Ativo*, *UserIDCreation*, que é o usuário que criou o esquema e *UserIDDisabled*, que é o usuário que desabilitou o esquema.

Node Nessa tabela ficam armazenadas informações sobre os nós do Grid (Bancos de Dados). Essa tabela é composta pelos seguintes atributos: *NodeID*, que é o identificador do nó e chave primária, *CreationDate*, que é a Data de Criação do nó, *NickName*, que é um apelido dado para o nó, sendo um valor único, não permitindo duplicidade de valores, usado para facilitar formas de chamar um nó, sendo muito útil no momento de consulta, *Host*, que é o endereço IP ou nome no DNS que o host pode ser encontrado, *UserName*, que é o Usuário que possui permissão no Database, *Password*, que é a senha do usuário, criptografada usando o método MD5, *InstanceName*, que ó nome da instância que o banco de dados se encontra, *DBSourceID*, que é a origem de dados desse nó, sendo uma chave estrangeira para a tabela de DBSource, *Description*, que é um pequeno descritivo sobre o nó, *Status*, que armazena informação sobre o estado do nó. Os status possíveis são: *0 - Inativo e 1- Ativo*, *Database Name*, que é o nome do Banco de dados. Por exemplo, no SQL Server uma instância pode possuir mais de um Banco de Dados. Logo, no nó deve ter

informação sobre qual Banco de dados esse nó se referencia, e *UserIDCreation*, que armazena o identificador do usuário que criou esse nó.

User Nessa tabela ficam armazenadas informações cadastrais sobre usuários na aplicação. Um usuário pode ser ou um usuário administrador, ou um usuário cliente. Os atributos dessa tabela são: *UserID*, que é o identificador da tabela, chave primária, *CreationDate*, que é a data de criação desse usuário, *DisabledDate*, que é a data de desativação desse usuário, *UserIDDisabled*, que é o usuário que desativou esse usuário, *UserName*, que é o nome do usuário, usado como login na aplicação, não permitindo duplicidade, *Password*, que é a senha desse usuário, armazenada usando o método de criptografia MD5, *UserType*, que identifica o tipo do usuário. Um usuário pode ter dois tipos: *0 - Administrador, 1 - Cliente*, *CreatorID*, que identica qual usuário criou o usuário, *Status*, que identifica o Status do Usuário no Sistema. Os status posíveis são: *0 - Inativo e 1- Ativo*, e, *UserIDDisabled*, que identifica qual o usuário que desabilitou o usuário. Um usuário root é criado na aplicação como um primeiro usuário, com o perfil de administrador.

SchemaTable Nessa tabela ficam armazenadas as Tabelas Virtuais. As tabelas virtuais são associadas a um e somente um esquema. Os atributos dessa tabela são: *SchemaTableID*, que identifica a tabela, chave primária, *CreationDate*, que é a data de criação dessa tabela virtual, *DisabledDate*, que é a data de desabilitação dessa tabela virtual, *UserIDCreation*, que é o usuário que criou essa tabela virtual, *UserIDDisabled*, que é o usuário que desabilitou essa tabela, *LogicalTableName*, que é o nome lógico dado à tabela virtual. Esse nome deve ser único em relação a um esquema, e *Status*, que é o estado da tabela virtual. Os status possíveis são: *0 - Inativo e 1- Ativo.*

SchemaTableColumn Nessa tabela ficam armazenadas quais são as colunas da tabela virtual. Os atributos dessa tabela são: *SchemaTableColumnID*, que identifica a tabela, chave primária, *SchemaTableID*, que é o identificador da tabela virtual, *LogicalColumnName*, que é o nome da coluna na tabela virtual, *GlobalDataTypeID*, que é o tipo de dado, sendo chave estrangeira da tabela GlobalDataType, *isPK*, que possui valores 0 e 1, que identifica se o campo é

chave primária ou não, *isNullable*, que identifica se o campo é NOT NULL ou não, e *UserIDCreation*, que identifica que usuário criou o registro na tabela.

SchemaTableNode Essa tabela é uma tabela associativa, que relaciona as tabelas virtuais criadas aos Nodes criados. Seus atributos são: *SchemaTAbleID*, que identifica a tabela junto com o campo *NodeID*, que é chave estrangeira da tabela de Node, *TableName*, que é o nome físico da tabela no node, *SelecT-Clause*, que informa qual será a lista de campos disponível na tabela física para consulta posterior, *WhereClause*, que será qual a seleção dos registros será disponível para consulta posterior, *Status*, que é o estado do registro na tabela. Os status possíveis são: *0 - Inativo e 1- Ativo*, e *UserIDCreation*, que identifica qual o usuário que criou o registro.

SchemaTableNodeColumn Essa tabela é uma tabela associativa, que relaciona as colunas da tabela física às colunas da tabela virtual. Os atributos dessa tabela são: *SchemaTableNodeColumnID*, que identifica a tabela junto com os campos *SchemaTableID*, identificador da tabela virtual e *NodeID*, que identifica o Node. Os demais atributos são: *SchemaTableCOlumnID*, que identifica qual a coluna lógica, *SourceDataType*, que identifica qual o tipo de dado de origem, *Nullable*, que identifica se a coluna permite null ou não e *UserIDCreation*, que identifica que usuário criou esse registro na tabela.

UserView Como mencionado anteriormente, o sistema possui dois tipos de usuários: *Administradores e Clientes*. Um usuário Administrador possui total acesso à aplicação. Já um usuário do tipo Cliente necessita ter permissões às tabelas virtuais criadas. Esse foi o motivo da criação da tabela UserView, para associar Usuários às tabelas virtuais. Os atributos dessa tabela são: *UserID*, que identifica o usuário, *SchemaTableID*, que identifica a tabela virtual que o mesmo tem acesso. Ambos os campos são identificadores da tabela, sendo chave primária.

History Nessa tabela estão todos os registros de todas as ações existentes no sistema. Essa tabela no futuro será muito útil, para por exemplo, definir critérios de escalonamentos para consultas. Os atributos dessa tabela são: *HistoryID*, chave primária, que identifica cada registro, *ActionID*, que identifica

qual ação desse registro, chave estrangeira da tabela de Action, *DateTime*, que é a data/hora que o registro foi inserido, *Description*, descrição da ação que foi feita (por exemplo, qual consulta foi submetida), textitUserID, que identifica qual usuário executou tal ação.

HistoryNode Nesta tabela estão armazenados os históricos das consultas que foram submetidas no sistema. No futuro, essa tabela será muito útil para estatísticas, por conter dados, que podem medir por exemplo, quais nós são mais acessados, quais nós estão com maiores percentuais de falhas, ou quais nodes estão tendo maior tempo de resposta. Estas informações podem ser úteis para auxiliar na distribuição de consultas e de dados, além de auxiliar na escolha da melhor localização de tabela física onde determinado dado deve ser armazenado. Os atributos dessa tabela são: *HistoryID*, que identifica qual o histório referente (chave estrangeira para a tabela de histórico), *NodeID*, chave estrangeira para a tabela Node, que identifica qual o nó, *StartDate*, que armazena a data inicial da operação, *End Date*, que armazena a data final da operação, *BytesTransfered*, que armazena a quantidade de bytes transferidos durante a operação *ElapsedTime*, que armazena o total de tempo gasto durante a operação e *Status*, que identiifca se a operação teve sucesso ou falha. Valores possíveis desse campo: *0 - falha e 1- sucesso.*

Um esquema nesse trabalho é uma forma de organizar as Tabelas Virtuais. Um esquema pode conter uma ou mais Tabelas Virtuais. Mas, uma Tabela Virtual pode pertencer a um e somente um esquema.

Uma Tabela Virtual pode ser composta por uma ou mais tabelas físicas em nós no Grid. As tabelas físicas não precisam ter o mesmo nome ou suas colunas também não precisam ter o mesmo nome das outras tabelas físicas que compõem a tabela virtual. Além disso, as tabelas físicas podem ser somente alguns registros selecionados para comporem a tabela virtual. E as tabelas físicas que compõem a tabela virtual podem ser de diferentes SGBDs. Atualmente a aplicação somente suporta Oracle XE, Sql Server 2005 Express Edition e MySQL.

Todas as ações na aplicação são armazenadas na tabela History. A tabela History possui informação como o tempo gasto no processamento, e, se o processamento foi feito com sucesso ou não. Essa tabela será muito útil no futuro para aspectos de

tomada de decisão como, por exemplo, escolher a melhor localização de tabela física para inserção de dados e escalonamento de consultas.

4.1.2 Casos de uso

Em um ambiente de Computação em Grids, podemos ter várias organizações, que podem estar, cada uma, em um país diferente. E, além disso, essas organizações podem ter alguns fatores em comum, podem entre si, compartilhar recursos. Um desses recursos, por exemplo, poderia ser uma tabela de Clientes de uma base de dados, onde, após uma associação entre as organizações, as mesmas "uniriam" essas bases de dados em uma só.

Poderíamos simplesmente fazer um trabalho de migração e escolher uma das Organizações para ter essa base de dados com todos os Clientes. Mas, quando temos Gygabytes de dados, isso exigiria uma máquina muito robusta, o que implicaria custos. Além disso, as organizações podem ter suas tabelas em bases distintas. Por exemplo, A organização A tem a sua tabela de Clientes em Oracle 10G Express Edition. Já a organização B tem a sua tabela de Clientes em SQL Server 2005 Express Edition. Já a organização C tem a sua tabela de Clientes em MySQL 5.0. Para juntar essas bases seria necessário um serviço de migração, e isso também implicaria em custo, visto que a hora de um Profissional Certificado Oracle é caríssima.

Poderíamos também, utilizar um recurso que o SQL Server possui que é o Linked Server. O Oracle possui um similar, que é o DBLINK. Para utilizar esse recurso, deveríamos então, para deixar invisível para o usuário, criar uma VIEW, que iria fazer o UNION de várias tabelas nos linked servers cadastrados. Mas, esse recurso não seria eficiente para nós, pois, caso algum dos nós da Grid esteja indisponível, a VIEW pára de funcionar, pois, o Linked Server/DB Link só irá retornar os dados se todos os nós estiverem disponíveis. Isso nos causaria uma limitação, pois, nem sempre todos os nós do Grid estarão ativos. E, com isso, estaríamos impossibilitando TODO o Sistema por causa dessa indisponibilidade.

