CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
MESTRADO EM TECNOLOGIA

CELSO HENRIQUE PODEROSO DE OLIVEIRA

UMA PROPOSTA PARA IMPLANTAÇÃO DE ALGORITMO DE PLANEJAMENTO PARA BUSCAS DE BANCO DE DADOS EM GRID COMPUTACIONAL

SÃO PAULO
DEZEMBRO DE 2006

CELSO HENRIQUE PODEROSO DE OLIVEIRA

UMA PROPOSTA PARA IMPLANTAÇÃO DE ALGORITMO DE PLANEJAMENTO PARA BUSCAS DE BANCO DE DADOS EM GRID COMPUTACIONAL

Dissertação apresentada como exigência parcial para obtenção do Título de Mestre em Tecnologia no Centro Estadual de Educação Tecnológica Paula Souza, no Programa de Mestrado em Tecnologia: Gestão Desenvolvimento e Formação, sob orientação do Prof. Dr. Maurício Amaral de Almeida.

SÃO PAULO

DEZEMBRO DE 2006

O48p Oliveira, Celso Henrique Poderoso de

Uma proposta para implantação de algoritmo de planejamento para buscas de banco de dados em grid computacional / Celso Henrique Poderoso de Oliveira. – São Paulo, 2006.

114 f. + anexos

Dissertação (Mestrado) – Centro Estadual de Educação Tecnológica Paula Souza, 2006.

1.Tecnologia da informação – gestão e desenvolvimento. 2. Grid Computing. 3. OGSA-DAI. 4. Banco de dados. 5. Web Services. 6. Inteligência Artificial. I. Título.

CDU 681.3:007

“Você não pode ensinar nada a um homem;
você pode apenas ajudá-lo a encontrar a resposta dentro dele mesmo”.
Galileu Galilei

Resumo

OLIVEIRA, C. H. P. *Uma proposta para implantação de algoritmo de planejamento para buscas de banco de dados em grid computacional.* 2006. Dissertação (Mestrado) – Centro Estadual de Educação Tecnológica Paula Souza.

O objetivo deste trabalho é contribuir com a utilização de grid computacional através da melhoria de serviços de integração dos sistemas gerenciadores de banco de dados neste ambiente.

As Grids Computacionais tem sido utilizadas no meio acadêmico como uma alternativa à utilização de computadores de alto desempenho por permitir que se utilize o poder de processamento de diversos microcomputadores em uma arquitetura distribuída. As aplicações científicas, por serem altamente paralelizáveis, têm tirado proveito deste ambiente. A maior parte das aplicações científicas utilizam arquivos simples para armazenamento de dados.

Alguns processos das aplicações comerciais também podem ser paralelizáveis e, portanto, podem tirar proveito da grid computacional. Contudo o mesmo não pode ser dito do meio de armazenamento. Este tipo de aplicação utiliza sistemas gerenciadores de banco de dados para armazenamento de dados. Integrar os produtos comerciais na grid tem demandado grande esforço da comunidade acadêmica e comercial.

A utilização dos padrões e a adequação aos *middlewares* disponíveis são fundamentais para esta integração. Há mecanismos que permitem vincular os sistemas gerenciadores de banco de dados em uma grid computacional, mas diversos serviços precisam ser desenvolvidos e adaptados.

Foi elaborada uma pesquisa dos fundamentos da grid, dos sistemas gerenciadores de banco de dados e dos algoritmos da inteligência artificial. Com base nos *middlewares* disponíveis, identificou-se aquele que oferece alguns serviços de integração e foi acrescentado um serviço específico para planejamento de utilização de recursos.

Este trabalho estabelece um serviço de planejamento de utilização de banco de dados em uma grid computacional. Para isso utiliza algoritmos específicos da Inteligência Artificial e o *middleware* que fornece os principais serviços de acesso e manipulação em banco de dados.

Palavras-Chave: Grid Computing, OGSA-DAI, Banco de dados, Web Services, Inteligência Artificial

Abstract

OLIVEIRA, C. H. P. *A computational planning algorithm proposal for implantation to database query in grid computing environment*. 2006. Dissertação (Mestrado) - Centro Estadual de Educação Tecnológica Paula Souza.

The objective of this work is to contribute with the improvement of grid computing through integration services of the database management systems in this environment.

The grid computing has been used in academic applications as an alternative to the use of high performance computers because it allows the equivalent power of processing of diverse microcomputers in a distributed architecture. Because scientific applications are highly parallelized, they have taken off advantage of this environment.

Most of the scientific applications use simple archives for data storage. Some commercial applications processes also can be parallelized and, therefore, they can take off the same advantage of grid computing. However the same it cannot be said about the storage. This type of application uses database management systems for data storage. To integrate the commercial products in the grid has demanded great academic and commercial communities' effort.

The use of grid standards and adequacy to middlewares are very important for this integration. The middlewares have mechanisms that allow tying the database management systems in the grid, but many services need to be developed and adapted.

A research of the grid beddings, the database management systems and of the artificial intelligence algorithms was elaborated. Based on available grid middlewares, it was identified the one that offers some integration services with databases and it was added a specific service for scheduling.

This work establishes a computational scheduling service of database systems in the grid. For this it uses specific artificial intelligence algorithms and the middleware that supplies the main services of access and manipulation in database systems.

Keywords: Grid Computing, OGSA-DAI, Databases, Web Services, Artificial Intelligence.

Lista de Figuras

Sumário

1 Introdução

As Grids Computacionais aparecem como uma alternativa bem-sucedida de otimização do uso de recursos em um ambiente distribuído. Por este motivo vários Centros de Pesquisa no mundo todo têm estudado formas de melhorar o desempenho das aplicações e aumentar a quantidade de serviços disponibilizados pelo ambiente.

A idealização desse ambiente se deu a partir da necessidade de se trabalhar com os recursos computacionais como se fossem serviços disponibilizados aos usuários. Atendendo a tendência de se trabalhar de maneira distribuída, os recursos computacionais podem estar dispersos na rede e serem alocados para realizar uma atividade durante o tempo em que estiverem ociosos. O poder computacional dessa solução equipara-se a supercomputadores, pois que são utilizadas dezenas e muitas vezes centenas de computadores para realizar as operações [Chede, 2005].

Para isso criou-se uma infra-estrutura capaz de realizar o intercâmbio de informações entre os recursos. Esta infra-estrutura denomina-se *middleware*. Através do *middleware* é possível alocar dinamicamente os recursos e vinculá-los a tarefas que, depois de processadas, retornam o resultado final a quem solicitou, independente de onde tenha sido realizado o processamento [Foster, 2001].

A evolução das técnicas de desenvolvimento de sistemas permitiu que se criassem serviços espalhados pela rede. Com um serviço de diretórios eficiente, os serviços são localizados e disponibilizados para os usuários. Isso resolve o problema do processamento dos dados.

A maior parte das aplicações em uso na Grid Computacional, por serem sistemas de análise de dados científicos, utilizam arquivos de dados ao invés de banco de dados [Watson et al., 2003].

Em grande parte isso acontece porque os gerenciadores de bancos de dados não estão adaptados para trabalhar adequadamente neste ambiente. O ambiente para o qual foram criados os gerenciadores está direcionado para uso comercial. Há mecanismos de agilização do processo de busca e facilidade na programação de rotinas específicas para tratamento de dados. Está comprovado que as aplicações científicas podem alcançar melhor desempenho se utilizarem bancos de dados por estes serem otimizados para a realização de tarefas intrínsecas do processamento de dados [Nieto-Santisteban et al, 2004].

Estabelecer o uso de bancos de dados em um ambiente de Grid Computacional tem requisitado muito esforço e pesquisa e a integração dos gerenciadores certamente dará um passo decisivo para a utilização da Grid Computacional [Watson et al, 2003].

O desenvolvimento baseado em serviços apresenta-se como o mais adequado para estabelecer um mecanismo de acesso e controle dos recursos de banco de dados [Watson et al., 2003]. Como esta arquitetura indica *o que* deve ser feito e não *como* será implementado, utilizar *web services* para descrever o que pode ser realizado por cada banco de dados é mais adequado dentro do objetivo básico da Grid Computacional que é a distribuição dos recursos computacionais.

O grupo DAIS (*Data Access and Integration Services*) do GGF (*Global Grid Forum*) tem trabalhado na definição dos requisitos de integração dos bancos de dados com a Grid Computacional. A idéia central é criar padrão de acesso a qualquer banco de dados que realize o intercâmbio entre as informações necessárias para as aplicações e o acesso e manipulação dos dados nos bancos de dados comerciais. O DAIS trabalha com um *framework* para os recursos de dados com objetivo de criar um padrão de serviço baseado na OGSA (*Open Grid Service Architecture*) [OGSI-GGF, 2005]. A OGSA é a arquitetura básica para implementação de serviços em uma Grid Computacional.

Esta arquitetura está baseada na SOA (*Service Oriented Architecture*). A proposta é baseada em quatro interfaces: Descrição de Dados, Acesso aos Dados, Fábrica dos Dados e Gerenciamento dos Dados. Segundo esta proposta, qualquer serviço que implemente pelo menos uma dessas interfaces é considerado um serviço de dados. O foco principal do grupo está nas três primeiras interfaces. A quarta interface (gerenciamento dos dados) tem a responsabilidade de especificar como a virtualização dos dados está construída e deve prover as operações de configuração, controle e política de acesso aos dados. A virtualização de dados é a possibilidade de se trabalhar com conjuntos de dados desvinculados da implementação física. É responsabilidade dessa interface definir os serviços que serão utilizados para atender aos requisitos de virtualização, gerenciamento e compartilhamento dos recursos. Desta forma é necessário identificar quais recursos estão disponíveis, programar a utilização dos mesmos e controlar a execução do serviço. Como há outros grupos que trabalham nessa interface, este não é o foco de estudo do grupo.

Em paralelo, um grupo tem desenvolvido um *middleware* que garante o acesso aos serviços disponíveis de cada banco de dados da Grid, mas sem reduzir a capacidade de processamento dos recursos.

O grupo DAIS criou um *middleware*, OGSA-DAI (*Data Access and Integration*), para permitir a disponibilização de serviços de banco de dados. Esta solução baseia-se na OGSA que, por sua vez, está baseado na OGSI (*Open Grid Service Infrastructure*) [OGSA-DAI, 2006]. Desta forma, uma arquitetura baseada em uma infra-estrutura permite que se estabeleçam os mecanismos de integração com os banco de dados e indicam quais serviços podem ser realizados em determinado contexto. O usuário solicita ao OGSA-DAI aquilo que é necessário realizar no banco de dados e este se incumbe de entregar a requisição aos recursos que foram especificados.

1.1 Tema e problema

O tema sob o qual versa o presente trabalho é o uso de Gerenciadores de Banco de Dados em Grid Computacional e o problema é o planejamento de utilização dos recursos com base nos comandos de busca em Banco de Dados distribuídos em Grid Computacional.

1.2 Objetivo

O objetivo deste trabalho é criar um sistema que intercepte as requisições do usuário, pois atualmente o usuário deve indicar o recurso e o comando de busca. Também deverá verificar quais recursos há no ambiente. Ao interceptar o comando e identificar os recursos, o sistema deverá subdividi-lo para que as buscas sejam particionadas e paralelizadas sempre que possível. Depois de escalonada a busca e estabelecido o fluxo de trabalho necessário para atender as requisições do usuário, o sistema deverá submetê-la ao OGSA-DAI que realizará a o trabalho de intercâmbio com os gerenciadores de banco de dados. Espera-se que, ao final do trabalho, o desempenho da pesquisa do usuário será melhor e haverá ganho de produtividade na utilização do ambiente.

1.3 Planejamento com IA

Para realizar o trabalho de planejamento serão utilizados algoritmos de Inteligência Artificial com objetivo de achar a melhor solução possível dentro do espaço de estados disponível. Os principais gerenciadores de bancos de dados fornecem informações do recurso onde está instalado e estas informações servirão para estabelecer uma heurística compatível com a necessidade de processamento da atividade.

O sistema planejador deverá estar baseado nas mais recentes discussões sobre a viabilidade e aplicabilidade da teoria de programação de restrições para programação de recursos em ambientes distribuídos.

1.4 Metodologia

A metodologia utilizada será o teste empírico da solução proposta. Poderá ser medida a eficiência da solução através da submissão e interceptação do comando de busca na OGSA-DAI e na verificação dos comandos de busca submetidos ao ambiente.

1.5 Estrutura do Trabalho

O trabalho está dividido em 5 capítulos:

- Grid Computacional: são comparadas as diversas formas de computação distribuída e apresentados os principais conceitos relacionados à Grid Computacional. Os principais *middlewares* disponíveis serão identificados e será dada uma ênfase especial no OGSA e OGSA-DAI.
- Banco de Dados em Grid: faz uma abordagem conceitual do que se espera de um banco de dados em Grid e as diversas formas que podem ser utilizados os bancos de dados em ambiente distribuído. Identifica uma solução que tem uma proposta parecida com a deste trabalho (OGSA-DQP).
- Planejamento em Grid: será feita uma revisão quanto aos métodos de programação de recursos com variáveis de restrição e serão estudadas formas de implementação da solução proposta. Serão estabelecidos os mecanismos básicos para realizar o planejamento dos serviços na OGSA-DAI e as heurísticas que se adaptam ao modelo proposto.
- Proposta: especifica a proposta apresentada por este trabalho, mesclando todos os conceitos para estabelecer um algoritmo básico para o planejamento proposto.
- Implementação da Solução e Avaliação: discute os aspectos básicos dos resultados alcançados.

Parte I: Fundamentação Teórica

2 Grid Computacional

2.1 Antecedentes

2.1.1 Computação Distribuída

Uma Grid Computacional é uma série de serviços para processamento e armazenamento de dados em um ambiente de Computação Distribuída. Diversas formas de distribuir processamento e armazenamento tem sido desenvolvidas ao longo da evolução da computação. Como os processadores têm evoluído em menor velocidade nas últimas décadas devido às limitações físicas dos componentes utilizados [Dantas, 2005], o processamento distribuído procura contornar esta situação ao ampliar os equipamentos envolvidos no processamento dos dados. Ainda segundo Dantas [Dantas, 2005], "a computação distribuída de alto desempenho pode ser entendida como um segmento da computação que tem como objetivo a melhoria do desempenho das aplicações distribuídas e paralelas, utilizando-se de complexas infra-estruturas computacionais".