O sistema foi desenvolvido em Java, e a sua interface está toda feita em Web Forms. A escolha por Web Forms foi feita pois outras abordagens, tais como Windows Forms, obrigam o usuário final a ter uma máquina compatível à tecnologia escolhida. O Windows Forms feito em Swing é bem pesado. Além disso, utilizando

Web Forms, precisaríamos apenas de um Servidor Web, que iria armazenar a nossa aplicação. Então, problemas como, ter que garantir que todos os usuários estão com a versão correta, que seria muito complicado de resolver com uma aplicação do tipo Windows Forms, ficaria resolvido, pois só teríamos que atualizar em um local, que seria o Servidor Web.

Implementamos o sistema com base em vários casos de uso. Estes casos de uso concentram-se nas seguintes ações:

- Controlar quais são os nós disponíveis no Grid.
- Permitir ao usuário fazer consultas, que irão fazer pesquisas nos nós que estão disponíveis do Grid.
- Controlar quais serão as tabelas que estarão sendo disponibilizadas no Grid.
- Controlar quais serão as visões de cada usuário em cada base de dados.
- Disponibilizar estatísticas sobre as consultas, como tempo de resposta.

Nas próximas subseções, descrevemos cada caso de uso no sistema.

4.1.2.1 Incluir Nó

Atores: Administrador do Sistema.

Finalidade: Incluir um novo nó do Grid, para que o mesmo participe dos esquemas.

Visão geral: O Administrador do Sistema irá entrar informações como a localização do nó, usuário e senha para ter acesso ao nó.

4.1.2.2 Alterar Nó

Atores: Administrador do Sistema.

Finalidade: Alterar um nó já existente do Grid.

Visão geral: O Administrador do Sistema irá alterar informações como a localização do nó, usuário e senha para ter acesso ao nó.

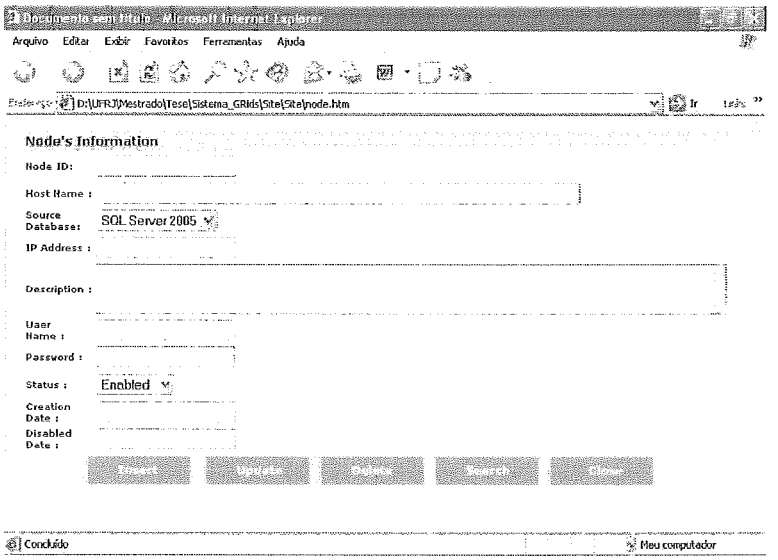


Figura 4.4: *Caso de uso: Nó*

Tabela 4.1: Caso de uso: Incluir Nó

Ação do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja incluir um novo nó do Grid.	
2. O administrador então entra com as informações sobre o Grid.	
3. O Administrador clica no botão de Insert, solicitando a inclusão do nó.	
4.	O Sistema valida se as informações digitadas estão válidas.
5.	O sistema valida se o nó incluído já existe.
6.	Caso não existam problemas, o sistema inclui o nó e envia uma mensagem para o usuário informando que o registro foi incluído com sucesso.

4.1.2.3 Excluir Nó

Atores: Administrador do Sistema.

Finalidade: Excluir um nó já existente do Grid. **Visão geral**: O Administrador do Sistema irá excluir um nó existente.

4.1.2.4 Criar Esquema

Atores: Administrador do Sistema.

Tabela 4.2: Caso de uso: Alterar Nó

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja alterar um Nó do Grid.	
2. O administrador localiza o Nó que ele deseja alterar.	
3. O administrador então entra com as informações sobre o Grid que ele deseja alterar.	
4.	O Sistema valida se as informações digitadas estão válidas.
5. O Administrador clica no botão de Update, solicitando a alteração do Nó.	
6.	O Sistema valida se as informações digitadas estão válidas.
7.	Caso não existam problemas, o Sistema altera o Nó e envia uma mensagem para o Usuário informando que o registro foi alterado com sucesso.

Finalidade: Incluir um novo esquema do Grid, para que o mesmo participe das consultas que os usuários possam executar.

Visão geral: O Administrador do Sistema irá entrar informações como o nome do esquema, e o seu esquema XML com sua definição de tabelas e campos.

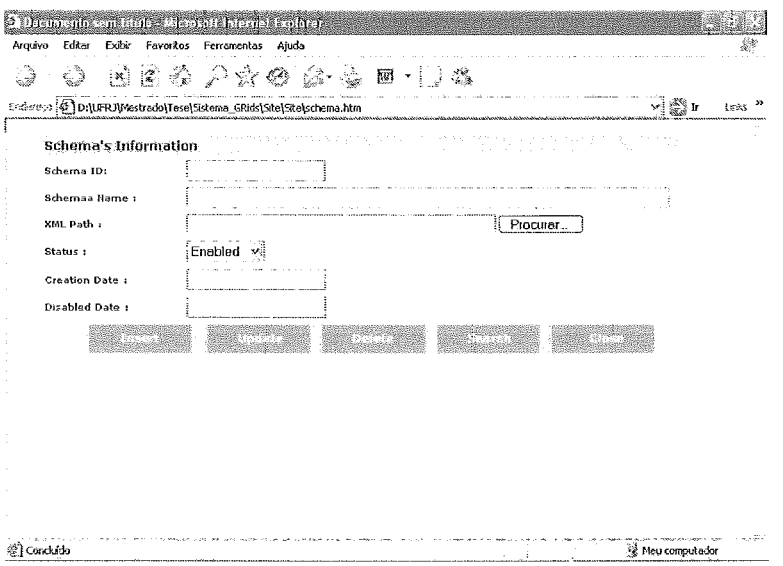


Figura 4.5: *Caso de uso: Esquema*

Tabela 4.3: Caso de uso: Excluir Nó

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja excluir um Nó do Grid.	
2. O administrador localiza o Nó que ele deseja alterar.	
3. O administrador clica no botao de Delete, solicitando a exclusão do Nó.	
4.	O Sistema solicita uma confirmação ao usuário se ele realmente deseja excluir o nó.
5.	Caso esse nó tenha algum relacionamento com as demais tabelas do sistema, o sistema emite uma mensagem indicando que esse registro possui dependências, e então, aconselha o usuário a desabilitar o Nó ao invés de excluir.
6.	Caso não existam dependências, o Sistema exclui o Nó, e envia uma mensagem para o usuário informando que o Nó foi excluído com sucesso.

4.1.2.5 Alterar Esquema

Atores: Administrador do Sistema.

Finalidade: Alterar um esquema do Grid.

Visão geral: O Administrador do Sistema irá entrar informações como o nome do esquema, e o seu esquema XML com sua definição de tabelas e campos.

4.1.2.6 Excluir Esquema

Atores: Administrador do Sistema.

Finalidade: Excluir um esquema do Grid.

Visão geral: O Administrador do Sistema irá excluir um esquema já existente do Grid.

4.1.2.7 Incluir Usuário

Atores: Administrador do Sistema.

Finalidade: Incluir um usuário do Grid.

Tabela 4.4: Caso de uso: Incluir Esquema

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja incluir um novo esquema no Grid.	
2. O administrador então entra com as informações sobre o Esquema.	
3. O Administrador clica no botão de Insert, solicitando a inclusão do Esquema.	
4.	O Sistema valida se as informações digitadas estão válidas.
5.	O Sistema valida se o Esquema incluído já existe.
6.	O Sistema valida o XML, e monta um LOG com possíveis inconsistencias, e, caso existam alguma, informa ao usuários todas as inconsistencias.
7.	Caso não existam inconsistências, o Sistema Inclui o Esquema, e informa ao usuário que o Esquema foi incluído com sucesso. As informações do XML são inseridas em tabelas do banco.

Visão geral: O Administrador do Sistema irá entrar informações como o nome do usuário, senha, tipo de usuário.

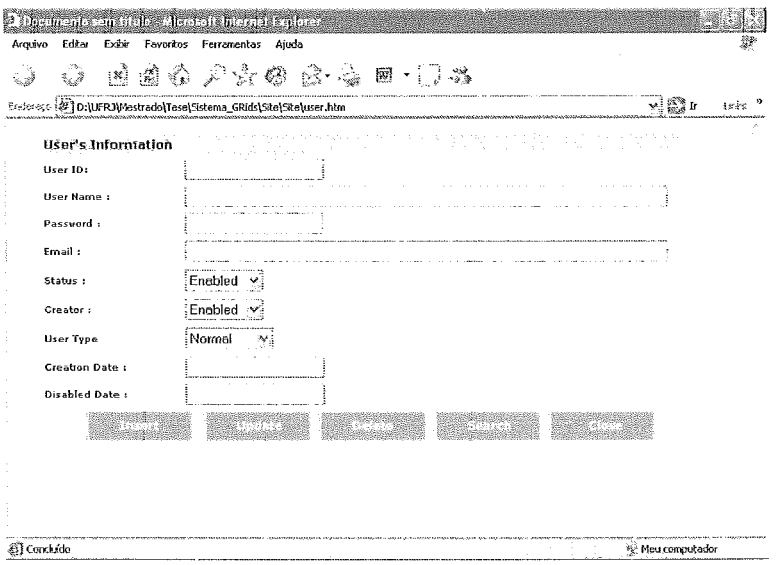


Figura 4.6: *Caso de uso: User*

Tabela 4.5: Caso de uso: Alterar Esquema

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja alterar um esquema no Grid.	
2. O administrador localiza o Esquema que ele deseja alterar.	
3. O administrador então entra com as informações sobre o Esquema.	
4.	O Administrador clica no botão de Update, solicitando a alteração do Esquema.
5.	O Sistema valida se as informações digitadas estão válidas.
6.	O Sistema valida se o Esquema incluído já existe.
7.	O Sistema valida o XML, e monta um LOG com possíveis inconsistencias, e, caso existam alguma, informa ao usuários todas as inconsistencias.
8.	Caso não existam inconsistências, o Sistema Inclui o Esquema, e informa ao usuário que o Esquema foi alterado com sucesso. As informações do XML são inseridas em tabelas do banco.