Inicialmente os computadores trabalhavam de maneira isolada. Depois passaram a operar em redes departamentais que, por sua vez, se interligavam aos grandes servidores. O resultado desta implementação, contudo, estava aquém do esperado e houve a necessidade de distribuir o processamento e armazenamento de dados. Da descentralização para a distribuição do processamento, processo que ainda está em fase de amadurecimento, os ganhos têm sido mais satisfatórios [Oliveira, 2004].

Inicialmente os servidores departamentais foram interligados em um ambiente único em muitos casos substituindo computadores de grande porte. O processo de agrupamento, conhecido por *clustering*, de servidores possibilita distribuir o processamento e armazenamento. Do ponto de vista do cliente da aplicação há um único servidor fornecendo dados e processamento à rede, mas na realidade os diversos servidores disponíveis estão encapsulados pela tecnologia. Em caso de falha em um servidor, outro automaticamente assume a tarefa que estava sendo executada. O poder de processamento dos servidores passou a competir com equipamentos de maior porte. Naturalmente este processo trouxe maior segurança e velocidade ao processamento dos dados e, então, os sistemas de missão crítica começaram a serem utilizados nestes servidores. A característica básica de um *cluster* é

permitir a distribuição de tarefas entre os servidores, mas esta tecnologia está restrita a um único (e limitado) ambiente físico e tem um único proprietário [Dantas, 2005]. O gerenciamento dos recursos de um *cluster* é centralizado. As principais características de um *cluster*, segundo [Dantas, 2005]:

- Equipamentos dedicados exclusivamente ou não à execução de tarefas.
- Componentes de hardware ou software homogêneos ou heterogêneos;
- Limita-se até as fronteiras da organização;
- Redes de conexão compartilhada, ponto-a-ponto ou híbrida;
- Orientadas a aplicações de alto desempenho.

Desta forma, pode-se entender *cluster* como sistemas de dois ou mais servidores que trabalham como se fossem um único para realizar processamento com alto volume de requisições. É utilizado para processamento paralelo, com balanceamento de carga e tolerância a falhas.

2.1.2 Redes de Computadores

Devido à centralização do processamento e à limitação tecnológica, os computadores de grande porte realizavam todo processamento, armazenamento e distribuição de dados pela rede. Os terminais de usuário não tinham qualquer capacidade de processamento. Serviam apenas como uma interface para o que era processado pelo computador central. Na década de 1970, devido a necessidade de comunicação mais rápida entre os meios acadêmicos de pesquisa, foi desenvolvido o *email*. Os protocolos de rede evoluíram rapidamente e, ainda nesta década, são disponibilizados a Ethernet e o TCP/IP. Este último deu o passo fundamental para a proliferação da era da Internet, já em um passado mais recente. A partir do momento em que os computadores pessoais foram difundidos, cada usuário começou a realizar algum processamento local. Houve necessidade de compartilhar recursos, como discos para armazenamento, impressoras, etc. Foram criadas as redes locais. Na década de 80 estabelece-se o padrão WWW e HTML. Os navegadores da Internet se popularizam com o conceito de *Web* e seus respectivos serviços. Novos padrões se estabelecem para fazer frente a crescente necessidade de integração do ambiente.

Com a ampla utilização da Internet, a banda de comunicação vem sendo aumentada para melhorar o desempenho e acesso à grande rede de computadores. Atualmente boa parte dos centros de pesquisa e até mesmo dos usuários comuns já têm acesso a Internet utilizando banda larga [Oliveira e Almeida, 2005].

A Internet é o ambiente natural para o processamento de uma Grid Computacional. Pode-se dizer que a Grid só conseguiu o avanço atual devido a massificação da Internet.

2.1.3 Sistemas Distribuídos

As aplicações que potencialmente podem ser utilizadas neste ambiente são as distribuídas e as paralelas. Aplicações distribuídas são as que utilizam recursos distribuídos, portanto podem ou não guardar relação entre si. Neste caso, uma aplicação pode ser executada independente das outras aplicações e o desempenho delas será completamente particular. Já as aplicações paralelas são as que se subdividem em porções menores. Estas porções são distribuídas entre processadores distintos e, naturalmente, guardam interdependência entre si. Ao final do processamento a união destas porções produzirá um único resultado [Dantas, 2005].

Ao se utilizar diversos recursos computacionais dispersos na rede e com pouco uso, obtém-se ganhos consideráveis sobre o investimento realizado. De qualquer forma, é necessário que se dedique um esforço para resolver alguns problemas específicos de um ambiente distribuído [Dantas, 2005]:

- Segurança: aplicativos executados remotamente podem estar expostos a indivíduos ou processos não autorizados.
- Retardo de comunicação: a comunicação entre os servidores pode ser dificultada ou impedida devido a restrições na rede.
- Disponibilidade dos recursos: em ambientes distribuídos pode-se, eventualmente, perder o contato com os servidores.
- Compatibilidade dos pacotes de software: a diferença entre processamento, sistema operacional e compiladores, entre outros, pode dificultar ou até mesmo impedir a realização do serviço.

2.2 Grid Computacional

Ainda na década de 1960, imaginou-se que o acesso às informações e ao processamento seria realizado da mesma forma como hoje em dia utilizamos a energia elétrica ou o telefone. Da mesma forma como não se precisa de uma usina geradora de energia em casa para ter acesso à energia elétrica, também não haveria necessidade de um computador para ter acesso ao processamento e armazenamento de dados [Oliveira, 2004].

Este tipo de computação deveria ser elaborado exatamente como as malhas de energia elétrica. Assim, quando fosse necessário utilizar um recurso, simplesmente o usuário deveria

se conectar a grandes servidores que forneceriam todo poder de processamento e armazenamento. Haveria uma taxa que seria paga de acordo com o uso, exatamente como acontece nos serviços de telefonia ou energia elétrica.

Para uma primeira definição, pode-se entender uma Grid Computacional como um grupo de recursos heterogêneos, distribuídos e integrados que compartilha diversos recursos como se fossem um único e que utilizam redes de altíssima velocidade [Foster, 2001]. É um ambiente em que se permite balanceamento de carga, segurança de acesso e é tolerante a falhas.

A diferença básica entre uma Grid e um *cluster* está na descentralização, amplitude e heterogeneidade dos recursos e serviços que são utilizados para prover o processamento e armazenamento de dados em uma rede. O gerenciamento de uma Grid é realizado de maneira descentralizada, normalmente pelo dono do recurso. Em [Dantas, 2005] há uma comparação com os serviços de telefonia: cada operadora seria um *cluster*, pois gerencia seus recursos de maneira centralizada. A interligação entre dois usuários de diferentes operadoras se dá através de protocolos combinados entre elas. Este é, portanto, um serviço prestado, exatamente como os serviços de uma Grid Computacional. Não há um gerenciamento centralizado, pois cada operadora responde individualmente pelo serviço que presta e disponibiliza.

De qualquer forma, deve estar claro que o limite entre um e outro é tênue e pode ser facilmente confundido. Pode-se entender que, baseado nas definições clássicas de uma Grid Computacional, um único computador pessoal é uma Grid ou que, por estar conectado a uma rede local, esteja em uma Grid de *cluster*s ou ainda que, por permitir o armazenamento em diversos recursos, esteja em uma Grid de armazenamento [Foster, 2002]. Como se nota, é possível obter diversas definições de Grid Computacional e todas têm um certo grau de coerência mesmo que completamente díspares.

2.2.1 Abrangência

Uma Grid Computacional pode ser formada a partir de diversos níveis de utilização. Conceitualmente divide-se em [Chede, 2005]:

- **Local**: são Grids internas de uma única empresa. A configuração pode ser vista como um *cluster*, com gerência de ambiente único [Dantas, 2005]**.** Também conhecida por *intragrid*, *enterprise* ou *campus*.
- **Regional**: são Grids formadas entre empresas parceiras ou que tenham interesse comum. Nesse caso deve haver mais de uma organização envolvida. Também conhecida por *extragrid* ou *partner*.

- **Global**: são Grids amplas, formadas por redes interligadas com abrangência em diversas localidades. Pode ser composta por centenas ou milhares de organizações. Também conhecida por *intergrid.*

2.2.2 Identificação

Para determinar se uma tecnologia está vinculada à Grid Computacional ou não, deve-se identificar [Foster, 2002]:

- Recursos não devem estar subordinados a um controle central: uma vez que a Grid integra diversos recursos, usuários e domínios, caso esteja vinculado a um controle único, tem-se um sistema de gerenciamento local e não uma Grid.
- Utilização de protocolos e interfaces padronizadas, de uso múltiplo e aberto: como a Grid trabalha com recursos heterogêneos, é necessário adotar padrões abertos de comunicação. Do contrário haverá um sistema de aplicação específica.
- Entrega de serviços de alta qualidade: o objetivo da Grid é fornecer poder de processamento e armazenamento onde se integrem diversos recursos dispersos pela rede. Caso o resultado final não seja melhor que a soma das partes, não faz sentido utilizá-la.

2.2.3 Uso

Podemos dividir as aplicações que podem extrair vantagens na utilização de uma Grid Computacional em dois grandes grupos:

- **Científicos**: utilizados para sintetização de teorias e modelos, representa a maior quantidade de Grids ativas atualmente. O desafio tem sido o de unir diversos conhecimentos e processar as informações em altas velocidades. Dado que as fontes de informação encontram-se dispersas, é necessário que haja integração entre elas para se chegar a resultados satisfatórios [Gil, 2004].
- **Uso Geral**: diversas empresas e segmentos poderão fazer uso da Grid Computacional para otimizar o uso dos recursos computacionais, diminuindo significativamente o investimento em novos equipamentos. Além disso, alguns processos empresariais tendem a utilizar altas cargas de processamento: simulações e cenários estão entre estas aplicações [Watson, 2003] e [Nieto-Santisteban et al, 2004].

Uma Grid Computacional deve ter o menor número possível de pré-requisitos, visto que o objetivo é integrar recursos heterogêneos. Por agregar estes recursos, é possível melhorar a colaboração entre os diversos usuários e recursos disponíveis.

2.2.4 Vantagens do Ambiente

Os sistemas acadêmicos e profissionais estão cada vez mais complexos, tanto em funcionamento quanto na necessidade de gerenciamento. Manter uma estrutura de manutenção e atualização de equipamentos não é fácil e muito menos barato. Ao aproveitar os recursos em sua totalidade, espera-se que a necessidade de aquisição de grandes servidores seja menor. Atualmente em algumas Grids Computacionais, é possível atingir o poder de processamento de um supercomputador ao se utilizar centenas de computadores interligados [Chede, 2005].

Acrescente-se o fato de que a maior parte do parque instalado dos computadores de qualquer organização, seja ela científica ou não, está subutilizada. Normalmente utiliza-se, em média, 20% da capacidade de processamento do parque instalado de computadores [Chede, 2005]. Durante o pico de utilização, os servidores ultrapassam 70% da capacidade de processamento. Isso reflete uma realidade no mercado atual de servidores: compram-se servidores pelo pico de utilização e não pela média. Ao utilizar completamente o potencial de todos os computadores da organização, será possível adquirir recursos computacionais pela média e não mais pelo pico [Oliveira, 2004].

Os benefícios da utilização de uma Grid Computacional estarão focados em aspectos de desempenho, qualidade e custo, visto que, será utilizada a totalidade dos recursos de processamento da organização. Além disso, a disponibilidade dos recursos, a escalabilidade das aplicações e o gerenciamento serão melhorados devido às características distribuídas desta tecnologia.

2.3 Arquitetura

Devido ao ambiente heterogêneo, é necessário que se estabeleçam padrões para permitir a interoperabilidade entre os recursos das organizações virtuais. Mesmo as tecnologias que são utilizadas para computação distribuída, como *Enterprise JavaBeans* (EJB) e *Common Object Request Broker Architecture* (CORBA), não possuem mecanismos adequados para organizações virtuais.

Os *Storage Service Providers* (SSP) permitem o compartilhamento do armazenamento, mas os clientes estão interligados por *Virtual Private Network* (VPN). O que se quer com uma Grid é o compartilhamento dos recursos ociosos de maneira descentralizada. Apesar de ser possível realizar as mesmas atividades através dos mecanismos da computação distribuída, esta tem como principal característica a centralização, além de possuir menor flexibilidade e controle de compartilhamento.

2.3.1 Organização Virtual

Entende-se como um grupo de organizações ou indivíduos que compartilham recursos de forma controlada. Desta forma os membros da comunidade podem colaborar entre si para atingir o objetivo compartilhado [Foster, 2001].

Ao agir por interesses comuns, empresas e pessoas podem definir o que, como e quando compartilhar tais recursos. Isso quer dizer que teremos uma Grid apenas quando permitimos que outras pessoas ou empresas acessem nossos recursos (processamento, armazenamento, etc.) de maneira controlada e segura.

As Organizações Virtuais são formadas por interesses comuns. Pode-se considerar as empresas, centros de pesquisa e universidades como exemplos de Organizações Virtuais [Dantas, 2005]. Desta forma, o compartilhamento dos recursos serão mais próximos às redes ponto-a-ponto (P2P) do que às do ambiente cliente-servidor. Um computador cliente pode servir a um computador central, trocando completamente sua função na rede. Um computador cliente pode se utilizar de outro computador cliente para realizar o processamento que por sua vez pode solicitar dados que estejam no computador central [Foster, 2001]. Este é o aspecto mais importante de colaboração em uma Organização Virtual.

É necessário que haja controle e informações sobre os recursos para que a Grid tenha condições de encaminhar serviços para serem executados. Métricas de desempenho, prioridade, expectativas e limitações de cada recurso, escalonamento e balanceamento de tarefas são extremamente importantes quando se trabalha com Organizações Virtuais [Foster, 2001].

Fica claro que a utilização dos recursos computacionais é mais racional ao se utilizar o conceito de Organização Virtual e Grid. Há possibilidade de um uso mais adequado para a largura de banda, os processadores remotos, espaço para armazenamento e memória [Dantas, 2005].

Em [Foster, 1999] há uma proposta de modelo de arquitetura para Grids (figura 2.1).

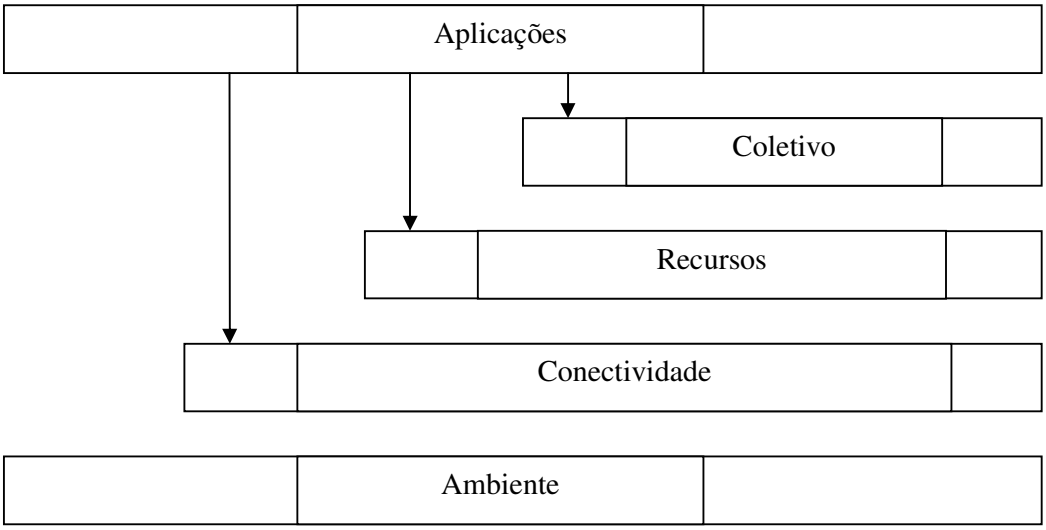


Figura 2.1: Arquitetura de Grid Computacional [Foster, 1999]

Os cinco níveis propostos são:

- **Ambiente**: responsável pela operação local em cada um dos recursos. Implementa mecanismos de negociação, solicitação, gerenciamento e monitoramento do recurso. Monitora a qualidade do serviço (QoS). Fazem parte deste nível recursos computacionais (estado e programação de equipamentos e *softwares*), sistemas de armazenamento (espaço disponível e utilização de banda), catálogos (busca e atualização de dados), recursos de rede (características e carga) e sensores. Há independência entre as funções e as operações de compartilhamento [Foster, 2001].
- **Conectividade**: estabelece, através de protocolos, a comunicação (TCP/IP, UCP e DNS) e autenticação (inclusive criptografia) para transações de rede. Como está entre as camadas *ambiente* e *recursos*, é responsável por construir os serviços de comunicação através da implementação de mecanismos de segurança e controle de acesso (usuários e recursos). Algumas das características mais importantes da camada de conectividade são [Foster, 2001]:
    - Entrada única (*single sign-on*): usuários devem se autenticar uma única vez e ter acesso a todos recursos necessários na Grid, como em uma organização virtual.
    - Delegação: sistemas devem ser executados sob os direitos do usuário e deve haver possibilidade de delegar a outros programas os mesmos direitos.

  - Integração com segurança local: não deve sobrepor os direitos específicos do recurso utilizado.
  - Relacionamento garantido: caso um usuário necessite utilizar mais de um recurso para os quais os direitos tenham sido atribuídos, isso deve ser feito sem que os recursos tenham que interagir entre eles para garantir este acesso. O controle é da Grid e não dos recursos.

- **Recursos**: implementa protocolos de comunicação e autenticação que requisitam funções do nível de ambiente para controlar os recursos locais. Estabelece o acesso aos recursos utilizando-se da camada de conectividade. Trabalha com cada recurso individualmente. Esta camada é responsável pela negociação, inicialização, monitoramento, controle, bilhetagem e pagamento pelos recursos utilizados. Há duas classes nesta camada [Foster, 2001]:
  - Protocolos de Informação: obtém informações da estrutura como configuração, carga atual e política de utilização.
  - Protocolo de Gerenciamento: negocia o acesso ao recurso compartilhado, como requerimentos, criação ou acesso aos dados.
- **Coletivo**: responsável pela troca de mensagens entre os recursos. Estabelece serviços de diretório (descoberta dos recursos disponíveis na organização virtual), alocação (programação dos recursos utilizados), diagnósticos (falhas, ataques e sobrecarga), replicação (armazenamento, disponibilidade, desempenho, tempo de resposta e custo), colaboração (troca de grande volume de dados entre organizações virtuais) e autorização (política da organização virtual de acesso aos recursos). Utiliza-se de *Application Programming Interfaces* (API) e *Software Development Kits* (SDK) específicas para facilitar a programação no *middleware* [Foster, 2001].
- **Aplicação**: são os sistemas executados pelos usuários e que utilizam a organização virtual para realizar as operações. As aplicações podem requisitar serviços de qualquer outra camada, através de protocolos ou APIs e SDKs. Isto é especificado pelo *framework* utilizado [Foster, 2001].

2.4 Desenvolvimento de Sistemas em ambientes distribuídos

Em [Dantas, 2005], há uma proposta para classificação de ambientes de software distribuídos. Conforme salienta o autor, não é uma taxonomia, mas permite entender a maneira como os sistemas distribuídos são desenvolvidos. A classificação é feita por: Ambiente de Programação, Ferramentas e *Middleware*.

2.4.1 Ambiente de Programação

É natural que o desenvolvimento de um sistema que será operado em um ambiente centralizado é diferente de um que atue em ambiente distribuído. Os recursos, em um ambiente distribuído, nem sempre estão à disposição do interessado. Alguns fatores afetam o desempenho das aplicações distribuídas: segurança e autenticação, largura de banda, rede de comunicação entre computadores clientes e servidores, diferentes configurações de sistemas operacionais, compiladores das linguagens de programação, etc.

Os serviços *Web* (*Web Services*) são utilizados na Internet para compartilhar a lógica de negócios e são úteis para desenvolver sistemas distribuídos. Estes serviços têm como característica básica a utilização de protocolos que facilitam a execução das aplicações em redes, sistemas e equipamentos heterogêneos. Dentre os principais padrões, destacam-se o *HyperText Transfer Protocol* (HTTP) , a linguagem *eXtensible Markup Language* (XML) e o *Web Service Description Language* (WSDL).

O XML é uma linguagem de marcação de dados. Através da marcação é possível definir, transmitir, validar e interpretar os dados. Os principais protocolos utilizados são: *Simple Object Access Protocol* (SOAP) e *Universal Description Discovery and Integration* (UDDI).

O SOAP é um protocolo que tem como base o XML e é utilizado para troca de mensagens pela Internet. As mensagens podem ser transmitidas via HTTP, SMTP e MIME. Ao utilizá-lo em sistemas distribuídos, deve-se levar em consideração o fato deste não ser um protocolo orientado a estado (*stateless*) e não ter coleta de lixo [Dantas, 2005]. Isso pode gerar problemas para identificação da sessão. A mesma preocupação deve ser tomada com a segurança, visto que não há uma preocupação específica do protocolo nesse aspecto.

Já o UDDI é utilizado como um diretório distribuído. Isso faz com que sejam possíveis a localização e descoberta dos serviços através da Internet. Com um servidor UDDI é possível que uma empresa possua múltiplas versões dos serviços disponíveis e estabeleça critérios de segurança [UDDI, 2005].

O XML é utilizado para marcar os dados, o WSDL para descrição, o SOAP para transferência e o UDDI para listar os serviços.

Os serviços são importantes porque indicam os protocolos e implementam um comportamento. Serviços padronizados para cada uma das necessidades da Grid permitem aos participantes de uma Organização Virtual saber o que se pode realizar com o recurso oferecido.

Por fim, as APIs e SDKs são importantes para que os desenvolvedores possam abstrair as necessidades dos recursos, facilitando a criação de aplicações disponíveis para a Grid. Percebe-se que, desta forma, os serviços trabalham em conjunto com os protocolos e não como uma alternativa à criação destes [Foster, 2001].

Uma das alternativas à utilização de serviços *Web* é o *Parallel Virtual Machine* (PVM). Este ambiente é um pacote que realiza a organização de computadores com arquiteturas heterogêneas ou não para programação paralela. É divido em duas camadas: uma biblioteca para programação paralela e o ambiente computacional. O desenvolvedor atua na camada da biblioteca e o programa resolve a distribuição no ambiente. Possui um aspecto visual que facilita a identificação e acompanhamento dos serviços executados. Podem ser utilizados nos sistemas operacionais Windows e Unix [Dantas, 2005].

Outra alternativa é a utilização do *Message Passing Interface* (MPI) que tem as mesmas características do PVM e também é utilizada para criar aplicações que sejam executadas em paralelo. O MPI suporta o modelo *Single Program Multiple Data* (SPMD) e tem pode ser utilizado em diferentes ambientes operacionais.

2.4.2 Ferramentas

De acordo com [Dantas, 2005], pode-se considerar ferramentas de software os pacotes que auxiliam a instalação distribuída dos programas, gerenciamento dos recursos, escalonamento, balanceamento, monitoração e acompanhamento dos processos. Pode ser facilmente confundida com o *middleware*, pois este é composto por ferramentas de um escopo maior.

As ferramentas são responsáveis por gerenciar um ambiente composto por diversos sistemas operacionais que, por sua vez, não têm responsabilidade além do gerenciamento de seus próprios recursos e processos. O tipo de aplicação que obtém melhores resultados nesse ambiente são as que possuem uma variação sistemática de carga dos computadores e as que fazem muitas operações de entrada e saída de dados [Dantas, 2005]. Essas ferramentas são conhecidas por *Resource Management and Systems* (RMS). Alguns exemplos: LSF (*Platform Computing*), Codine (Sun), Loadlever (IBM) e Condor (Universidade de Wisconsin).

2.4.3 Middleware

O *middleware* guarda alguma semelhança com as ferramentas, mas são mais completas. Com isso é possível realizar as tarefas em um ambiente amigável e com maior controle e recursos. Este ambiente permite realizar mais facilmente as tarefas de submissão de

aplicações, gerenciamento de recursos e serviços distribuídos, controle de acesso e permissão de instalação de pacotes [Dantas, 2005].

Dentro dessa categoria há sistemas de imagem única. A função básica desses sistemas é abstrair complemente a execução (em que computador ou recurso) se executa a aplicação. Com isso é possível reduzir o tempo despendido no gerenciamento do sistema e aumentar o grau de confiança na execução do serviço. Alguns exemplos: OSCAR (*Open Cluster Group*), Glunix e Sprite (Universidade de Berkeley), OpenMosix (Universidade Hebrew), Unixware (SCO) e Solaris MC (Sun).

2.5 Middleware para Grid Computacional

Entende-se *middleware* como sendo um software que conecta duas ou mais aplicações. É um conceito diferente de importar e exportar dados, pois o *middleware* se conecta às aplicações, lendo e enviando as informações necessárias para o processamento.

De acordo com Foster [Foster, 2001], um *middleware* para Grid seria a união de protocolos, serviços, APIs e SDKs.

A Grid Computacional é, portanto, um meio de comunicação, troca de informações e compartilhamento de recursos entre aplicações.

É fundamental a existência de *middleware* para permitir a interoperabilidade entre as organizações virtuais. As organizações virtuais, para existir, precisam de mecanismos de descoberta, identidade, autorização e compartilhamento [Foster, 2001].

Há alguns *middlewares* disponíveis. Pode-se notar que alguns trabalham em uma camada mais baixa de serviços e outros complementam com serviços específicos.

Entre eles, destacam-se: *Globus Toolkit*, *Storage Resource Broker*, *Gridbus*, *Alchemi* e *Legion*.

O *Alchemi* é um *framework* que permite criar um ambiente de Grid com o mínimo de esforço (*plug & play*), segundo o próprio grupo define. É uma ferramenta de *open-souce* (código aberto) que permite a execução de aplicações na Grid.

O *GridBus* é um acrônimo de GRID e BUSiness. O objetivo é criar tecnologias que integrem Grids com negócios. Espera-se disponibilizar um ambiente baseado em especificações *open-source* adequado para aplicações de *eScience* e *eBusiness* em uma visão de arquitetura orientada a serviços (SOA) e computação colaborativa. Utiliza recursos disponibilizados pelo *Globus Toolkit* e do *Alchemi* para aprimorar serviços específicos do escopo do projeto [GridBus, 2005].

O *Globus* é um conjunto de ferramentas que realiza o trabalho de conectar e gerenciar recursos. É o principal *middleware* utilizado e é formado por um consórcio de várias empresas de tecnologia interessadas em disseminar a utilização das Grids Computacionais. Utiliza o GridFTP como mecanismo básico de acesso e transferência de dados e foi integrado a um banco de dados chamado *Objectivity* apenas para copiar objetos armazenados. É integrado ao *Grid Security Infrastructure* (GSI) que provê os serviços de segurança da Grid, entre eles o controle de acesso [Globus, 2005]. O *Globus Resource Allocation Manager* (GRAM) é responsável pela interface do *middleware* com os recursos locais e o *Metacomputing Directory Services* (MDS) realiza a localização e informação dos recursos (serviço de diretório).

O *Storage Resource Broker* (SRB) foi projetado para permitir que aplicações pudessem acessar informações distribuídas. É um *middleware* cliente-servidor que foi projetado para permitir que as aplicações pudessem acessar informações distribuídas em recursos heterogêneos. Possui um servidor de *metadados* que permite vincular o nome lógico com a localização física dos arquivos e contém mecanismos de cobrança por uso e acesso além de guardar informações de usuários e recursos. Este *middleware* é capaz de selecionar, criar e manter réplica dos dados, requisito importante quando se trabalha com distribuição de dados. Apesar de haver alguma tendência a utilizá-lo em conjunto com bancos de dados para distribuição de informações, há algumas limitações que precisam ser vencidas, como a transferência dos dados para o servidor. Esse processo tende a gerar perda no desempenho e diminuir a escalabilidade das aplicações [SRB, 2005].