4.1.2.8 Alterar Usuário

Atores: Administrador do Sistema.

Finalidade: Alterar um usuário do Grid.

Visão geral: O Administrador do Sistema irá alterar informações como o nome do usuário, senha, tipo de usuário.

4.1.2.9 Excluir Usuário

Atores: Administrador do Sistema.

Finalidade: Excluir um usuário do Grid.

Visão geral: O Administrador do Sistema irá excluir um usuário já existente.

4.1.2.10 Associar Visões de Usuários a Esquemas

Atores: Administrador do Sistema.

Tabela 4.6: Caso de uso: Excluir Esquema

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja excluir um esquema no Grid.	
2. O administrador localiza o Esquema que ele deseja excluir.	
3. O Administrador clica no botão de Delete, solicitando a exclusão do Esquema.	
4.	O Sistema valida se o esquema selecionado pode ser excluído, se existe alguma dependência.
5.	Caso exista alguma dependência, o sistema envia uma mensagem informando que o esquema não pode ser excluído devido as dependências, e sugere que o mesmo seja desabilitado ao invés de excluído.
6.	Caso não existam dependências, o esquema é excluído e o sistema envia uma mensagem para o usuário informando que o esquema foi excluído com sucesso.

Tabela 4.7: Caso de uso: Incluir Esquema

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja incluir um usuário no Grid.	
2. O administrador então entra com as informações sobre o Usuário.	
3. O Administrador clica no botão de Insert, solicitando a inclusão do Usuário.	
4.	O Sistema valida se as informações digitadas estão válidas.
5.	O Sistema valida se o Usuário incluído já existe.
6.	Caso não existam inconsistências, o Sistema Inclui o Usuário, e informa ao usuário que o Usuário foi incluído com sucesso.

Finalidade: Associar aos usuários não administradores existentes, visões aos esquemas existentes.

Tabela 4.8: Caso de uso: Alterar Usuário

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja alterar um usuário no Grid.	
2. O administrador localiza o usuário que ele deseja alterar.	
3. O administrador então entra com as informações sobre o Usuário.	
4. O Administrador clica no botão de Update, solicitando a alteração do Usuário.	
5.	O Sistema valida se as informações digitadas estão válidas.
6.	Caso não existam inconsistências, o Sistema altera o Usuário, e informa ao usuário que o Usuário foi alterado com sucesso.

Tabela 4.9: Caso de uso: Excluir Usuário

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja excluir um usuário no Grid.	
2. O administrador localiza o usuário que ele deseja excluir.	
3. O Administrador clica no botão de Delete, solicitando a exclusão do Usuário.	
4.	O Sistema valida se o usuário pode ser excluído, se existe alguma dependência dele no sistema. Caso positivo, o sistema envia uma mensagem para o usuário informando que o usuário não pode ser excluído. Somente um usuário com perfil de SuperUser pode excluir um usuário do tipo Administrador.
5.	Caso não existam inconsistências, o Sistema excluído o Usuário, e informa ao usuário que o Usuário foi excluído com sucesso.

Visão geral: O Administrador do Sistema irá associar esquemas à usuários não administradores.

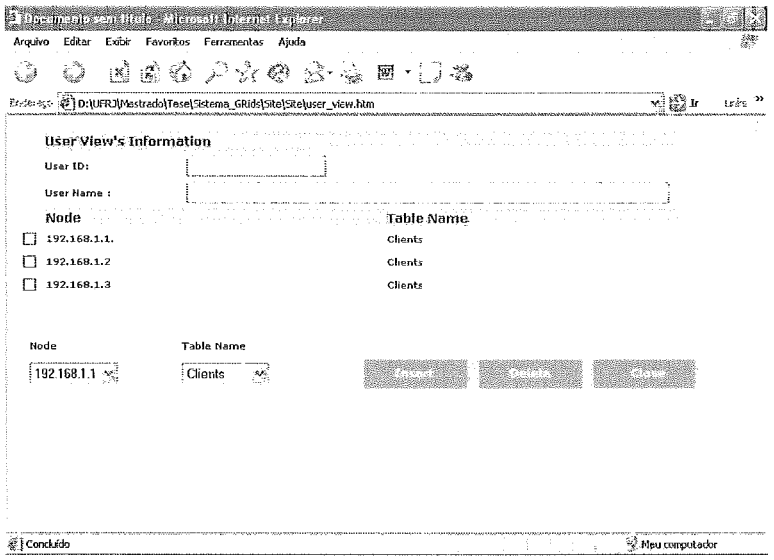


Figura 4.7: *Caso de uso: Associar Visoes de Usuarios a Esquemas*

Tabela 4.10: Caso de uso: Associar Visões de Usuários à Esquemas

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja associar um esquema do Grid a um determinado usuário (dando permissão ao usuário a poder consultar esse esquema). Esse usuário não possui um perfil de administrador/superuser do Sistema.	
2. O administrador localiza o usuário que ele deseja associar o esquema.	
3. O administrador então seleciona o nó e a tabela que ele deseja associar.	
4. O Administrador clica no botão de Insert.	
5.	O Sistema valida se as informações digitadas estão válidas.
6.	Caso não existam inconsistências, o Sistema associa a tabela/nó, e informa ao usuário que o a tabela/nó foi associada com sucesso sucesso.

4.1.2.11 Desassociar Visões de Usuários a Esquemas

Atores: Administrador do Sistema.

Finalidade: Desassociar aos usuários não administradores existentes, visões aos

esquemas existentes.

Visão geral: O Administrador do Sistema irá desassociar esquemas à usuários não administradores.

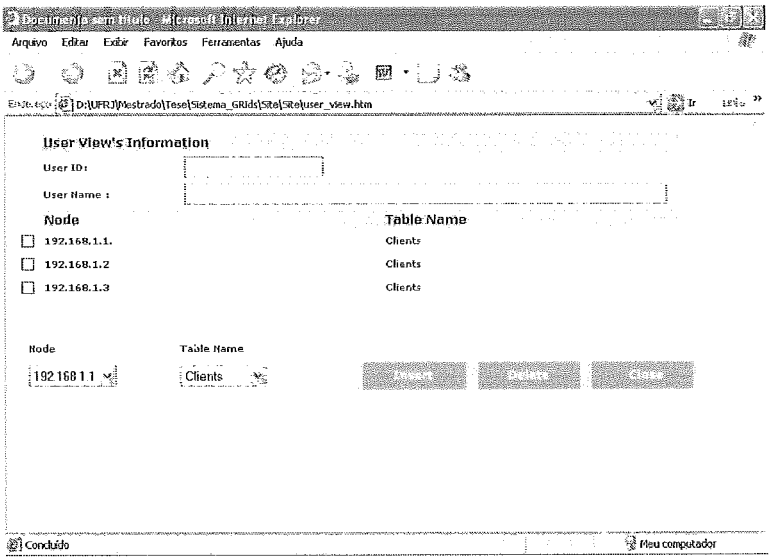


Figura 4.8: *Caso de uso: Desassociar Visoes de Usuarios a Esquemas*

Tabela 4.11: Caso de uso: Desassociar Visões de Usuários à Esquemas

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja desassociar um esquema do Grid a um determinado usuário (dando permissão ao usuário a poder consultar esse esquema). Esse usuário não possui um perfil de administrador/superuser do Sistema.	
2. O administrador localiza o usuário que ele deseja associar o esquema.	
3. O administrador então seleciona o nó e a tabela que ele deseja desassociar.	
4. O Administrador clica no botão de Delete.	
5.	O Sistema valida se as informações digitadas estão válidas.
6.	Caso não existam inconsistências, o Sistema desassocia a tabela/nó, e informa ao usuário que o a tabela/nó foi desassociada com sucesso sucesso.

4.1.2.12 Executar Consultas

Atores: Usuário do Sistema / Administrador.

Finalidade: Executar consultas nos esquemas existentes no Grid que o usuário logado tenha permissão.

Visão geral: O Administrador do Sistema / Cliente irá executar consultas.

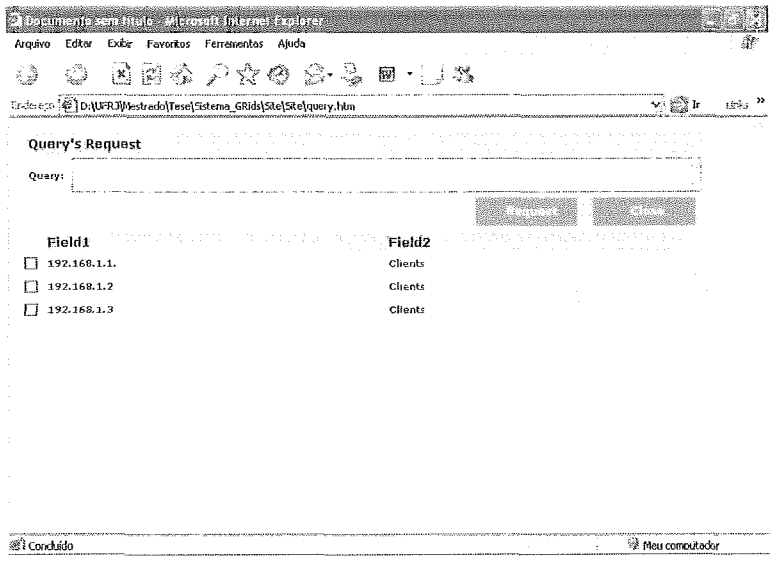


Figura 4.9: *Caso de uso: Executar Consultas*

4.1.2.13 Consultar Histórico

Atores: Administrador.

Finalidade: Visualizar as ações já executadas no Grid.

Visão geral: O Administrador do Sistema irá visualizar as informações já executadas no Grid.

4.1.2.14 Efetuar Login

Atores: Usuário do Sistema./ Administrador.

Finalidade: Autenticar o Usuário no Sistema.

Visão geral: O usuário irá se logar no sistema, e dependendo do seu perfil, terá permissões a determindas funcionalidades.

Tabela 4.12: Caso de uso: Executar Consultas

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja executar uma determinada consulta de select no Grid.	
2. O administrador digita a consulta no padrao ANSI SQL.	
3. O Administrador clica no botão de Request.	
4.	O Sistema valida se as a sintaxe da consulta está OK.
5.	O Sistema valida se as tabelas solicitadas estão na visão do usuário logado.
6.	Caso não seja validada, o Sistema emite uma mensagem para o usuário informando o erro.
7.	Caso seja validada, o Sistema então percorrerá todos os nós dos Grids que estão na visão do usuário, nas tabelas solicitadas na consulta, e retorna os dados, fazendo um UNION de todas as informações encontradas nos nós dos grids, e mostra para os usuário em forma de XML.

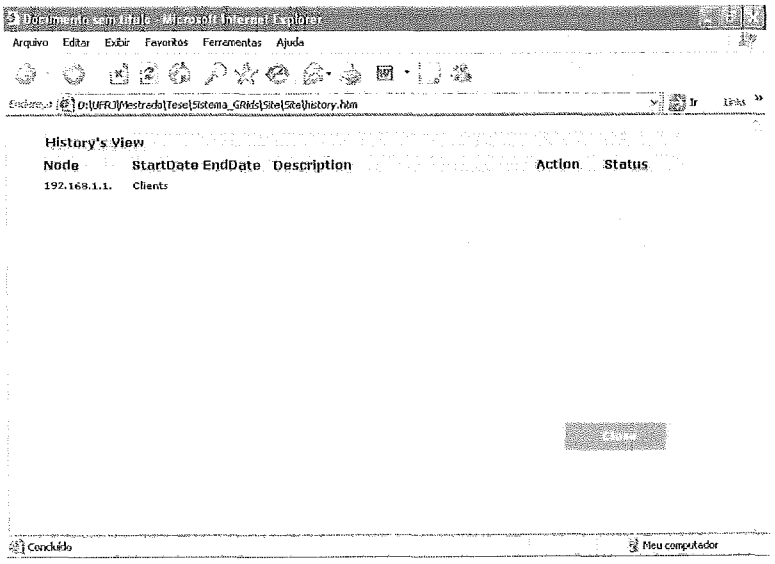


Figura 4.10: *Caso de uso: Consultar Histórico*

Tabela 4.13: Caso de uso: Consultar Histórico

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o administrador do Sistema deseja consultar o histórico do Grid.	
2.	O Sistema mostra todo o histórico de todas as ações realizadas no Grid até o momento.

Figura 4.11: *Caso de uso: Autenticar o Usuário no Sistema.*

Tabela 4.14: Caso de uso: Autenticar o Usuário no Sistema

Acao do ator	Resposta do Sistema
1. Este caso de uso inicia quando o usuário deseja entrar no sistema.	
2. O usuário entra com seu login/senha.	
3. O Usuário clica no botão de Enter.	
4.	O Sistema valida se os campos foram preenchidos.
5.	O Sistema valida se usuário/senha foram validados.
6.	Caso não seja validada, o Sistema emite uma mensagem para o usuário informando que usuário/senha inválidos.
7.	Caso seja validado, o sistema então verifica qual é o perfil do usuário, e dependendo do perfil, ele mostra uma tela diferente.

4.2 Ferramentas de desenvolvimento

Nessa seção descrevemos as tecnologias usadas para desenvolvimento da aplicação e do CORE do Processamento. Como mencionado anteriormente, foi utilizado Java, e, banco de dados MySQL.

4.2.1 Aplicação

Nessa seção descrevemos duas tecnologias usadas na Aplicação: Java e XML. Java foi escolhida como linguagem de desenvolvimento devido a sua aceitação no mundo do Open Source, e também, por possuir características de portabilidade, características essas muito interessantes, visto que podemos em um futuro próximo migrar a plataforma do mesmo, sem, teoricamente, nenhuma alteração de código.

4.2.1.1 Java

Java foi uma idéia de James Gosling, Patrick Naughton, Chris Warth, Ed Frank e Mike Sheridan na Sun Microsystems, Inc. em 1991. Demorou cerca de 18 meses para desenvolver a primeira versão. Inicialmente a linguagem foi chamada de "Oak", mas foi nomeada para "Java" em 1995. Entre a implementação inicial de Oak de 1992 e o anúncio público de Java na primavera de 1995, muitas pessoas contruiram para o design e evolução da linguagem. Bill Joy, Arthur van Hoff, Jonathan Payne, Frank Yellin e Tim Lindholm foram contribuidores chaves para a maturidade do protótipo inicial [27].

De alguma forma, o impulso original para Java não foi a Internet. Ao contrário, a motivação principal foi a necessidade de uma linguagem de plataforma independente que poderia ser usada para criar software a ser embutido em vários dispositivos eletrônicos, como micro-ondas e controles remotos. O problema com C e C++ (e maioria das outras linguagens) é que eles são feitos para ser compilados para um arquitetura específica. Apesar que é possível compilar um programa C++, para algum tipo de CPU, mas para fazer isso é requerido um compilador total C++ direcionado para uma CPU específica. O problema é que compiladores são caros e demoram tempo para serem criados. Uma solução simples e com melhor custo-benefício era necessária [27]. Gosling e os outros iniciaram o trabalho em uma linguagem portátil, independente de plataforma que poderia ser usada para

produzir código que poderia rodar em uma variedade de CPUs abaixo de ambientes diferentes. Esse esforço liderou a criação de Java. Java possui vantagens sobre seus competidores, citadas pelos criadores de Java:

- simples, orientada a objeto e familiar. Java é simples e possui muitas semelhanças com C++;
- robusta;
- distribuída e segura - uma linguagem e sistema run-time designada para operar em ambientes distribuídos precisa embutir características de segurança;
- interpretada, arquitetura neutra e portável - o uso de uma máquina virtual que interpreta uma arquitetura neutra garante que programas são portáveis de uma plataforma para outra;
- alta performance;
- dinâmica - o link é dinâmico; classes somente são carregadas quando necessárias.
- multithread - suporte da linguagem para programação concorrente permite aplicações portáveis e multithreads serem construídas.

Em 1997 a Sun Microsystems tentou submeter a linguagem a padronização pelos orgãos ISO/IEC e ECMA, mas acabou desistindo. [1][2][3] Java ainda é um standard de fato, que é controlada através da JCP Java Community Process.[4] Em 13 de Novembro de 2006, a Sun lançou a maior parte do Java como Software Livre sob os termos da GNU General Public License (GPL). Em 8 de Maio de 2007 a Sun finalizou o processo, tornando praticamente todo o código Java como software de código aberto, menos uma pequena porção que a Sun não possui copyright.

Java hoje já possui um desempenho próximo do C++. Isto é possível graças a otimizações como a compilação especulativa, que aproveita o tempo ocioso do processador para pré-compilar bytecode para código nativo. Outros mecanismos ainda mais elaborados como o HotSpot da Sun, que guarda informações disponíveis somente em tempo de execução (ex.: número de usuários, processamento usado, memória disponível), para otimizar o funcionamento da JVM, possibilitando que a

JVM vá "aprendendo" e melhorando seu desempenho. Isto é uma realidade tão presente que hoje é fácil encontrar programas corporativos e de missão crítica usando tecnologia Java. No Brasil, por exemplo, a maioria dos Bancos utiliza a tecnologia Java para construir seus home banks, que são acessados por milhares de usuários diariamente. Grandes sítios como o eBay utilizam Java para garantir alto desempenho. E a cada ano Java tem se tornado mais rápido, na medida que se evolui o compilador dinâmico.

Essa implementação no entanto tem alguns problemas. A pré-compilação exige tempo, o que faz com que programas Java demorem um tempo significativamente mais para começarem a funcionar. Soma-se a isso o tempo de carregamento da máquina virtual. Isso não é um grande problema para programas que rodam em servidores e que deveriam ser inicializados apenas uma vez. No entanto, isso pode ser bastante indesejável para computadores pessoais, onde o usuário deseja que o programa rode logo depois de abri-lo. A próxima versão da máquina virtual produzida pela Sun promete novos recursos que irão minimizar este fato.

O Java ainda possui uma outra desvantagem considerável em programas que usam bastante processamento numérico. O padrão Java tem uma especificação rígida de como devem funcionar os tipos numéricos. Essa especificação não condiz com a implementação de pontos flutuantes na maioria dos processadores o que faz com que o Java seja significativamente mais lento para estas aplicações quando comparado a outras linguagens.

Os bytecodes produzidos pelos compiladores Java podem ser usados num processo de engenharia reversa para a recuperação do programa-fonte original. Esta é uma característica que atinge em menor grau todas as linguagens compiladas. No entanto, já existem hoje tecnologias que "embaralham" e até mesmo criptografam os bytecodes praticamente impedindo a engenharia reversa.

Atualmente, Java compete muito com o .NET, da Microsoft. Muitas empresas têm optado pelo Java, por causa do Open Source, mas, com desvantagens do custo do profissional java, que é elevado, quando comparado ao custo de um profissional .NET. Já, outras empresas optam pelo .NET, da Microsoft, principalmente pelo custo baixo do profissional, quando comparado com o profissional Java.

4.2.1.2 XML

A XML foi inventada a partir da SGML (Standard Generalized Markup Language), linguagens de marcação e padrão generalizada usada por muitos anos. A própria HTML é um aplicativo SGML, e a XML é um subconjunto da SGML ideal para ser usada na Internet; então podemos dizer que todo documento XML é um documento SGML e nem todo o documento SGML é um documento XML. A XML simplifica a SGML já que remove todas as opções que não são absolutamente necessárias na SGML.