O *Legion* é um projeto da Universidade da Virgínia que utilizou o paradigma da orientação a objeto para processamento paralelo e distribuído. Segundo o sítio Legion [Legion, 2005], o sistema tem alta escalabilidade, facilidade de programação, tolerância a falhas, segurança e autonomia. O sistema permite criar um computador virtual, com os recursos do computador recebendo o tratamento como se fosse objeto. A interpretação de um arquivo é dada da mesma maneira, ou seja, como objetos de persistência [Dantas, 2005].

Os demais *middlewares* têm utilização e características específicas que não diferem muito do objetivo dos anteriores.

A grande vantagem na utilização de um *middleware* é que, devido a padronização, pode-se agregar diversos recursos à Grid. Em Wang [Wang et al., 2005], identifica-se um equipamento que serve como um grande armazenador de dados integrado ao *middleware* Globus. Por ter integração com o GSI e com o GridFTP, permite o acesso de clientes às informações de maneira mais eficiente e segura.

2.5.1 Open Grid Services Architecture (OGSA) e Open Grid Services Infrastructure (OGSI)

A infra-estrutura (OGSI) trabalha em conjunto com a arquitetura (OGSA) da Grid para permitir a interação entre os aplicativos dos usuários e os serviços *Web* (*Web Services*).

Pode-se entender a OGSI como um conjunto de especificações WSDL que define padrões de interface e ações para a OGSA [Dantas, 2005]. Os serviços podem ser persistentes (longa duração, com busca de aplicações distribuídas) ou transientes (curta duração, com criação e destruição dinâmica). Os serviços *Web* concentram-se no primeiro grupo e os serviços de Grid em ambos.

Um elemento fundamental nos serviços de Grid é o *Service Data Element* (SDE). O SDE é responsável pela consulta, adição e remoção das instâncias de serviço de Grid.

OGSA é um conjunto de definições, conceitos e tecnologia de serviços para *Web*. O objetivo principal é estabelecer uma padronização no desenvolvimento de software para Grids. Este padrão faz com que os serviços (compartilhamento, acesso e gerência de recursos) de uma Grid possam ter uma abordagem comum. Com isso é facilitado o desenvolvimento de aplicações. Outro ponto importante é que, com esta arquitetura, é possível garantir uma alta disponibilidade, acesso seguro e distribuído além de uma grande escalabilidade dos recursos [Dantas, 2005]. A figura 2.2 mostra o modelo OGSA.

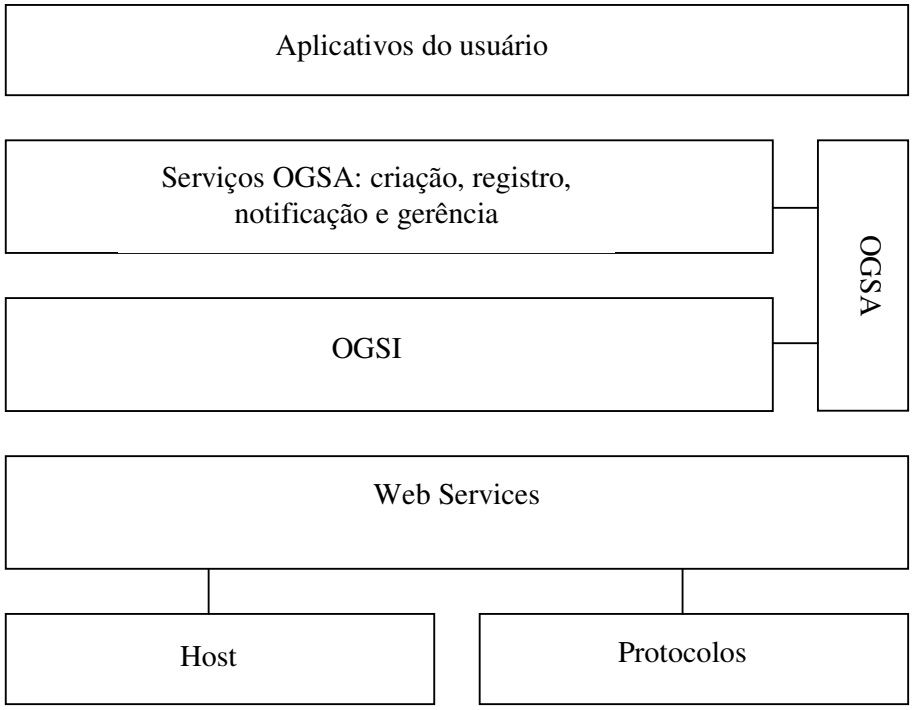


Figura 2.2: Modelo OGSA. Fonte: OGSA.

A opção pelos *Web Services* estão intimamente relacionados à possibilidade de integração e distribuição dos serviços, mesmos quando os recursos são heterogêneos. Os *Web Services* são independentes de linguagem de programação e sistema operacional e, no caso da OGSA, é utilizado para criar um *framework* para sistemas distribuídos baseados na *Open Grid Services Infrastructure* (OGSI) [OGSI, 2003].

A OGSA fornece a descrição do ambiente utilizando *Web Services Description Language* (WSDL) e uma coleção de interfaces padrão dos serviços. Este mecanismo permite um padrão para criação, identificação, comportamento e descoberta de instância de serviços. A coordenação da distribuição dos serviços fica sob responsabilidade desta arquitetura. Por sua vez o WSDL utiliza o XML para descrever os *Web Services* [Watson, 2002]. Também são utilizados: protocolo *Simple Object Access Protocol* (SOAP) e *Web Services Inspections Language* (WSInspection). A OGSI estende as definições de WSDL e XML para agregar conceitos de serviços *Web* orientadas a estado (*stateful*)**,** notificação assíncrona para troca de estado, referência a instâncias de serviços, coleções de instâncias de serviços e estado dos serviços de dados [OGSI, 2003].

Na OGSA, define-se um modelo que represente os recursos reais e lógicos da Grid. Esse modelo é composto pelos elementos básicos dos serviços disponibilizados, como processadores, unidades de disco e memória.

Os serviços de Grid (*Grid Services*) são especificações de interfaces e procedimentos [Dantas, 2005]. Todo *Grid Service* é um *Web Service*, mas o inverso não é verdadeiro [OGSI, 2003].

Para se ter uma visão ampla do *framework*, serão especificadas as principais funções disponíveis e importantes para este trabalho.

Os principais conceitos são [OGSI, 2003]:

- *Grid Service Description*: indica como o cliente pode interagir com as instâncias de serviços. Serve para descrever a interface de serviço (*portType*) e para descoberta de serviços (provavelmente através da *semantic Web*).
- *Grid Service Instance*: pode utilizar mais que uma descrição (GSD). Cada instância retém informações que indicam como interagir com o serviço. Possui um ou mais *Grid Service Handle* e uma ou mais referências (*Grid Service References*).
- Declaração do tempo de vida: cada instância pode ter um tempo de vida diferente. Há atributos XML específicos para indicar o tempo de vida esperado para os serviços.

As principais propriedades da infra-estrutura são [OGSI, 2003]:

- *GridService Factory* (GSF): responsável por criar uma nova instância do serviço. Como definição da instância, é possível atribuir parâmetros no momento da criação.
- *GridServiceHandle* (GSH): é o identificador único do serviço de Grid retornado na criação da instância. É apenas o nome, portanto não permite a localização e comunicação do cliente com o serviço. É válido para todo ciclo de vida da instância do serviço.
- *GridServiceReference* (GSR): estabelece a comunicação com os serviços da Grid. Contém o protocolo e o endereço de rede que permite localizar o serviço. Nota-se que esta propriedade funciona como um ponteiro que indica a localização do serviço. Como depende do protocolo, se utilizar RMI/IIOP, o GSR assumirá o formato IOR. Se utilizar SOAP, assumirá o respectivo documento WSDL. Pode se tornar inválido durante o ciclo de vida do serviço. Por este motivo, um GSH pode se relacionar a diversos GSR.

A resolução que deve ocorrer entre GSH e GSR é realizada pelo *HandleResolver portType*.

Um serviço de Grid é uma instância endereçável que implementa uma ou mais interfaces *portType* de um WSDL [OGSI, 2003]. Todos os serviços de Grid devem implementar esta interface. Esta *portType* é semelhante a um ***base Object class*** de uma linguagem orientada a objetos: encapsula o procedimento padrão do componente. O procedimento encapsulado está relacionado a busca e atualização do conjunto de serviço de dados. Portanto, realiza operações básicas de gerenciamento na instância do serviço da Grid [OGSI, 2003].

Esta interface é responsável por especificar quais conjuntos de serviços, dados e elementos descrevem as propriedades e capacidades dos serviços da Grid [Dantas, 2005].

Alguns exemplos de declarações de serviços de dados são:

- *interface*: estabelece relação com a instância do serviço;
- *gridServiceHandle*: indica zero ou mais GSH para a instância;
- *gridServiceReference*: indica um ou mais GSR para a instância. Um valor do serviço de dados deve ser a representação WSDL do GSR;
- *terminationTime*: indica quando a instância irá ser encerrada.

Alguns exemplos de operações nos *portTypes* dos serviços de Grid são:

- *findServiceData*: realiza a busca na instância e retorna a descrição do serviço;
- *setServiceData*: modifica um SDE, desde que permitido, e a informação da respectiva descrição. Esta modificação afetará a instância do serviço;
- *requestTerminationAfter*: faz com que seja alterada a hora de encerramento do serviço. Indica o tempo mais recente para encerramento do serviço;
- *requestTerminationBefore*: também altera a hora de encerramento do serviço, mas indica o tempo mais distante para que isso ocorra;
- *destroy*: encerra o serviço.

Há ainda um serviço para notificação (*Notification portType)* , cujo objetivo é distribuir mensagens entre as instâncias de serviços da Grid. O *Factory portType* serve para que uma instância do serviço solicite a criação de um nova instância. Ele inicia o processo de vinculação com o GSH. Por fim, há o *ServiceGroup* que mantém as informações sobre o grupo de serviços da Grid. Este grupo é um dos elementos responsáveis pela identificação da federalização de serviços ou simplesmente pela identificação dos serviços existentes em um diretório ou índice. Três *portTypes* implementam a interface: *ServiceGroup*, *ServiceGroupEntry* e *ServiceGroupRegistration*.

2.6 Desenvolvimento para uma Grid Computacional

Uma Grid Computacional é uma grande Organização Virtual (OV) porque reúne todos os recursos espalhados pela rede como se fosse um único. Isso faz com que seja possível aumentar o desempenho das aplicações e a capacidade de armazenamento. Sistemas que necessitam de um alto poder de processamento e armazenamento são os que mais se beneficiam do uso da Grid. Quanto mais o processamento puder ser paralelizado ou distribuído, melhor será o desempenho da aplicação na Grid.

A evolução que tem ocorrido no desenvolvimento de sistemas, em especial nas aplicações distribuídas baseadas em serviços, faz com que a utilização das Grids Computacionais seja viável. Com aplicações baseadas em serviços não é necessário depender de um único tipo de recurso, como sistema operacional, rede ou armazenamento. Quando um serviço não está disponível, outro assume automaticamente seu lugar. O processamento não é interrompido com a perda de um recurso. Outra grande vantagem na utilização de serviços está no fato de este ser uma entidade que provê compatibilidade através do uso de mensagens, com facilidade para descrever, localizar e utilizar outros serviços [Watson et al, 2002]. Os serviços baseados na *Web* têm interface definida em padrões abertos, como XML. Isso faz com que diferentes tipos de sistemas possam ser integrados [Chede, 2005].

Neste momento, o desenvolvimento de aplicações baseado em *Web Services* é cada vez mais utilizado. *Web Services* definem uma técnica para descrever o acesso aos componentes de sistemas (serviços) com métodos para descoberta e protocolos de acesso. Ao serem utilizados em uma Grid Computacional, as interfaces e modelos utilizados pelos *Web Services* necessitam suporte ao gerenciamento distribuído, incluindo serviços temporários e seus respectivos ciclos de vida e estágio. Tanto a Grid quanto os *Web Services* não interferem no modelo ou linguagem de programação e nem no sistema operacional utilizado [Gil, 2004].

O modelo de Grid Computacional é uma tendência quando se fala em ambiente distribuído e atenderá melhor às necessidades dos usuários. Contudo, há uma série de etapas que se devem ultrapassar para que isso se torne realidade. Padrões devem ser estabelecidos para que as aplicações possam trocar informações e os servidores tenham condições de compreender as requisições e enviar os resultados.

As Grids evoluíram rapidamente de uma infra-estrutura altamente dependente de configurações específicas para um ambiente virtual único e dinâmico. A utilização dos *Web Services* trouxe muitos ganhos para os desenvolvedores por adotar padrões, mas, por outro

lado, tem gerado uma competição de *como* implementar a arquitetura e quais padrões seguir [Baker et al., 2005].

2.7 Descrição de Serviços em Grid Computacional

A característica básica do acesso e restrições de acesso aos recursos está em definir quais são as condições que devem ser satisfeitas. A partir do acesso é necessário estabelecer um contrato de serviço entre o fornecedor e o consumidor para garantir a ambos as condições de fornecimento e utilização do recurso. Os protocolos são fundamentais nesse acordo para definir forma e comportamento esperados [Pernas e Dantas, 2004].

2.7.1 Ontologias

Define-se como o desenvolvimento de um vocabulário que contém conceitos relativos ao domínio de aplicação. É uma especificação formal (interpretável por máquina) e explícita de uma conceituação (modelo abstrato de um fenômeno do mundo) compartilhada (captada por um grupo).

Os componentes de uma ontologia são: classes (expressa qualquer coisa sobre a qual alguma coisa é dita); relações (interação entre a classe e o domínio); funções (um único elemento se relaciona com os elementos anteriores) e axiomas (sentenças verdadeiras) [Pernas e Dantas, 2004].

Ao se utilizarem as ontologias é possível proporcionar um entendimento amplo sobre as características e propriedades de suas classes e relacionamentos.

2.7.3 Ontologias para descrição de recursos do Grid

Utilizam-se axiomas para determinar o conhecimento sobre o recurso e seu funcionamento. Além disso, os recursos são descritos por: metadados (guardam informações sobre os recursos computacionais) e visões semânticas (retorna informações sobre o funcionamento no momento da pesquisa) [Pernas e Dantas, 2004]. Desta forma, é natural que todos os recursos do Grid devem estar presentes nos metadados e as visões semânticas, sendo dinâmicas, devem refletir a situação do recurso.