Woldwide Web Consortium (W3C) definiu diversos padrões que complementam a XML, assim como outros fornecedores também podem criar ferramentas para a XML. Todas essas ferramentas e padrões são úteis para criar, manipular e armazenar documentos XML. Três desses padrões são o XLS, DOM e Namespace:

Folhas de estilo É fundamental para um sítio ter boa aparência. Com XML não é problema, pois pode-se utilizar folhas de estilo XLS e CSS para exibir o conteúdo de um XML. Podemos, processar a XML no servidor com ASP ou CGI, processar a XML no cliente usando as linguagens de script e exibi-la diretamente com as folhas de estilo XLS ou CSS.

DOM Sua sigla vem de Document Object Model, são APIs de acesso aos documentos XML. Elas permitem percorrer e extrair informações específicas do documento, além, de poder adicionar, alterar ou excluir dados.

Namespace É um sistema de nomeação global (URI) para os nomes de elementos e atributos.

Muitos aplicativos no futuro utilizarão em algum momento a XML, pois um grande número de empresas já está desenvolvendo, ou já desenvolveu algum produto com suporte a XML. Neste cenário podemos citar o suporte a XML incluído no SQL Server 2000, onde é possível agorar criar documentos XML de maneira rápida e fácil. Podemos também citar um produto chamado Tamino que cria bancos de dados XML [20]. Atualmente, o SQL Server 2005 já possui um tipo de dados XML, o que demonstra que [20] estava certo na sua premonição.

A escolha de XML como output desse projeto se deu pela popularidade de XML. Podíamos simplesmente exportar os resultados em um arquivo texto, ou até

mesmo, mostrar de forma organizada em tabelas, via HTML. Mas, visto que muitas ferramentas atuais conseguem usar arquivos de XML como bases de dados, como é o caso de .NET, que possui o *DataSet* que pode ter como origem de dados um documento XML.

4.3 Suporte a Banco de Dados

Nosso sistema atualmente suporta três bases de dados, como citados em capítulos anteriores. Vamos descrever um pouco a origem e características de cada um dos bancos de dados que a ferramenta suporta.

4.3.1 MySQL

O MySQL, como citado no capitulo anterior, é uma base de dados open-source, multi-plataforma, muito respeitada tanto mo mundo científico, quanto na indústria.

Características do MySQL, citadas em [3]:

- portátil;
- escrito em C e C++;
- testado em uma ampla faixa de compiladores diferentes;
- funciona em diversas plataformas;
- usa o GNU Automake, Autoconf e LibTool para portabilidade;
- APIs para C, C++, Eiffel, Java, Perl, Php, Python, Ruby e Tcl estão disponíveis.
- suporte total a multi-threads usando threads diretamente no kernel. Significa que se pode facilmente usar múltiplas CPUs, se disponível;
- tabelas em disco baseadas em árvores B, com compressão de índices;
- sistema de alocação de memória rápido e baseado em processo;
- tabelas hash em memória que são usadas como tabelas temporárias;

- seu código foi testado com o Purify (um detector comercial de falhas de memória) e também com o Valgrind, uma ferramenta GPL;
- disponível como versão cliente/servidor.

4.3.2 Oracle XE

O Oracle Database 10G Express Edition (Oracle XE) é um banco de dados com código baseado no Oracle Database 10g Release 2 que é gratuito para desenvolver, publicar e distribuir, simples para administrar. É um ótimo banco de dados inicial para:

- desenvolvedores trabalhando em PHP, Java, .NET, XML e aplicações open source;
- administradores de banco de dados que precisam de um banco de dados inicial, gratuito para treinamento e desenvolvimento;
- vendedores independentes de software e vendedores de hardware que querem um banco de dados inicial para distribuir gratuitamente;
- instituições educacionais e estudantes que precisam de um banco de dados gratuito em seu currículo.

Com o Oracle XE, é possível desenvolver e publicar aplicações com infra-estrutura potente, e, caso necessário, fazer um upgrade, quando necessário, sem o custo de migrações complexas.

O Oracle XE pode ser instalado com as limitações de apenas um banco de dados por máquina, armazenamento de até 4 GB de dados de usuários, e até 1 GB de memória, e uso de apenas uma CPU na máquina host.

Quando este trabalho foi iniciado, estava sendo lançado o Oracle XE. Decidimos, então, colocá-lo como um dos bancos de dados com suporte no projeto. O Oracle é o banco de dados mais respeitado em todo o mercado das grandes empresas. A instalação do Oracle XE é muito simples e a administração é similar à administração do Oracle. Não permite bancos de dados maiores de 4 GB. Mas, como um dos propósitos deste trabalho é integrar bases de dados no Grid, caso as tabelas atinjam os 4 GB, poderíamos particionar a tabela em uma outra máquina do Grid, que poderia atingir até 4 GB e assim por diante.

4.3.3 Microsoft SQL Server 2005 Express Edition

O SQL Server 2005 Express Edition foi considerada a próxima versão de MSDE e é gratuita, facil de usar, leve do SQL Server 2005.

Disponível para download, gratuita para redistribuir, fácil para novos desenvolvedores para uso imediato. Fácil gerenciamento do banco de dados usando o Management Studio Express.

Possui praticamente todos os recursos do SQL Server 2005, como Stored Procedures, Reporting Services, como outros.

Mas, como versão gratuita, possui suas limitações (bem parecidas com as limitações do Oracle XE da Oracle):

- suporte somente para 1 CPU;
- só pode ter 1 GB endereçado de memória RAM;
- o tamanho máximo do banco de dados é de 4 GB;

Ele é compatível com as versões do SQL Server, ou seja, pode, ser facilmente migrado para um SQL Server 2005 Standard ou Enterprise Edition.

A escolha de suporte ao Microsoft SQL Server 2005 foi o mesmo motivo da escolha do Oracle XE: um banco de dados gratuito que surgiu de um banco de dados proprietário. A instalação e administração são simples. Existem limitações na ferramenta de administração (Management Studio Express), quando comparado com o Management Studio do SQL Server 2005. Mas, nada que dificulte o trabalho de um administrador, que, no mínimo deve saber comandos SQL e não ficar preso à interfaces.

4.4 PaP Datagrid Core

Como citado anteriormente, o sistema consiste em duas áreas distintas: Administrativa e Usuário.

A Área Administrativa é a área onde os usuários com um perfil de administrador podem configurar todo o Grid (adicionar nos, tabelas virtuais, esquemas e, além disso, adicionar novos usuários e configurar suas permissões às tabelas virtuais).

A Área do Usuário é a área onde os usuários com um perfil de Usuario podem fazer Consultas às Tabelas Virtuais.

4.4.1 Area de administração

Essa área é composta pelos seguintes casos de usos, já descritos nesse capítulo:

- Incluir Nó
- Alterar Nó
- Excluir Nó
- Criar Esquema
- Alterar Esquema
- Excluir Esquema
- Incluir Usuario
- Alterar Usuario
- Excluir Usuario
- Associar Visoes de Usuarios a Esquemas
- Desassociar Visoes de Usuarios a Esquemas
- Consultar Historico
- Efetuar Login
- Executar Consultas

Além de poder configurar todo o ambiente, o administrador também possui permissões para consultar históricos, que são gravados em tabelas do Sistema de Metadado. Informações, como, quais consultas foram submetidas, e, que nós foram envolvidos nas consultas, e, quanto tempo foi gasto em cada nó serão informações bem importantes para um futuro próximo, quando, pudermos usar esse histórico como base de conhecimento. Uma vez que o sistema tiver replicação funcionando, quando um usuário submeter uma consulta, dados de histórico como qual o nó mais rápido, podem ser usados para escolha.

O foco do sistema nessa fase não foi dado à parte administrativa. Todo o core da parte administrativa foi criado (Tabelas no MySQL, interfaces, mas nao foi dado um foco no funcionamento dessa parte, uma vez que, fazendo manualmente o cadastro na base de dados, permitiria o uso do sistema).

O foco do projeto deu-se à parte de usuário, mais especificamente, ao Processamento de Consultas, que será discutido na próxima seção.

4.4.2 Área de usuários

Essa área é composta pelos seguintes casos de usos, já descritos nesse capítulo:

- Efetuar Login
- Executar Consultas

Basicamente, o usuário irá escrever consultas usando um padrão SQL ANSI, e, a consulta irá ser processada pelo sistema, o qual irá traduzir essa consulta, processando-a nos respectivos nós do Grid, os quais compõem as tabelas virtuais contidas na consulta.

O Sistema então, retorna ao usuário um XML com os resultados da consulta retornados pelo Grid.

4.4.3 Processamento de Consultas

Quando um usuário submete uma consulta em padrão ANSI, esta será executada em cima das Tabelas Virtuais. O Sistema precisa traduzir essa consulta para ser processada localmente nos bancos de dados físicos que compõem as tabelas virtuais.

Como uma Tabela Virtual pode ser composta por uma ou mais colunas, e, Tabelas Virtuais podem ser compostas também por uma ou mais Tabelas Físicas, que podem estar em diferentes Sistemas Gerenciadores de Banco de Dados, o sistema precisa fazer uma Conversão da Consulta que foi feita nas Tabelas Virtuais para as Tabelas Fisicas, dependendo do nó que está compondo essa Tabela Virtual.

O usuário submete uma consulta ao sistema usando a Tabela Virtual. Nesse caso, a Tabela Virtual é composta por 3 Tabelas Físicas, localizadas em três diferentes nós: Nó 1 possui o SGBD MySQL. Nó 2 possui o SGBD SQL Server 2005, e, o Nó 3 possui o SGBD Oracle XE. O Sistema precisa traduzir essa consulta para cada nó,

e, após isso, armazenar na tabela de histórico o tempo gasto, e, se o processamento foi feito com sucesso ou não, e, então, fazer um *UNION ALL* de todos os resultados em um XML com os resultados da consulta. Lembrando que o usuário não possui nenhum conhecimento sobre Nós ou Tabelas Físcias. Essa informação é transparente para ele.

Nesse momento, quando o sistema recebe uma consulta na Tabela Virtual, o sistema faz uma busca sobre quais nós compõem essa Tabela Virtual. Para cada nó que compõe essa Tabela Virtual, o sistema traduz a consulta e então executa a consulta em cada nó, e armazena os resultados em cache. Esse processamento não é feito de uma forma paralela ainda. Mas, como um trabalho futuro, seria fazer esse processamento nos nós em paralelo.