2.7.4 Arquitetura de Grids a partir de Ontologias

A ontologia proposta atua nos serviços de diretórios do Grid. As consultas realizadas para localização dos recursos são feitas com base no vocabulário definido na ontologia. A ontologia se localiza em uma camada à parte do Grid uma vez que as solicitações atuam

inicialmente nos metadados que consultam as visões semânticas para retornar informações sobre os recursos disponíveis [Pernas e Dantas, 2004].

3 Banco de Dados em Grid Computacional

As principais aplicações em uso na Grid são baseadas em arquivo de dados porque esta tem sido mais utilizada no meio acadêmico e científico [Watson, 2003]. Neste meio praticamente não há utilização de sistemas gerenciadores de banco de dados (SGBD).

Por outro lado, as aplicações comerciais utilizam os SGBD em larga escala devido à facilidade de manipulação e controle, especialmente em dados transacionais, que a linguagem SQL (*Structured Query Language*) oferece.

Um requisito básico para que a Grid Computacional possa extrapolar o uso no meio acadêmico e ser também utilizado por empresas comerciais é a necessidade de se integrar aos principais sistemas gerenciadores de bancos de dados. Até o momento, poucos trabalhos têm explorado essas formas de integração [Watson, 2003], mas os que estudam a adoção de SGBD nas aplicações em Grid, demonstram que há ganhos significativos mesmo em aplicações estritamente científicas [Santisteban et al., 2004]. A constatação que se faz dessa experiência é que os SGBDs possuem melhores meios de filtro de dados, indexação e pode utilizar paralelismo nas buscas. Ao concentrar as atividades dos pesquisadores na ciência e ao se deixar que o SGBD cuide da complexidade de acesso e gerenciamento de dados, o ganho de produtividade é significativamente aumentado.

No meio acadêmico uma Grid Computacional trabalha com um volume muito grande de dados. Fala-se que, em um futuro próximo, será alcançado o patamar de *petabytes* de informação [Chede, 2005]. Ao se optar pela utilização de SGBD será possível manter os dados com maior segurança, obter melhor desempenho e encapsular processos complexos necessários para manipulação de arquivos de dados.

Os SGBDs possuem recursos muito mais potentes do que os sistemas baseados em arquivos de dados. Além do simples armazenamento de dados, os SGBDs garantem maior segurança e agilidade ao processo. Por este motivo é necessário criar condições de integração à Grid e esta integração não deve diminuir os recursos utilizados pelas aplicações comerciais. De outra forma, a adoção da Grid pela comunidade não acadêmica estará prejudicada.

Parece consenso entre os principais estudiosos do assunto que não se deve criar um novo banco de dados para ser utilizado na Grid Computacional [Watson, 2003]. Isso demandaria tempo e esforços que já foram feitos. O que se tem feito é buscar criar algum mecanismo semelhante a um protocolo e que permitisse realizar os principais serviços dos bancos de dados, tirando o máximo proveito dos serviços da Grid Computacional. A este

mecanismo que, conforme analisado anteriormente, deve ser baseado em serviços, dá-se o nome de Banco de Dados Federalizado ou Banco de Dados Virtual (figura 3.1).

Tanto no meio acadêmico e científico como no ambiente empresarial, é necessário trabalhar com dados estruturados, semi-estruturados e não estruturados. Este Banco de Dados Virtual deve prever a utilização desses tipos de dados. O mecanismo sugerido guarda alguma semelhança com os padrões ODBC e JDBC, mas há interesse em não limitar ou nivelar por baixo os bancos de dados, como acontece com estas APIs (*Application Program Interface*). Como a arquitetura proposta é baseada em serviços, os produtos deverão indicar o que se pode ou não realizar [Watson, 2003].

Imagina-se que, com a adoção crescente das Grids Computacionais por parte das empresas, haverá interesse por parte das empresas fornecedoras de SGBDs de portar seus produtos para o Banco de Dados Virtual [Watson, 2003].

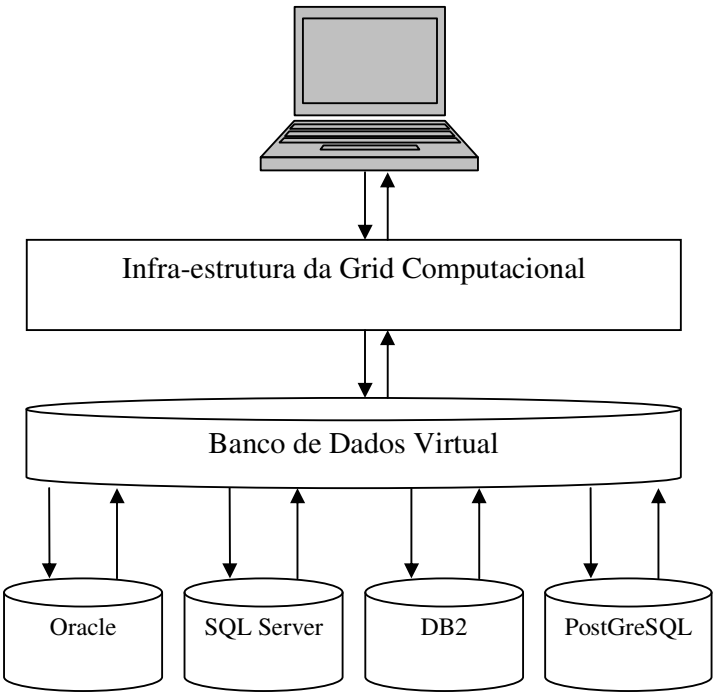


Figura 3.1 – Nesta estrutura, o Banco de Dados Virtual se encarrega de armazenar e distribuir as informações entre os bancos de dados que estiverem na Grid Computacional. As aplicações devem se comunicar com o Banco de Dados Virtual e ele resolve as demais questões. A comunicação do banco de dados virtual pode se dar com dados estruturados, semi-estruturados ou não estruturados.

3.1 Data Grids

Utiliza-se este termo para indicar qualquer tipo de acesso seguro a dados armazenados em uma Grid. É assim que os dados podem ser distribuídos geograficamente e com

mecanismos de acesso e recuperação de maneira transparente. Cada nó da Grid é responsável pela disponibilização e controle de seus recursos. Podem utilizar diferentes tecnologias e meios de armazenamento [Chede, 2005].

O principal objetivo dos Data Grids é permitir que haja acesso controlado aos dados distribuídos. [Chede, 2005] cita as tentativas de realizar este controle através de sistemas de arquivos, como NFS (*Network File System*) e protocolos de transferência, como FTP (*File Transfer Protocol*). Mesmo utilizando mais seguros como as VPNs (*Virtual Private Network*), este mecanismo esbarra em diversas limitações que precisam ser transpostas para utilização em Grid.

Chede [Chede, 2005] cita a utilização do AFS (*Andrew File System*) que se utiliza do Kerberos para criar um mecanismo que permite maior controle e distribuição de arquivos. Cita ainda o Avaki Data Grid que se propõe a implementar uma camada de serviços para acesso a dados distribuídos. Todas estas tentativas, contudo, esbarram na utilização de arquivos de dados. Isso pode resolver uma parte do problema, mas efetivamente não resolve o ponto central que é a utilização de banco de dados comerciais em ambiente de Grid.

3.2 SGBD comerciais

A plataforma básica para disponibilizar um SGBD em uma Grid Computacional é formada por [Oracle, 2005 e IBM, 2005]:

- **Cluster**: permite que mais de um gerenciador de banco de dados, do mesmo fornecedor, seja instalado em diferentes servidores. Estes bancos de dados devem ser compartilhados e administrados como se fossem um único. Pode ou não haver mecanismos de replicação dos dados para garantir a disponibilidade da informação e a paralelização das transações para agilizar a recuperação de dados.
- **Auto-gerenciamento e auto-ajuste**: como não é possível determinar a carga de trabalho de cada computador *clusterizado*, o SGBD deve ser capaz de gerenciar e ajustar-se à demanda de dados e processamento. Sabe-se que cada gerenciador possui características próprias que permitem melhorar o próprio desempenho. Normalmente este serviço é realizado por um profissional, no caso o Administrador de Banco de Dados (DBA). Como em uma Grid a demanda não é contínua e nem previsível, não se pode depender exclusivamente de um profissional para realizar esta tarefa. Deve-se criar um repositório que armazene os dados históricos da utilização do banco e, com base no volume de dados e utilização, são feitas alterações nos parâmetros do SGBD para otimizar o desempenho (figura 3.2).

- **Armazenamento auto-gerenciado**: é um mecanismo que permite o acesso a dados que não estejam fisicamente no mesmo servidor onde está instalado o SGBD, ou, caso esteja, não dependa do sistema operacional para gerenciá-lo. Com este mecanismo é possível distribuir os dados em diferentes servidores, sendo, contudo, gerenciado pelo próprio SGBD e não pelo sistema operacional. Isso faz com que a carga e armazenamento de dados possam estar distribuídos através de diversos computadores. Este mecanismo permite que o próprio SGBD replique os dados que julgar necessário para diminuir a interferência de um profissional. Isso faz com que a segurança da informação seja aumentada, mais ou menos como acontece com RAID (figura 3.2).
- **Federalização**: com o banco de dados federalizado é permitido o acesso a qualquer dado distribuído pela Grid computacional, seja ele estruturado ou não. Há possibilidade de acesso a diferentes bancos de dados e as consultas são otimizadas baseadas em custo (CBO). Esta estrutura garante a possibilidade de distribuir o armazenamento e processamento sem se importar onde, qual dialeto SQL, como está armazenado ou qual protocolo se deve utilizar. O funcionamento da estrutura do banco de dados federalizado é: o cliente utiliza JDBC, ODBC ou Web Service para interagir com o banco de dados. Este, por sua vez, comunica-se com a fonte de dados (pode ser interna ou externa). Quando o dado for processado, o banco de dados federalizado retorna a informação a quem solicitou. Há uma preocupação em armazenar metadados das fontes não relacionais. Com isso, novos acessos são otimizados com uso de heurísticas específicas (figura 3.3).

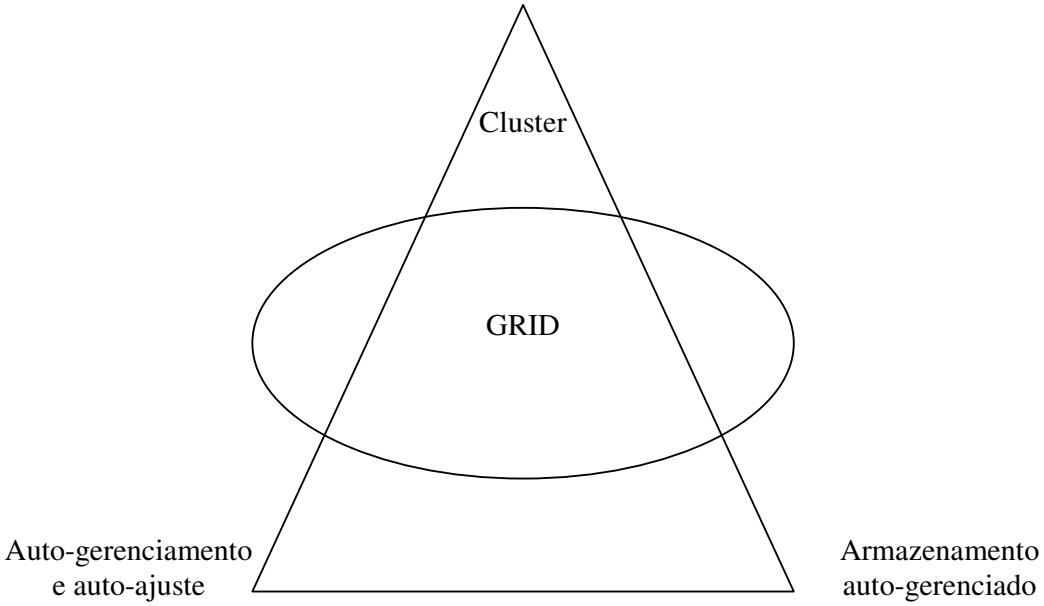


Figura 3.2 – Arquitetura da Oracle: o banco de dados se encaixa em uma Grid Computacional através da clusterização de servidores, do armazenamento auto-gerenciado e dos mecanismos de auto-gerenciamento e auto-ajuste. Para se comunicar com outras fontes de dados, utiliza o Oracle Streams.

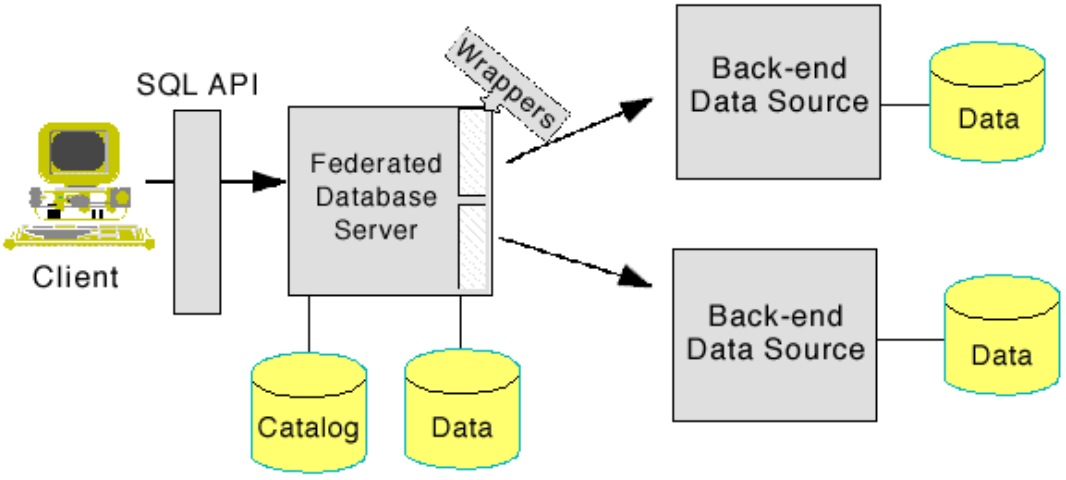


Figura 3.3 – Arquitetura da IBM: as aplicações comunicam-se com o banco de dados federalizado utilizando interfaces JDBC, ODBC ou Web Services. O banco de dados federalizado se comunica com as demais fontes de dados [IBM, 2006].