A Conversão da Consulta é feita da seguinte forma:

- São separados os campos da Cláusula SELECT;
- São separados os campos da Cláusula WHERE;
- São separadas as palavras reservadas;
- É feito o DE X Para da tabela encontrada no FROM;
- É feito o DE X Para dos campos encontrados no SELECT e no WHERE;

Após feito esse processo, a consulta é processada no Nó, e, após processado em todos os nós, é feita união dos resultados, e, então, enviado o *resultset*.

Capítulo 5

Experimentos

O *PaP Meta-data Database System* foi implementado e testado em um ambiente heterogêneo consistindo de três SGBDs diferentes, dispersos geograficamente:

- PapZ, localizado na rede local. Servidor contendo o SGBD SQL Server 2005.
- PapNote, localizado na rede local. Servidor contendo o SGBD Oracle XE.
- SSServer, localizado em Benfica-RJ. Servidor contendo o MySQL Versao 5.0. O Firewall desse servidor foi configurado para permitir o IP da máquina onde estava instalada a aplicação na porta do MySQL (3306).

O objetivo do teste foi avaliar a viabilidade da Integração de SGBD heterogêneos em ambientes de Grid utilizando tabelas virtuais e a escalabilidade. Para isto fizemos três simulações, onde uma tabela física foi distribuída pelos três sítios. A distribuição foi feita da seguinte forma: 40 % das linhas foram armazenados no Oracle XE, 40 % no SQL Server 2005 e 20 % no MySQL. Essa distribuição foi feita desta forma devido à robustez e maior rapidez do Oracle e do SQL Server em relação ao MySQL. Nestas três simulações, as tabelas variaram de tamanho. Três tipos de consultas foram realizadas:

- Consulta1: SELECT Codigo, Nome, Telefone from Telefone where Codigo=5 Esta consulta retorna apenas um registro.
- Consulta2: SELECT Codigo, Nome, Telefone from Telefone where Codigo>5 Esta consulta retorna praticamente todos os registros da tabela.

- Consulta3: SELECT Codigo, Nome, Telefone from Telefone where Codigo>5 and Codigo<50 Esta consulta retorna um subconjunto pequeno de registros.

As tabelas físicas foram criadas da seguinte forma:

- No *Oracle XE*, localizado no Servidor PapNote, foi criada a tabela Client, com os campos {ClientID int, Phone varchar2, Name varchar2}.
- No *SQL Server 2005 Express*, localizado no Servidor Papz foi criada a tabela Cliente, com os campos {ClienteID int, Telefone varchar(12), Nome varchar(50)}.
- No *MySQL*, localizado no Servidor SSERVER foi criada a tabela Client, com os campos {ID int, PhoneNumber varchar(12), ClientName varchar(50)}.

Na aplicação, foi realizado o seguinte procedimento para as simulações:

1. Criado o Nó John SQLServer2005 na tabela Node, com o Host: 192.168.0.127, do tipo SQL Server 2005.
2. Criado o Nó SS Mysql na tabela Node, com o Host: ssserver.sytes.net, do tipo MySQL.
3. Criado o Nó John Oracle XE na tabela Node, com o Host: 192.168.0.110, do tipo Oracle XE, na instancia GRIDS.
4. Criado o esquema Grid, na tabela Schema.
5. Criado a tabela virtual Telefone, na tabela SchemaTable.
6. Criado as colunas da tabela virtual Telefone (Codigo, Nome e Telefone) na tabela SchemaTableColumn.
7. Associado a tabela Virtual Telefone ao Nó John SQlServer2005, com a Select Clause: CLientID, Phone, Name, e, a where Clause vazia, na tabela SchemaTableNode.
8. Associado a tabela Virtual Telefone ao Nó SSMySQL, com a Select Clause: ID, PhoneNumber, CLientName, e, a where Clause vazia na tabela SchemaTableNode.

9. Associado a tabela Virtual Telefone ao Nó John Oracle XE, com a Select Clause: CLienteID, Telefone,Nome, e, a where Clause vazia, na tabela SchemaTableNode.

10. Associada as colunas da Tabela Virtual Telefone às colunas da Tabela Fisica do Nó John SqlServer2005, na tabela SchemaTableNodeColumn.

11. Associada as colunas da Tabela Virtual Telefone às colunas da Tabela Fisica do Nó SS MySQL, na tabela SchemaTableNodeColumn.

12. Associada as colunas da Tabela Virtual Telefone às colunas da Tabela Fisica do Nó John Oracle XE, na tabela SchemaTableNodeColumn.

A seguir apresentamos os resultados das três simulações.

5.1 Simulação 1

Nessa simulação, foi particionada uma tabela de Clientes nos 3 Servidores contendo um total de 74.654 registros.

A tabela 5.1 mostra a quantidade de registros retornados para cada tabela física e o tempo gasto, em segundos, pela consulta em cada sítio, para as três consultas: 1, 2 e 3.

Tabela 5.1: Resultados da Simulação 1

	Consulta 1		Consulta 2		Consulta 3	
Nó	Tot Reg	Tempo	Tot Reg	Tempo	Tot Reg	Tempo
John SQLServer2005	1	00:11	29.995	00:18	3	00:10
SS MySQL	0	00:14	14.657	00:36	10	00:11
John Oracle XE	0	00:11	29.998	00:20	30	00:12
	1	00:36	74.650	01:14	43	00:33

5.2 Simulação 2

Nessa simulação, foi particionada uma tabela de Clientes nos 3 Servidores contendo um total de 149.308 registros.

A tabela 5.2 mostra a quantidade de registros retornados na execução da consulta para cada tabela física e o tempo gasto, em segundos, pela consulta em cada sítio, para as três consultas, 1, 2 e 3.

Tabela 5.2: Resultados da Simulação 2

	Consulta 1		Consulta 2		Consulta 3	
Nó	Tot Reg	Tempo	Tot Reg	Tempo	Tot Reg	Tempo
John SQLServer2005	1	00:11	29.995	00:33	3	00:10
SS MySQL	0	00:14	14.657	00:54	10	00:11
John Oracle XE	0	00:11	29.998	00:19	30	00:11
	1	00:36	74.650	01:46	43	00:32

5.3 Simulação 3

Nessa simulação, foi particionada uma tabela de Clientes nos 3 Servidores contendo um total de 226.962 registros.

A tabela 5.3 mostra a quantidade de registros retornados na execução da consulta para cada tabela física e o tempo gasto, em segundos, pela consulta em cada sítio, para as três consultas.

Tabela 5.3: Resultados da Simulação 1

	Consulta 1		Consulta 2		Consulta 3	
Nó	Tot Reg	Tempo	Tot Reg	Tempo	Tot Reg	Tempo
John SQLServer2005	1	00:10	29.995	–	3	00:10
SS MySQL	0	00:23	14.657	00:60	10	00:11
John Oracle XE	0	00:11	29.998	–	30	00:11
	1	00:44	74.650	00:60	43	00:32

5.4 Discussão

A Figura 5.1 mostra as curvas de tempo para as três simulações e três consultas realizadas. Podemos notar que para as consultas que retornam poucos registros os tempos de execução permanecem constantes independente da quantidade de linhas da tabela física. Para a consulta que retorna um conjunto grande de dados (praticamente a tabela inteira), o tempo de execução cresce linearmente a medida

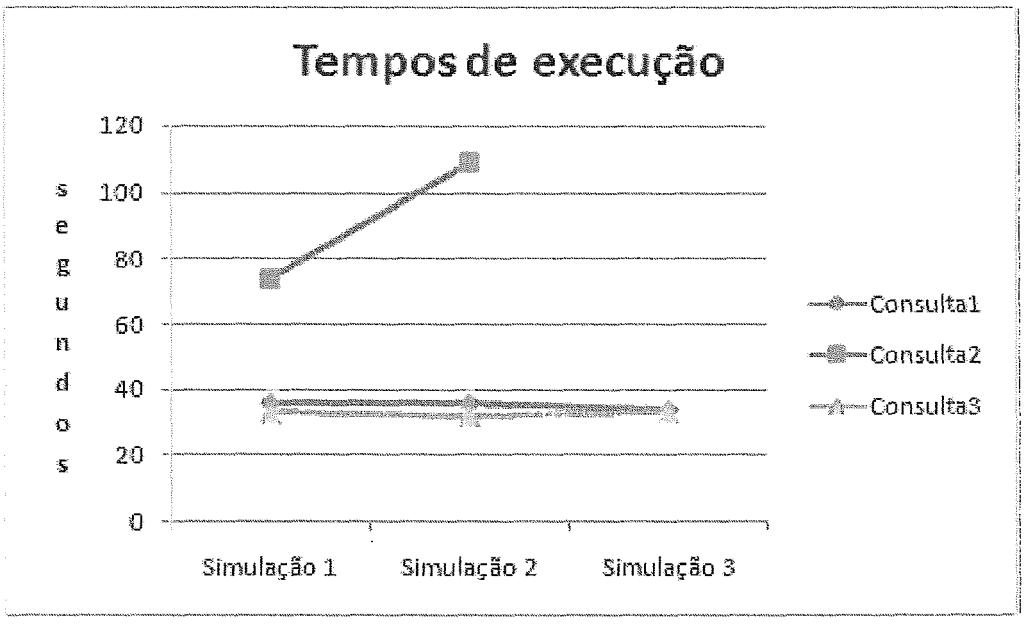


Figura 5.1: *Tempo de execução das Consultas 1, 2 e 3 para os três cenários simulados.*

que aumentamos a quantidade de linhas das tabelas físicas. Isto também se verifica quando mantemos o total de registros em uma única máquina, porém pagamos um alto preço pelo suporte dos bancos no grid.

Na Figura 5.3, o efeito do aumento do número de registros para a consulta 2 é bem notado. Porém se compararmos os tempos de execução mostrados na Figura 5.2 com os tempos de execução da Figura 5.4, observa-se o overhead dos acessos remotos. A Figura 5.1 mostra tempos de execução quase 10 vezes maior do que o tempo de consulta assumindo que os registros estão todos na mesma máquina. Este overhead é devido a 2 fatores principais: a latência da rede e ao protocolo JDBC utilizado para acessar as tabelas físicas remotas. Este estudo pode indicar parâmetros como a quantidade de registros que deve-se alocar por cada tabela física, de forma a minimizar os overheads.

Para o caso de termos bancos de dados realmente federados, deve-se pagar o preço dos overheads, a menos que se use um protocolo mais ameno do que JDBC.

Estes experimentos não pretendem aferir com detalhes as fontes de overhead ou fazer comparações diretas entre sistemas gerenciadores de bancos de dados. Nosso objetivo com estes experimentos foi apenas mostrar a viabilidade do *Pap Meta-data Database System* e testar e depurar a implementação. Possíveis continuações deste

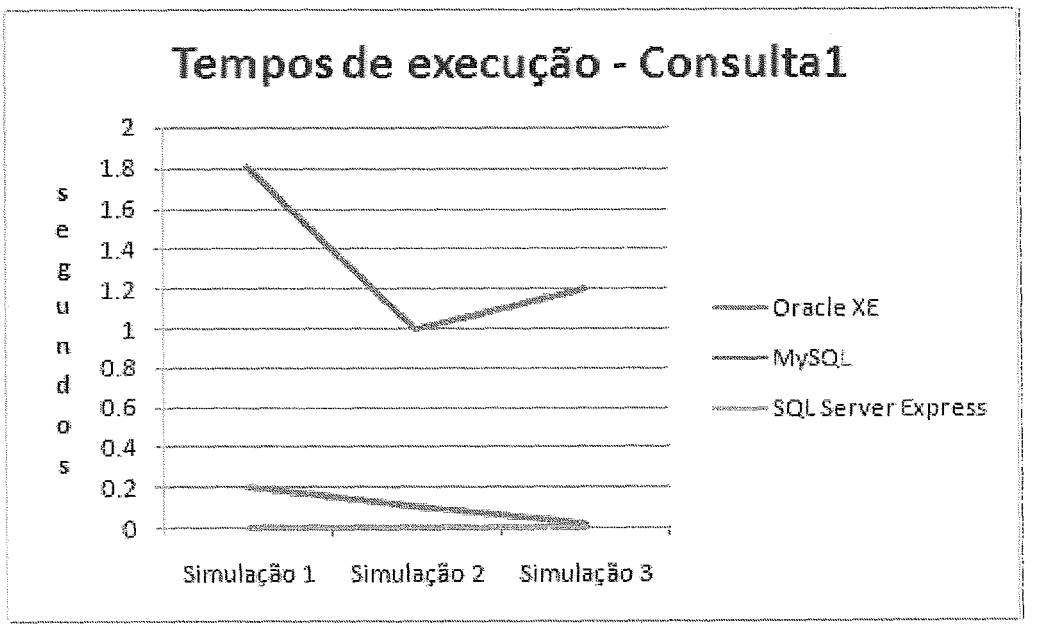


Figura 5.2: *Tempo de execução da Consulta 1 rodando localmente.*

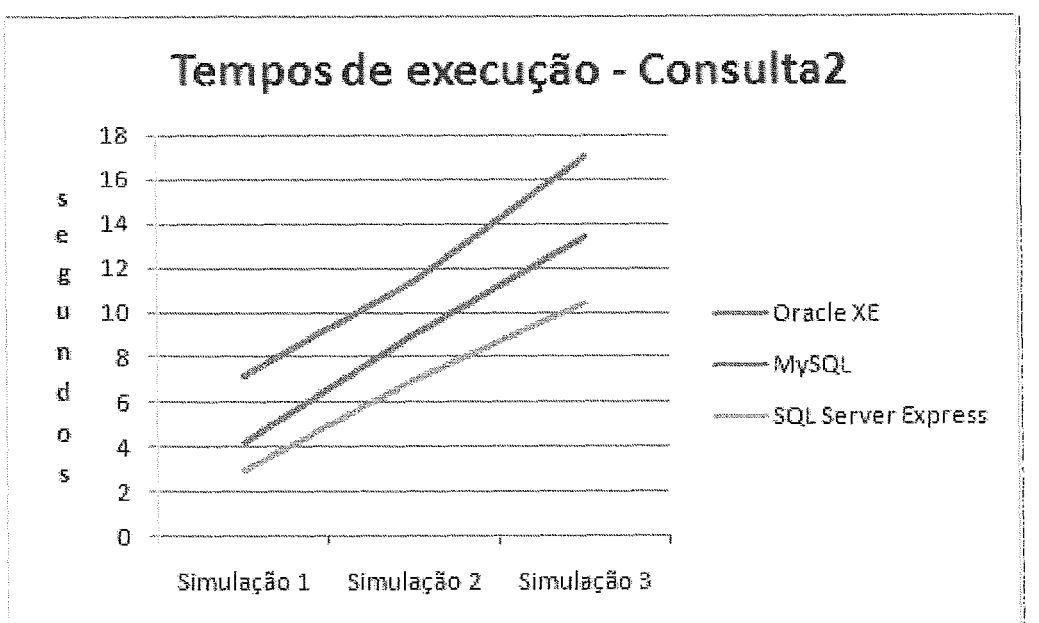


Figura 5.3: *Tempo de execução da Consulta 2 rodando localmente.*

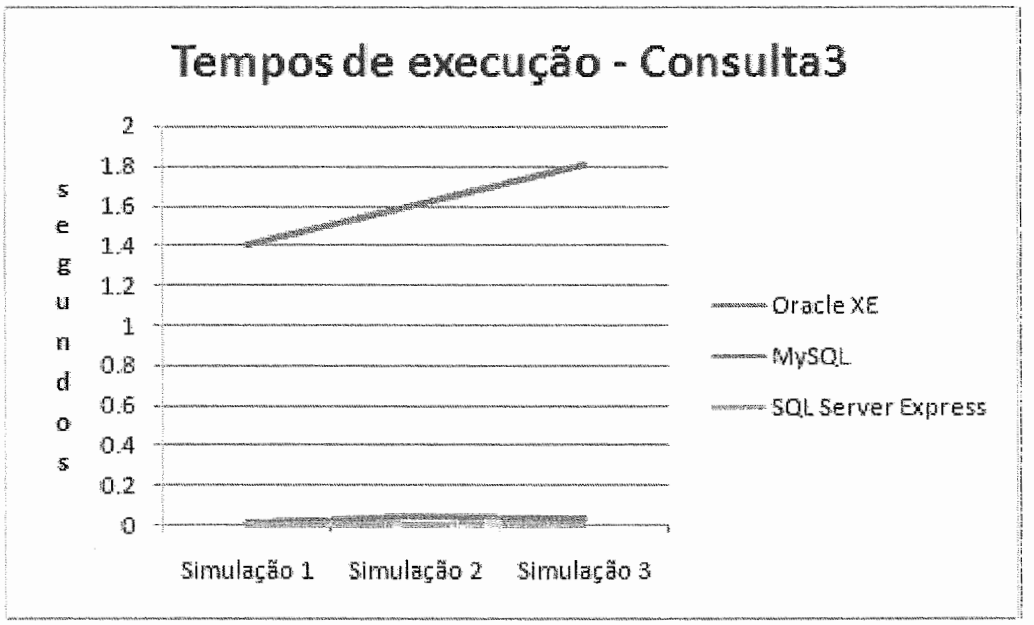


Figura 5.4: *Tempo de execução da Consulta 3 rodando localmente.*

trabalho são discutidas no capítulo 6.

Capítulo 6

Conclusões e Trabalhos Futuros

DataGrids são infraestruturas que disponibilizam dados em ambientes de Grid. Podem conter dados armazenados em arquivos ou em sistemas gerenciadores de bancos de dados (SGBDs). Nosso trabalho focou em sistemas gerenciadores de bancos de dados. Sistemas tais como AMGA, GREIC ou OGSA-DAI, no contexto do Open Grid Forum, procuram disponibilizar dados dispersos geograficamente aos usuários de Grids, porém, normalmente nestas soluções o usuário precisa saber a procedência dos dados, ou localização física, e o SGBD utilizado remotamente, de forma a fazer as consultas de forma apropriada e na linguagem apropriada.

Pacitti *et* al. [23] defendem que um dos principais desafios de DataGrids está relacionado ao gerenciamento de esquemas (Schema management) em DataGrid. Em nosso trabalho apresentamos uma solução baseada em tabelas virtuais, o *PaP Metadata Database System*, onde o usuário do Grid somente conhece um único esquema configurado pelos administradores locais de cada nó do Grid. Conhecendo este esquema, os usuários fazem suas consultas de forma transparente, não precisando sequer saber que os dados estão armazenados em nós remotos. Os usuários locais continuam com a visão local dos dados e fazem suas consultas locais como se este banco de dados não estivesse interligado a outros. Portanto, o banco de dados apresenta-se com dupla visão (local e global) dependendo da natureza do usuário e do tipo de aplicação que está rodando. Este esquema de tabelas virtuais é muito útil, por exemplo, para empresas que possuem vários bancos de dados dispersos geograficamente em várias filiais e precisam integrar os dados de alguma forma para ter uma visão única das tabelas físicas, sem ter que disponibilizar explicitamente a localização de tais tabelas.

Este esquema virtual tem várias vantagens:

- permite que o administrador defina permissões e políticas de acesso aos diversos usuários de cada sítio;
- oferece ao usuário uma única visão dos dados;
- "esconde" do usuário a organização física das tabelas, o que pode reforçar a segurança dos dados;
- pode ser utilizado para armazenar maiores quantidades de dados, visto que as tabelas físicas estão distribuídas.

O *PaP Meta-data Database System* foi implementado e testado. Fizemos um estudo de caso com uma tabela de clientes com mais de 100.000 registros, cujos registros encontram-se dispersos geograficamente. Definimos a tabela virtual, em modo administrador, e testamos mostrando sua viabilidade. Como era de se esperar, consultas no ambiente de Grid têm um overhead quando comparadas com consultas onde considera-se que a tabela física está toda local. Por outro lado, manter tabelas fisicamente distribuídas pode trazer a vantagem de maior capacidade de armazenamento. Além disso, as tabelas podem estar intrinsicamente distribuídas.