3.3 SGBD Distribuídos e Virtuais

A Grid é um ambiente para processamento distribuído, conforme amplamente discutido anteriormente. A arquitetura de serviços foi criada para atuar neste ambiente. Bancos de dados virtuais, contudo, não possuem técnicas específicas para integrar a arquitetura de serviços em um ambiente de Grid. Algumas considerações [Watson, 2002]:

3.3.1 Banco de Dados e Distribuição

Em um banco de dados distribuído, as informações estão dispersas em um determinado número de sites. O controle, contudo, é centralizado em algum destes sites. Em um banco de dados federalizado (ou virtual), cada qual, apesar de estar igualmente disperso, possui autonomia total sobre suas operações. As fontes de informações são escondidas da complexidade da implementação.

Um banco de dados distribuído ou federalizado pode ser visto como um serviço que permite a solicitação de informações armazenadas. Desenvolver serviços para bancos de dados virtuais passa a ser importante para uso nas configurações da Grid. O desenvolvimento de serviços padronizados para acesso aos atuais bancos de dados trará um grande benefício. A alocação de recursos no *middleware* poderá se utilizar destes serviços para avaliar a distribuição de buscas.

3.3.2 Serviço de Busca Distribuída

Uma aplicação da Grid pode se conectar a diversos bancos de dados ao mesmo tempo. Cada busca poderá extrair dados de um único banco de dados. As solicitações das aplicações precisam ser subdivididas para realizar a busca em mais de um banco de dados. O que uma busca distribuída deve fazer é permitir que buscas individuais possam ser realizadas em vários bancos de dados. O sistema deve ser responsável pela otimização das buscas.

Uma arquitetura para o desenvolvimento e aplicação do serviço de busca distribuído será útil porque pode ser disponibilizado e acessado como qualquer serviço de banco de dados e pode utilizar as facilidades de descoberta, integração e busca de serviços de banco de dados.

3.3.3 Replicação Seletiva

As Grids possuem mecanismos de replicação dos dados, da mesma forma como os bancos de dados também possuem. Regras podem ser impostas a visões materializadas das tabelas replicadas. Um serviço de notificação pode ser utilizado para indicar que houve modificações na visão e esta modificação pode ser propagada automaticamente. Os serviços

de replicação de banco de dados podem ser indicados na interface do serviço. O conhecimento acerca da replicação dos dados pode ser útil para o serviço de programação de recursos da Grid.

Outros serviços relacionados à replicação podem ser criados para integrar diferentes fornecedores de bancos de dados. Desta forma, o serviço de busca, entrega e armazenamento de dados poderá ser otimizado.

3.4 Requisitos de um Banco de Dados em Grid

Os sistemas gerenciadores de banco de dados (SGBD) alcançaram um nível de complexidade bastante alto. Isso faz com que a Grid Computacional tenha que fornecer mecanismos complexos para efetuar buscas e controlar transações. Cada implementação de banco de dados possui uma estrutura própria de gerenciamento e segurança que deve se integrar às aplicações da Grid.

Por outro lado, pelo conceito de Grid, não é admissível que se utilize uma única implementação de banco de dados, como Oracle, SQL Server ou DB2. É necessário integrar diferentes produtos e fazê-los trabalhar como se fossem um único. Dessa forma, deve ser possível utilizar em uma mesma Grid, além dos servidores citados, servidores livres como PostGreSQL e MySQL. Por outro lado, é necessário que os SGBD não se limitem a um único *middleware*.

As principais necessidades de integração entre uma Grid Computacional e um banco de dados são [Watson, 2003]:

- **Segurança**: a Grid possui uma estrutura própria de identificação dos usuários, como o *Grid Security Infrastructure*. Este mecanismo une os usuários em uma Organização Virtual, porém autenticada e com critérios rígidos de segurança. Os gerenciadores de bancos de dados devem integrar a segurança da Grid com seus mecanismos próprios e garantir que os usuários tenham acesso às informações necessárias para suas aplicações. Em um ambiente empresarial, a utilização do banco de dados é previsível uma vez que os usuários dos sistemas são conhecidos (funcionários da empresa, por exemplo) e há controle na utilização e acesso aos dados. Em uma Grid ocorre exatamente o contrário: não se sabe quem utilizará nem quando o recurso será utilizado.
- **Programação**: cada banco de dados possui sua própria linguagem de programação. Tem sido tentada a padronização das linguagens de programação de banco de dados desde a versão SQL de 1999. Os principais produtos do mercado já possuem

linguagens próprias de programação e poderão se adaptar ao padrão, mas dificilmente irão aderir completamente a este [Oliveira, 2002]. Contudo, é necessário que haja uma forma única de comunicação e isso inclui a programação dos bancos de dados. A Grid, como foi visto, utiliza prioritariamente serviços baseados em Web (*Web Services*) para descoberta, execução e encerramento. Os fornecedores de banco de dados deverão indicar quais serviços poderão executar e em qual padrão. Com isso as aplicações poderão requisitar tais serviços nos bancos de dados.

- **Monitoramento de Desempenho**: deve haver uma forma única de avaliar como e quando um serviço é entregue e em que prazo será devolvido. Se isso não for possível, a Grid Computacional não terá condições de avaliar e estimar o prazo de entrega do serviço àquele que solicitou o processamento. Atualmente não há um padrão entre os SGBD que garantam a uniformidade dessa informação. Os bancos habilitados à Grid deverão possuir mecanismos de auto-ajuste e autocontrole que permitam dimensionar a carga de serviço atual, prevista e o tempo esperado para execução do serviço que foi submetido.
- **Programação de Uso**: tão importante quanto o monitoramento, a programação de uso para em ambiente de Grid é necessário. Nem sempre a necessidade de processamento se dá imediatamente. Deve haver forma adequada de programar o uso futuro. Deve-se levar em consideração a utilização média e prever a disponibilidade futura do recurso.
- **Custo**: potencialmente os serviços disponibilizados em uma Grid serão cobrados no futuro. Com isso é necessário estabelecer uma forma única para calcular o quanto de processamento e armazenamento está sendo utilizado.
- **Escalabilidade**: uma Grid Computacional demanda volumes de dados muito grandes, chegando a 1 Terabyte por hora em algumas aplicações atuais. Devido a este alto volume, é desejável que o banco de dados possa paralelizar o processamento. Na área de *metadados*, o SGBD indicaria à Grid como localizar e permitir acesso aos dados. Os sistemas baseados em Grid são semelhantes aos de *data warehouse* na forma, mas potencialmente podem envolver muito mais usuários e com a requisição de um volume de dados muito maior.
- **Disponibilização de Metadados**: atualmente localizar dados na Grid Computacional é relativamente simples, visto que basta mapear logicamente onde os dados estão localizados fisicamente. Com diversos bancos de dados isso tende a ser mais complicado. Por este motivo, é necessário indicar em diversos *metadados* espalhados

pela Grid onde está a informação necessária para depois acessar a informação. Some-se a esta complexidade o processamento de transações entre os diversos bancos de dados e a necessidade de replicar dados, para resolver eventuais problemas de queda de servidores.

Por outro lado, os atuais SGBD disponibilizam para as aplicações típicas de uma Grid Computacional diversas possibilidades que atualmente não existem. Entre elas pode-se destacar: o gerenciamento dos dados que deixam de ser responsabilidade da aplicação e passam para o gerenciador de banco de dados; o gerenciamento dos recursos do servidor, uma vez que a maior parte dos gerenciadores já tem mecanismos de integração com o equipamento e sistema operacional; cópias de segurança dos dados armazenados, visto que possuem diversos mecanismos para agilizar e controlar este processo; replicação de dados controlada pelo mesmo servidor e, finalmente, o controle de concorrência e transações dos dados. Além destes, é possível adaptar-se informações que são utilizadas pelos SGBD e disponibilizá-las para os *middlewares*. Isso inclui o inventário dos recursos, manutenção de regras específicas de controle e acesso aos dados com o objetivo de agilizar o processo de consulta e criação de um repositório do projeto para facilitar a busca e intercâmbio de informações entre pesquisadores.

3.4.1 Desafios

Os grandes desafios de aplicações robustas para atuar em uma Grid Computacional e que podem ser extrapolados para a utilização de SGBD são classificados por [Gil, 2004]:

- **Captura do conhecimento**: identificação dos recursos disponíveis: SGBDs, dados, relacionamentos entre os dados e entre os SGBDs e a capacidade de processamento e armazenamento de cada servidor.
- **Utilidade**: deve haver plano de contenção para falhas de equipamentos, sistemas operacionais e SGBD. Da mesma forma é necessário estabelecer uma política para utilização dos recursos da Grid. Deve-se prever formas de diminuir o trabalho dos usuários das aplicações, facilitando a programação (s*chedule*) de utilização dos recursos.
- **Robustez**: deve-se prever o maior nível de utilização dos servidores para cada serviço requisitado. Os servidores devem, preferencialmente, estar disponíveis 24 horas por dia e 7 dias por semana. Isso envolve sistema operacional, software de rede e SGBD.

- **Acesso**: a quantidade de usuários em aplicações típicas da Grid Computacional é muito grande. Estes usuários estão dispersos em Organizações Virtuais. O controle de acesso e permissões é fundamental para o sucesso de uma aplicação em Grid.
- **Escala**: quando se fala em Grid Computacional, as necessidades são muito grandes. Processamento, armazenamento e a quantidade de usuários são altíssimos. Os dados, por segurança e velocidade, precisam estar dispersos e muitas vezes replicados para agilizar o processamento e a busca. Para atender estes requisitos, é necessário definir um sistema de fluxo de trabalho (*workflow*) adequado para determinar o uso dos recursos e o caminho a seguir em caso de falhas.

3.5 Serviços de Acesso a Banco de Dados

Uma vez que set torna necessário utilizar banco de dados em ambiente de Grid computacional, deve-se estabelecer claramente a forma que a integração entre o *middleware* e os bancos de dados serão realizados.

Segundo Watson [Watson, 2002], ao se utilizar serviços para esta finalidade determina-se *o que* será realizado e não *como* fazer. Assim, não importa onde ou por quem será realizada a operação, mas sim o retorno da solicitação. Este é o objetivo de criar os serviços virtuais, que permitirão realizar buscas, atualizações e coordenação de transações entre diversos bancos de dados. Para o usuário, contudo, deve haver total transparência de como este processo se desenvolve.

Watson [Watson, 2002] propõe um desenvolvimento em estágios de um conjunto de serviços para bancos de dados em Grid. Desta forma pode-se adotar padrões de integração entre bancos de dados. Assim, quanto maior o número de pessoas envolvidas, melhor será o resultado.

Dentro do escopo da proposta, é necessário conceituar um serviço de banco de dados.

Entende-se serviço como uma entidade de rede que provê condições para troca de mensagens. Padrões de serviços Web (*Web Services*) especificam facilidades para descrever, descobrir e utilizar serviços. Um destes padrões é o *Web Services Description Language* (WSDL) que utiliza o XML para descrever tais serviços. São utilizadas especificações WSDL para tipos (definição do tipo de dado para descrever a informação trocada), mensagens (definição dos dados enviados e recebidos do serviço) e tipo porta (operação suportada pelo serviço. Para cada operação, inclui nome, entrada, saída e mensagens de falha). A definição do WSDL fornece informações do que o serviço poder realizar e informações de onde e como as mensagens serão enviadas.

A utilização do termo *interface* servirá para se referir a um conjunto de tipos, mensagens e portas que descrevem uma característica específica do serviço.

3.5.1 Serviço de Banco de Dados

Entende-se um Serviço de Banco de Dados como qualquer serviço que suporte uma interface de banco de dados. Pode ser persistente ou implementada através de um gerenciador de banco de dados. Com visto, os serviços não indicam "como" a informação será armazenada e também não especificam o formato ou modelo de armazenamento (relacional, objeto, XML, etc.).

As operações básicas da interface de serviço do banco de dados são:

- IN: parâmetro de entrada
- OUT: parâmetro de saída
- OPTIN e OPTOUT: parâmetros opcionais de entrada e saída

As operações devem ser atômicas, isto é, ou são executadas completamente ou não realizam trabalho algum. Como as transações geralmente são muito grandes, há necessidade de controle para interrupções e gerenciamento de falhas.

A estrutura para busca, atualização e carga de dados se dá em três fases:

- **Preparação e validação**: a operação é verificada para garantir a consistência e o modelo de dados do banco de dados.
- **Aplicação**: tempo em que as atualizações ou busca são efetuadas e os resultados construídos.
- **Entrega de Resultado**: quando os resultados são disponibilizados para o requisitante.

O primeiro passo normalmente leva o mesmo tempo que a soma dos outros dois. Em caso de erro, o parâmetro de saída *resultHandle* indica de maneira assíncrona quando houve problemas durante o processamento. Os parâmetros para busca, atualização, carga e modificação sempre vêm acompanhados da notação padrão, que pode se utilizar da linguagem SQL, e o comando no formato especificado. Há um parâmetro para indicar o tempo de expiração do comando. Quando não for especificado, este deve ter um conteúdo padrão. Um último parâmetro refere-se ao controle de transações. É opcional, o que indica que, em caso de haver necessidade de controlar a transação entre diversas operações, ele deve ser informado.

Outro ponto importante a se considerar são os mecanismos de entrega [Watson, 2002]. O sistema de entrega deve existir para transporte de um grande volume de dados. Deve ser complementar a protocolos e serviços existentes no *middleware*, como GridFTP. Uma

interface possível para os sistemas de entrega deve conter três operações: origem dos dados, canal de comunicação e entrega. No canal de comunicação é indicado: mecanismo de entrega, destino e opção para expiração do comando. Um sistema mais completo deve conter mecanismos de encriptação, mapeamento para o formato de condução dos dados antes e depois da entrega e indicador de progresso do serviço.

O controle de transações é essencial para SGBD [Watson, 2002]. Há outros mecanismos de controle de transações através de padrões de serviços, como J2EE e OMG. A definição de uma interface básica envolve o início, gravação ou abandono das transações. O retorno do identificador único da transação deverá ser utilizado pelos comandos de banco de dados, conforme visto anteriormente.

Para controlar transações que envolvam mais de um banco de dados, é necessário que mais um serviço seja acrescentado e outro modificado. O serviço modificado é o do início da transação, visto que há necessidade de indicar um tempo de expiração para que não haja alocação desnecessária do recurso. O novo serviço será uma preparação para gravação. Este serviço é útil para utilizar o protocolo de gravação em duas fases (*two-fase commit*).

Uma Grid computacional necessita de alguns outros controles que normalmente não são necessários para os bancos de dados, mesmo quando trabalham de maneira distribuída. São eles:

- Colaboração multi-site com troca de mensagens assíncronas. Embora haja em bancos de dados distribuídos, pela característica da Grid, pode haver necessidade específica como parte da execução concorrente da aplicação.
- As operações da Grid são compostas por processos que podem ser de diversas regiões de controle. Normalmente em banco de dados, os processos de transação são relativos a um único recurso. Há, contudo, SGBD que realizam o mesmo controle quando trabalham distribuídos.
- As transações de banco de dados normalmente são de grande volume e de pequena duração. Em operações na Grid, as transações podem ser de longa duração.

Para descoberta de serviço na Grid, é necessário estabelecer informações básicas que facilitem este processo:

- **Descrição de conteúdo**: deve conter as informações relacionadas aos objetos que o banco de dados mantém. Deve indicar desde quais são as tabelas, índices, relacionamentos e tipos de objetos armazenados, até as estatísticas destes objetos,

proprietário das informações e critérios de segurança de acesso. Desta forma será possível agilizar os processos de busca e transações.

- **Descrição da capacidade**: devem indicar quais são os recursos disponíveis para serem utilizados. Assim, a linguagem, transações e conexões são exemplos deste tipo de informação.

3.5.2 Padronização

Fica claro que há necessidade de estabelecer padrões entre os bancos de dados para que estes possam ser utilizados em uma Grid Computacional. A padronização passa pelo estabelecimento de protocolos de comunicação que contenham basicamente: acesso aos *metadados* dos SGBD para localização de dados e execução de comandos transacionais, controle de acesso aos dados, notificação de onde está e quanto tempo poderá durar uma determinada transação, programação de recursos, estabelecimento de serviços a serem oferecidos e definição de custo e segurança não apenas no acesso aos dados, mas também quanto à política de *backup* e recuperação de dados.

O escopo da proposta segue alguns parâmetros aceitáveis tanto do ponto de vista da Grid Computacional quanto dos SGBD [Watson, 2002]:

- **Independência de Grid**: não deve estar vinculada a um *middleware* específico. Atualmente as Grids evoluem rapidamente e vincular a uma única implementação pode ser caótico por utilizar um conjunto específico de funcionalidades e ferramentas que não estejam disponíveis para todas implementações.
- **Não criar um novo Banco de Dados**: estabelecer mecanismos de controle de transações, buscas e outros serviços que existem nos principais SGBD. Os bancos de dados mais completos, contudo, terão maior aderência à Grid e os demais vão precisar se adaptar às necessidades.
- **Independência de Banco de Dados**: não se deve seguir uma implementação única para modelo de dados, comandos transacionais e programação de banco de dados.

Conforme visto, a idéia básica é criar um banco de dados virtual que se integre aos SGBD atuais e que tenha serviços para oferecer ao ambiente computacional. Com isso, os SGBD poderiam informar o nível de integração e como resolveriam determinadas questões. Este banco de dados virtual tomaria a decisão de utilizar ou não um ou outro SGBD. Os principais serviços estariam relacionados com a distribuição dos dados, busca distribuída e replicação seletiva de dados [Watson, 2002].

A padronização passa necessariamente pela integração com os recursos disponíveis nos *middlewares*, como GridFTP e GSI abordados anteriormente. Pode-se utilizar outros padrões específicos de banco de dados, como a linguagem SQL e os padrões de comunicação de objetos definidos pela ODMG (*Object Database Management Group*).

Segundo [Watson, 2003], a utilização de APIs semelhantes ao JDBC, mas orientada a serviço podem oferecer uma maior integração aos *middlewares*. As características dos serviços de dados são:

- **Metadata**: o objetivo é ter um nome lógico e físico do banco de dados e uma definição dos serviços que podem ser oferecidos à Grid.
- **Busca**: padronizar uma forma de acesso aos dados que possa ser estendido para cada implementação de SGBD. Com isso cada fornecedor poderá, baseado no padrão, realizar internamente o serviço de busca.
- **Transação**: as transações podem ser executadas em diferentes implementações de banco de dados. Há necessidade de controlar as transações que foram distribuídas nos diversos SGBD. Atualmente os produtos comerciais já realizam este trabalho, mas é necessário compartilhar a informação no *middleware* para que se possa distribuir o processo. Essa distribuição passa necessariamente pela criação de réplicas dos dados armazenados.
- **Transferências de Lotes de Dados**: as buscas podem ser paralelizadas para garantir maior velocidade e disponibilidade da informação. É necessário que se criem mecanismos para unir as informações depois de processadas para que se dê um resultado único ao usuário.
- **Notificação**: deve ser possível através de mecanismos automáticos, como gatilhos, informar à Grid sobre modificações ocorridas na base de dados. Com isso, havendo registro do interesse nas informações daquela base de dados por parte dos clientes, estes poderão ser notificados da mudança.
- **Escalonamento dos recursos**: deve ser possível requisitar recursos para processamento das informações. Os recursos passam por computadores e banda de comunicação até o conjunto de dados que serão analisados. O gerenciamento desta programação e a alocação dos recursos devem ser realizados e planejados com eficácia através de um fluxo consistente de trabalho.

- **Custo**: o SGBD deve ser capaz de medir o custo da informação coletada, seja para efeito de dimensionamento de tempo e desempenho ou até mesmo para uma eventual cobrança pelo recurso utilizado.
- **Navegação**: deve haver uma interface que facilite a utilização e programação direta dos recursos.

3.6 OGSA, OGSA-DAIS e OGSA-DAI

A evolução para uma arquitetura orientada a serviço é possível devido à infra-estrutura de rede disponível e a criação de organizações virtuais torna-se possível através do modelo de distribuição de serviços [Alpdemir, 2003]. A abordagem baseada em serviços para acesso aos dados não possui um alto nível de acoplamento e possui pouca granularidade. Já a tecnologia de processamento de buscas possui um alto nível de coesão. Isso acaba por gerar um impacto em aplicações distribuídas. Por este motivo é necessário estabelecer mecanismos de controle que possibilitem a realização do serviço de acesso aos dados de maneira satisfatória.

O *Open Grid Services Architecture* (OGSA) é um conjunto de definições de serviços web. O OGSA fornece: descrição da capacidade da grid utilizando WSDL, coleção de interfaces padrão dos serviços da grid. Este documento faz uma proposta para uma coleção de serviços de bancos de dados no contexto da OGSA [Watson, 2002].

Um serviço de dados (*Grid Data Service*) da OGSA (*Open Grid Services Architecture*) é um serviço da Grid que implementa um ou mais interfaces para acesso ou gerenciamento de recursos distribuídos. Os serviços de dados são construídos sobre a OGSI (*Open Grid Services Infrastructure*) que estende os *Web Services* para incorporar mecanismos de gerenciamento e controle da Grid [Foster, 2003] e [Alpdemir, 2003].

Para estabelecer um limite claro entre o que é e o que não é um serviço de dados foram definidas quatro interfaces básicas de dados para se implementar em diferentes comportamentos. Estas interfaces representam portas específicas do WSDL. Qualquer serviço que implemente uma destas interfaces é considerado um *Web service* compatível com a OGSI. As quatro interfaces são [Foster, 2003]:

- ***DataDescription***: define um elemento do serviço de dados da OGSI que são fundamentais para virtualização de dados encapsuladas por um serviço de dados.
- ***DataAccess***: provê operações de acesso e modificação de conteúdo dos dados virtualizados encapsulados por um serviço de dados.
- ***DataFactory***: provê operações para criar um novo serviço de dados em uma nova virtualização de dados herdada de outra virtualização do serviço de dados.

- ***DataManegement***: operações para monitorar e gerenciar os serviços de dados da virtualização de dados, incluindo os recursos necessários para execução do serviço.

O grupo DAIS (*Data Access and Integration Services*) trabalha atualmente na definição dos serviços de bancos de dados. Para isso foram utilizadas as quatro interfaces de dados citadas anteriormente e que devem ser implementadas para atender aos comportamentos dos bancos de dados. Segundo [Alpdemir, 2003], o principal objetivo do OGSA-DAI (*Data Access and Integration*) é criar uma infra-estrutura para entrega de serviços de alta qualidade e que agreguem valor ao gerenciamento de dados da Grid, especificamente nos *Grid Database Services* (GDS). Os componentes servem tanto para acesso quanto para integração dos dados. Desta forma, os *Grid Services* (GS) são estendidos nos seguintes serviços e portas:

- ***Grid Data Service Registry*** (GDSR): serve para publicação dos GDS. Os serviços são registrados e são utilizados para descrever um serviço que é um subconjunto de um SDE. Deve ser capaz de indicar às instâncias dos serviços registrados eventuais mudanças ocorridas em seu domínio.
- ***Grid Data Service Factory*** (GDSF): cria instâncias GDS para finalidades específicas.
- ***Grid Data Service*** (GDS): aceita uma solicitação que instrui a instância GDS a interagir com o banco de dados para criar, recuperar, atualizar ou excluir dados.
- ***Grid Data Transport*** (GDT): realiza o transporte de dados entre instâncias do GDS.

3.6.1 Virtualização dos Dados

Um ponto importante nessa abordagem é a necessidade de virtualização dos dados. Um sistema distribuído pode conter dados armazenados em fontes de dados diferentes e dispersas, com sintaxes próprias de acesso, gerenciados por diversos mecanismos e disponibilizados através de vários protocolos e interfaces. Mesmo com as particularidades de cada fonte de dados, é possível, através da virtualização, criar interfaces orientadas a serviço para permitir o acesso dos clientes. Portanto, a virtualização dos dados é um termo aplicado para designar uma interface orientada a serviço que permite o acesso a uma ou mais fontes de dados [Foster, 2003]. Pode ser simples, quando disponibiliza uma interface do sistema de armazenamento ou complexa, quando transforma de uma para outra forma de armazenamento. Pode ainda ser um conjunto de dados de uma única fonte de dados ou de várias fontes federalizadas.

Cada serviço de dados possui um nome específico que permite a descoberta de seus respectivos atributos. Com isso é possível utilizar mecanismos próprios da Grid

Computacional para descobrir tais serviços. É possível utilizar mecanismos próprios da OGSI para determinar o tempo de vida do serviço. Em alguns casos o serviço pode ser criado e destruído, mesmo que o recurso continue existindo fisicamente. Em outros casos é possível que a própria existência dos dados dependa do serviço executado, tendo um início, meio e fim definido pela Grid Computacional.

Com isso as sessões de cada cliente são gerenciadas pela Grid e podem ser requisitadas ao mesmo tempo por diversos clientes de diferentes formas. Pode se realizar o processamento seqüencial de cada solicitação, prevendo mecanismos de controle de concorrência na própria interface.

Um recurso de dados é virtualizado por um serviço de dados que é encapsulado por uma implementação da interface e não é nem visível nem acessível por outros usuários do serviço. Naturalmente a fonte de dados pode ser acessada por outros mecanismos externos à OGSA, mas, nesse caso, não fazem parte da Grid. O gerenciador do recurso (por exemplo, o banco de dados) realiza a comunicação com os serviços de dados e possui características próprias que independem do modelo de serviços da Grid [Foster, 2003].

3.6.2 Representação

Qualquer serviço de dados tem um nome global único e um *Service Data Elements* (SDE) que permite a descoberta dos atributos do serviço. Um SDE descreve os principais parâmetros da virtualização e as operações realizadas pelos serviços da Grid. Estas operações permitem que os clientes verifiquem o acesso aos dados, utilizem operações adequadas, derivem novos dados dos antigos e gerenciem a virtualização dos dados.

Um determinado serviço de dado poderá implementar diversas interfaces da OGSI, mas deverá, conforme visto, utilizar pelo menos uma das quatro interfaces básicas. Foi estabelecido um *framework* para o serviço de dados definidos na OGSA que determina [Foster, 2003] e [Alpdemir, 2003]:

- **Estado do serviço de dados**: utiliza a SDE para permitir pesquisa e descoberta via mecanismos padronizados. Pode ser utilizado para descrever os metadados a respeito da virtualização de dados (*Grid Service Registry* – GSR).
- **Nomes Globais**: cada serviço de dados tem um nome único durante todo seu tempo de vida. Normalmente o nome global é utilizado para se obter as informações necessárias à execução das atividades da organização virtual e para se comunicar com os serviços de dados (*Grid Service Handle* – GSH).

- **Gerenciamento de ciclo de vida**: há diversos tipos de serviços e diversas formas de persistência de objetos. Em alguns serviços, os dados serão criados fora do *middleware* mas se utilizarão deste para mapear recursos persistentes. Outros poderão criados dinamicamente para um aplicativo específico e terão ciclo de vida finito (criado, utilizado e destruído) e gerenciado através da OGSI. Nota-se que o ciclo de vida dependerá da definição do serviço e de sua implementação.
- **Sessões como serviços temporários**: serviços de dados exploram os serviços temporários da Grid para gerenciar sessões de clientes. Um serviço de dados pode ser acessado por diversos clientes e deve lidar com solicitações concorrentes em diversas de formas. Isso inclui a serialização das solicitações e o controle de concorrência das transações.
- **Notificação**: *grid services* distribuídos podem notificar um ao outro de maneira assíncrona sempre que houver uma mudança significativa no próprio estado.

3.6.3 Implementação

O mapeamento da virtualização de dados para o recurso é determinado pela implementação do serviço. Um recurso irá virtualizar seus dados através de um serviço encapsulado e não será visível nem acessível para outros usuários a não ser pelas operações descritas no serviço. Naturalmente os dados poderão ser acessados por mecanismos externos à OGSA, como arquivos de Entrada e Saída ou JDBC. Porém o acesso externo está fora do escopo do *middleware* e não terá controle ou gerenciamento pela Grid.