Defendemos que a abordagem híbrida utilizada neste trabalho, onde a visão local do usuário é mantida e uma visão global é apresentada aos usuários de Grid contribui como uma solução viável e elegante para o problema de integração de sistemas gerenciadores de bancos de dados em ambientes de Grids.

O sistema implementado apresenta algumas limitações. A seguir discutimos estas limitações e alguns pontos que devem ser tratados na continuação deste trabalho.

6.0.1 Arquitetura

Um ponto de investigação futura para a arquitetura do *PaP Meta-data Database System* é o suporte à paginação. Atualmente, a arquitetura recebe todas os registros que são retornados pela consulta em um único *resultset*. Isso causa uma limitação, porque, quanto maior for o número de registros do *resultset*, mais memória será necessária no servidor. Isso pode ser contornado fazendo paginação dos *resultsets*. Mas um estudo precisa ser feito para identificar o número de registros que deve estar presente em cada paginação.

6.0.2 Suporte a outros Bancos de Dados

Uma limitação atual do *PaP Meta-data Database System* é o suporte a outros bancos de dados. Atualmente ela somente suporta 3 SGBDS (Oracle, MySQL e SQL Server). A intenção é aumentar o suporte a outros bancos de dados. Como o sistema foi desenvolvido de forma modular o fato de adicionar novos SGBDs implica somente o tratamento do novo SGBD na Camada de Acesso a Dados. As outras camadas não precisarão ser modificadas. Como esse "piloto" foi feito utilizando ODBC, o mesmo tem que ser modificado para usar OGSA-DAI.

6.0.3 Segurança

Um dos aspectos que não foi tratado no sistema foi em relação à segurança. Hoje dependemos que a segurança seja feita no lado dos Nós onde estão localizados os Servidores de Banco de Dados. Atualmente a segurança que está sendo feita do lado dos Servidores é uma regra de firewall, que somente permite acesso à porta do SGBD para um determinado IP, que é o IP onde está o Servidor da Aplicação.

A intenção é usarmos OGSA-DAI, e, colocarmos a nossa aplicação para rodar em cima de um OGSA-DAI, que já tem requisitos de segurança atendidos.

Acreditamos que da forma que a aplicação foi desenvolvida, de forma modular, essa mudança não seja muito drástica, pois, só seria necessário mexer nos módulos correspondentes que fazem comunicação com os nós do grid.

6.0.4 Replicação

Atualmente não temos replicação em nosso sistema. É uma característica importante, que deve ser apontada para um projeto futuro. Com essa característica implementada, com dados replicados em diversos nós do Grid, podemos começar a pensar em um escalonamento melhor para a aplicação.

Isto é, atualmente, gravamos sempre um histórico de toda a transação que foi efetuada na Aplicação. Cada processamento nos nós, é registrado, com a informação se o processamento foi feito com sucesso, ou não, e, o tempo gasto no processamento. Essa informação passa a ser valiosa para um critério de escalonamento, no momento que existe replicação, pois, podemos saber quais são os nós mais rápidos, que podem trazer a informação mais rápido.

6.0.5 Balanceamento de Carga

Atualmente o balanceamento de carga é feito de forma manual. Manualmente as tabelas particionadas foram postas nos Nós, e configuradas as Tabelas Virtuais no Sistema.

Mas, o balanço de carga se torna importante no momento que a ferramenta suporta operações de Insert/Update/Delete. Quando for feito um Insert, deve ser necessário escolher qual o melhor Nó para esse dado, de forma que aproveite sempre da melhor forma os recursos disponíveis, não deixando engargalado. No momento do DELETE, também, deve ser pensando um balanceamento de carga de outros nós para o nó que está sendo excluído.

6.0.6 Inserção / Atualização / Remoção de Dados

Estas operações são de grande complexidade em um ambiente distribuído e muitas soluções já têm sido apontadas na literatura. Em ambientes de grid, ainda há muito estudo relevante e interessante para ser realizado, principalmente quando se leva em consideração as latências entre os nós do grid.

6.0.7 Paralelismo

Atualmente, a ferramenta quando recebe uma consulta, faz a conversão da mesma, e verifica quais são os nós envolvidos nessa operação. Para cada nó, ela faz a conversão da consulta para o SGBD correspondente, e, guarda o *resultset*. Após o *resultset* de todos os nós, o XML é montado e retornado para o usuário.

Esse procedimento para cada nó pode ser paralelizado, de forma que otimize o tempo de resposta para o usuário, usando "threads" que o Java suporta.

6.0.8 SOA

SOA é uma Arquitetura Orientada a Serviços. Também é um objetivo da ferramenta, ser um serviço que possa ser consultado por outras aplicações. Não ser necessário o *Web Site*, e, sim, um Serviço disponível que forneça os *Resultsets* ou XMLs requisitados.

Referências Bibliográficas

[1] *GridSphere.* www.gridsphere.org.

[2] *The LHC Project.* http://lhc.web.cern.ch/lhc/.

[3] *MySQL.* http://www.mysql.com.

[4] *Oracle.* http://www.oracle.com/technology/xe/index.html.

[5] *SQL Server 2005 Express Edition.* http://www.microsoft.com/sql/editions/express/default.msp

[6] *Web Services Data Access and Integration - The Core (WS-DAI) Specification, Version 1.0.* http://www.ogf.org.

[7] Bill Allcock, Lee Liming, Steven Tuecke, and Ann Chervenak. Gridftp: A data transfer protocol for the grid. In *OGF Grid Forum Data Working Group*, 2003.

[8] Giovanni Aloisio, Massimo Cafaro, Sandro Fiore, Maria Mirto, and Salvatore Vadacca. Greic data gather service: a step towards p2p production grids. In *SAC 2007*, pages 561–565, 2007.

[9] Mario Antonioletti, Malcolm Atkinson, Rob Baxter, Andrew Borley, Neil P. Chue Hong, Brian Collins, Neil Hardman, Alastair C. Hume, Alan Knox, Mike Jackson, Amy Krause, Simon Laws, James Magowan, Norman W. Paton, Dave Pearson, Tom Sugden, Paul Watson, and Martin Westhead. The design and implementation of grid database services in ogsa-dai. *Concurrency and Computation: Practice and Experience*, 17(2–4):357–376, 2005.

[10] Yigal Arens, Chun-Nan Hsu, , and Craig A. *Query processing in the sims information mediator*, chapter Advanced Planning Technology: Technological Achievements of the ARPA/Rome Laboratory Planning Initiative. AAAI Press, California, CA, 1996.

[11] M. Baker, R. Buyya, and D. Laforenza. The grid: International efforts in global computing. In *Proceedings of the International Conference on Advances in Infrastructure for Electronic Business (SSGRR 2000), (Rome, Italy)*, 2000. July 31 August 6 2000.

[12] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.

[13] Bart Jacob et al. *Introduction to Grid Computing.* IBM Redbooks, 2005.

[14] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15, 2001.

[15] Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: An information integration system. In *in proceedings of 1997 ACM SIGMOD Conference*, pages 539–542, 1997.

[16] Zachary G. Ives, Alon Y. Levy, Daniel S. Weld, Daniela Florescu, and Marc Friedman. Adaptive query processing for internet applications. *IEEE Data Engineering Bulletin*, 23:200–0, 2000.

[17] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32:135–164, 2002.

[18] B. Krishnamurthy, C. Wills, and Y. Zang. On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 169–182, 2001.

[19] Maurizio Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[20] Alfredo Lotar. *XML para programadores ASP.* Axcel Books, 2001.

[21] Patrícia K. V. Mangan. *GRAND: a Hierarchical Model for Application Management in Grid Environments.* PhD thesis, Department of Systems Engineering and Computer Science, Federal University of Rio de Janeiro, COPPE/Sistemas/UFRJ, March 2006.

[22] Mario A. Nascimento. Peer-to-peer: harnessing the power of disruptive technologies. *SIGMOD Rec.*, 32(2):57–58, 2003.

[23] Esther Pacitti, Patrick Valduriez, and Marta Mattoso. Grid data management: Open problems and new issues. *Journal of Grid Computing*, 5(3):273–281, September 2007.

[24] Norman W Paton, Vijay Dialani, Tony Storey, Malcolm P Atkinson, Dave Pearson, and Paul Watson. Database access and integration services on the grid. In *In Fourth Global Grid Forum (GGF 4) Databases and the Grid BOF*, 2002.

[25] Jay Ramachandran. *Designing security architecture solutions.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[26] Nuno Santos and Birger Koblitz. Metadata services on grid. In *Proceedings of Advanced Computing and Analysis Techniques (ACAT'05), Zeuthen, Berlin*, May 2005.

[27] Helbert Schildt. *Java: The Complete Reference, J2SE 5th Edition.* McGraw-Hill/Osborne, 2005.

[28] Heinz Stockinger. Distributed database management systems and the data grid. In *18th IEEE Symposium on Mass Storage Systems and 9th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 17–20, 2001.

[29] Giuliano Taffoni, Sandro Fiore, Giacinto Donvito, Atul Jain, Birger Koblitz, Antonio Calanducci, Claudio Vuerli, Andrea Barisani, Emidio Giorgio, Cristina Aiftmiei, Luciana Carota, Antonio Pierro, Marco Varlato, Massimo Cafaro, Salvatore Vadacca, Alessandro Negro, Roberto Barbera, Giovanni Aloisio, Giorgio Maggi, and Fabio Pasian. How to access databases from egee-ii grid

environment: a comparison of tools and middleware. In *Third EELA Conference*, 2007.

[30] Douglas Thain, Todd Tannenbaum, and Miron Livny. "distributed computing in practice: The condor experience". *Concurrency and Computation: Practice and Experience*, 17:323–356, 2005.

[31] Prem Uppuluri, Sachin Singh, and Uday Joshi. Remotefs: Accessing remote file systems for desktop grid computing. In *Symposium on Applied Computing*, pages 1203–1204, 2007.

[32] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys*, 38:2006, 2007.

[33] James Weaver, Kevin Mukhar, and Jim Crume. *Beginning J2EE 1.4: From Novice to Professiona.*