O gerenciador do recurso mediará o acesso aos dados encapsulados pelo serviço para prover a virtualização da instância de dado na infra-estrutura da Grid. O gerenciador de dados não faz parte da arquitetura do modelo, mas pode ser útil para descrever os serviços de dados e como eles se relacionarão com os dados, conforme abordado anteriormente. Também poderá auxiliar na descrição dos serviços, dos dados disponíveis (metadados) e nas demais informações úteis para execução dos aplicativos dos usuários.

3.6.4 Serviços e Interfaces de Dados

Um serviço de dados implementa uma ou mais interfaces associada a um comportamento para manipulação dos dados virtualizados. Cada interface de dados estende uma interface *WS-Agreement* que, por sua vez, estende um serviço da OGSI. Cada interface de dado pode ser especializada por um tipo particular da virtualização de dados. Um serviço

de dados pode implementar várias combinações destas interfaces. As interfaces especificadas são [Foster, 2003]:

3.6.5 DataDescription

São utilizadas para informar clientes sobre os detalhes dos serviços da virtualização de dados. Desta forma, os clientes podem realizar requisições para as interfaces de *DataAccess*, *DataFactory* e *DataManagement* suportadas pelo serviço. Esta interface vai definir SDEs comuns entre as virtualizações de dados. As descrições da interface podem ser genéricas (tabela, coluna, tipo de dado, etc.) ou específicas do domínio (modelo climático, análise financeira, etc.).

3.6.5.1 DataAccess

Esta interface provê operações para acesso e modificação do conteúdo do serviço de dados virtualizado. Um conjunto de dados são dados armazenados em algum computador e disponíveis através de algum mecanismo ou padrão, como XML. Esta terminologia (conjunto de dados) refere-se a uma sintaxe e não ao serviço. Estes conjuntos aparecem como parâmetros de entrada ou saída para a interface *DataAccess*. Alguns exemplos desta especificação:

- ***SQLAccess***: buscas e atualizações SQL realizadas em virtualizações de dados relacionais. Esta interface pode ser *stateless* que indica que as buscas devem retornar um conjunto de dados completo e as atualizações serão realizadas com o conjunto de dados fornecido. Para criar um conjunto de resultados incrementais, pode-se utilizar uma interface baseada em cursor (*stateful*). A interface *SQLFactory* poderia ser utilizada para criar um novo serviço de dados que contém o resultado virtualizado e que implemente interfaces *RowSetDescription* e *CursorRowSetAccess*. A seguir são listados exemplos desta interface. Acredita-se que futuramente outros serão necessários e complementarão esta lista.
- ***CursorRowSetAccess***: acesso de um cursor para uma linha do dado virtualizado. É uma interface *stateful* o que significa que uma operação pode afetar o comportamento de operações futuras. Com isso uma operação que recupere *N* linhas atualizará o cursor no conjunto de linhas e, logo, as operações seguintes não terão o mesmo resultado.
- ***XMLCollectionAccess***: acesso a uma coleção de dados XML virtualizados através do *XPath*, *XQuery* e *XUpdate*. Esta é uma interface *stateless* que pode ser combinada com

a interface *XMLCollectionFactory* para criar um serviço de dados *stateful* para recuperação incremental de dados.

- ***StreamAccess***: operações de leitura e gravações incrementais em uma virtualização de dados. É uma interface *stateful* que será criada em um serviço criado pela interface *DataFactory*.
- ***FileAccess***: é uma extensão do *StreamAccess* que permite ler e gravar em dados virtualizados no formato Posix e no formato texto. É uma interface *stateful* que será criada em um serviço criado pela interface *DataFactory*.
- ***BlockAccess***: operações de leitura e escrita em blocos de dados (posição e tamanho) em uma virtualização de dados. É uma interface *stateless*.
- ***TransferSourceAccess***, ***TransferSinkAccess***: pontos finais para multi-protocolos e transferências entre dois serviços de dados. É uma interface *stateful* para configurar e gerenciar a criação de um conjunto de dados que seja visto como uma virtualização de dados.

3.6.5.2 DataFactory

Esta interface suporta a requisição para criar um novo serviço de dados de onde a virtualização de dados é derivada. O serviço que implementa a *DataFactory* é responsável pela derivação da virtualização dos dados do serviço original. A solicitação para a *DataFactory* resulta na criação do serviço de dado e retorna o *Grid Service Handle* (GSH). Este GSH pode ser utilizada pelos solicitantes para direcionar buscas a qualquer interface implementada pelo serviço. Cada instância tem seu próprio ciclo de vida, serviço de dados e estado. É importante frisar que esta interface não contém os dados físicos.

- **Utilização**: deve ser utilizado para criar um nome para a virtualização de dados (cada dado virtualizado tem um nome único definido através da GSH, mas não possui o dado em si), criar uma sessão para um cliente (algumas formas de acesso necessitam de sessões *stateful* e não são criadas por conveniência, mas para agilizar as operações) e para criar uma virtualização de dados vazia (necessária quando se quer adicionar elementos gerados pelo processo).
- **Utilização do *AgreementProvider***: a interface *DataFactory* define termos do acordo que são importantes para todos os serviços de dados. Cada extensão define termos adicionais que são importantes para um serviço específico que é criado através de uma extensão do *DataFactory*.

- **Extensão**: os exemplos de extensão para o *DataFactory* são os mesmos apresentados no *DataAccess*. São eles: *FileSelectionFactory* (criação de outro serviço de sistema de arquivo para um subconjunto de arquivos no serviço principal); *SQLFactory* (novo serviço de dados virtualizado que é um subconjunto de uma visão do banco de dados); *XMLCollectionFactory* (semelhante ao *SQLFactory*, mas para uso com XML); *TransferFactory* (utilizado para transferência de dados com a interface *TransferSourceAccess* e *TransferSinkAccess*) e *CollectionSelectionFactory* (cria um novo serviço de dados com os dados virtualizados de um dos recursos). Outros deverão ser definidos futuramente.
- **Federalização**: cria um virtualização de dados relacionais que integra os dados de diversos serviços de dados relacionais. Um *DataFactory* deve ser utilizado para criar o novo serviço federalizado com uma virtualização relacional vazia. A seguir, uma combinação de operações de *DataAccess* e *DataManagement* será utilizada para acrescentar dados na nova fonte virtualizada. O *DataAccess* criará cópias dos dados no serviço federalizado ao passo que o *DataManagement* realizará o mapeamento de cada pedaço da virtualização. As operações de *DataAcess* utilizarão este mapeamento para realizar as operações necessárias. Por definição, uma Organização Virtual deve conter um serviço de dados específico que represente todos os outros serviços de dados utilizados por ela.

3.6.5.3 DataManegement

O principal objetivo desta interface é permitir aos clientes especificar como os dados virtualizados em diversas fontes são construídos. Deve prover operações de configuração e política de controle para acesso à base e os acordos que os solicitantes devem negociar com os serviços de dados. Esta interface deve definir SDEs relacionadas às virtualizações de dados, recursos e gerenciadores de recursos. Desta forma, os clientes podem monitorar o estágio do serviço e o desempenho entre outras métricas.

3.7 Utilização do WS-Agreement

Este acordo é crítico para o funcionamento dos serviços. Todos os acessos aos dados são regidos pelos acordos.

O *WS-Agreement* é composto por 3 componentes [Foster, 2003]:

- Linguagem do acordo que provê um *framework* para expressar termos do acordo entre consumidor e provedor do serviço.

- *AgreementProvider* Interface: estende a linguagem para instanciar o acordo com o provedor do serviço.
- *Agreement* Interface: estende o serviço da OGSI para prover operações de monitoramento e renegociação dos termos do acordo.

A interface *DataFactory* estende a interface *AgreementProvider* enquanto as interfaces *DataDescription*, *DataAccess* e *DataManagement* estendem a interface *Agreement*. Os termos podem ser gerais ou específicos para estender a interface *DataFactory* e pode endereçar, por exemplo, como as novas virtualizações de dados devem ser derivadas das fontes principais, definir desempenho e características de qualidade do novo serviço, como os acordos serão monitorados para garantir os termos acordados e as informações para bilhetagem.

3.8 OGSA-DQP

Para demonstrar a possibilidade de implementação dos serviços de dados, foi criado o serviço de processamento distribuído de buscas (*Distributed Query Processing* – DQP). O objetivo do DQP é prover uma maneira de implementar linguagens declarativas e de alto nível para acesso e análise de dados além de permitir a facilidade de gerenciamento de recursos que uma Grid computacional possui [Smith, 2003].

Em uma situação onde há mais de um banco de dados em ambiente distribuído é interessante possuir um mecanismo de gerenciamento de alto nível. O DQP pode ser utilizado em vários contextos: sistemas de banco de dados distribuídos, banco de dados federalizados e em *middlewares* para processo de busca.

A arquitetura proposta consiste em cumprir duas principais funções: compilar e executar a busca. Uma Grid possui mecanismos que facilitam este processo [Smith, 2003]:

- Dicionário de dados que descreve o armazenamento remoto dos dados e verifica os recursos disponíveis;
- *Scheduler* que permite distribuir a execução nos recursos disponíveis;
- Identificação dos usuários;
- Padronização através de protocolos.

O DQP utiliza duas fontes de informação para alocar os recursos: o otimizador de busca estima o custo de execução de cada parte da busca (o que permite identificar eventuais gargalos de processamento) e o *middleware* que informa quais recursos estão disponíveis.

Para o planejamento da busca é utilizado o OQL (*Object Query Language*) por ser um bom modelo para representar dados intermediários e para dividir o trabalho entre os recursos.

Na primeira fase um nó produz um plano de busca como se a execução fosse realizada em um único processador. Na segunda fase, o plano é dividido em subplanos que são alocados em recursos computacionais pelo *scheduler*. O responsável pela execução do plano é o sistema Polar*. Segundo [Smith, 2003], o Polar é um servidor de banco de dados paralelo e é nele que a lógica de distribuição da busca é realizada.

Em [Smith, 2003], há sugestões para que se estabeleçam métricas mais adequadas para distribuição do serviço além das existentes, como velocidade de CPU, dimensionamento de memória estática e disponível (dinâmica). Além disso, há indícios de que se pode melhorar a álgebra dos operadores físicos, aumentar as informações do sistema utilizadas pelo planejador do sistema, explorar algoritmos mais poderosos de programação de recursos e realizar testes de desempenho em mais bancos de dados e com maior volume de dados armazenados.

Baseado no DQP foi construído o OGSA-DQP, que é um *framework* de serviço de Grid para integração e análise de dados. Segundo [Alpdemir, 2003-2], o OGSA-DQP é um sistema de fluxo de dados distribuídos de alto desempenho que utiliza a abstração de orientação a serviços dos recursos de uma Grid e assume que os dados podem ser acessados através de interfaces. Utiliza dois serviços para atingir o objetivo: *Grid Distributed Query Service* (GDQS) e o *Grid Query Evaluation Service* (GQES). O GDQS provê a interface de comunicação com o usuário e o GQES avalia a busca submetida ao GDQS. A quantidade de instâncias GQES utilizadas dependerá basicamente das decisões tomadas pelo otimizador de busca.

O processo de criação da sessão de busca inicia pela busca e descoberta de recursos, mas não está disponível na versão atual do sistema. Utiliza-se um arquivo XML com a lista dos recursos disponíveis para uso que inclui o nome do *schema* de cada banco de dados. O GDQS obtém os metadados relacionados a cada recurso da lista. Para cada instância GDQS são realizadas a compilação, otimização, particionamento e programação de uso da busca. Cada partição é atribuída a um nó de execução. A partir desta atribuição serão criados tantos GQES quanto necessários. A coordenação do processo fica por conta do GDQS. O GDQS é semelhante à raiz, o GDS são as folhas e o GQES são o meio [Alpdemir, 2003-2].

Pode-se notar que o OGSA-DQP é uma extensão do DQP, pois utiliza o otimizador baseado no Polar*. Com isso houve um significativo avanço em direção à utilização de buscas distribuídas em um ambiente de Grid computacional, mas o serviço de busca, alocação e programação de utilização dos recursos estão fora do *middleware*.

A Grid Computacional ainda não está completamente adaptada para receber os SGBDs e estes, por sua vez, não estão suficientemente padronizados para trabalhar na Grid. A

Grid foi criada para ser um ambiente distribuído. Os SGBD foram criados para atender aos usuários de uma organização.

Para resolver esse descompasso, é necessário estabelecer (e seguir) padrões que possibilitem a integração entre ambos. As recentes pesquisas apontam para a criação de um banco de dados virtual, onde os serviços de banco de dados possam ser disponibilizados através da Grid. Esses serviços devem seguir padrões internacionalmente aceitos, em especial os *Web Services* e baseados nos conceitos da OGSA. Dessa forma fica garantida a possibilidade de computação distribuída e é possível garantir a entrega do serviço através da Grid, com mecanismos de versionamento e expiração.

Só isso, porém, não é capaz de resolver todas as questões. Os dados estarão disponíveis em servidores que por sua vez podem se utilizar de outros dispositivos para armazená-los. Saber onde esses dados estão, distribuir e encerrar o serviço é parte do problema. Da mesma forma como em outras áreas já se pensa em definir um gerenciador que coordene e fiscalize a execução dos serviços, será necessário fazê-lo com os SGBDs.

4 Planejamento em Grid Computacional

Uma das áreas que mais demandam esforços da comunidade acadêmica de pesquisa é a de Planejamento em Grid Computacional. Como há uma dispersão de recursos pela rede é necessário planejar a utilização eficaz para garantir a otimização das tarefas. Coordenar as responsabilidades dos recursos, localizá-los, programá-los e recuperar as respostas são exemplos de atividades comuns em um ambiente de Grid Computacional.

Para resolver problemas de planejamento utilizam-se técnicas de Inteligência Artificial.

De acordo com [Freuder et al., 2005], o problema de planejamento compreende uma descrição do estado inicial, uma descrição parcial dos objetivos e um conjunto de ações que mapeiam a transformação dos elementos de um estado para outro. O problema pode ser aprimorado com questões relacionadas a aspectos temporais, incertezas ou otimização de algumas propriedades. Como solução tem-se uma seqüência de ações que levam do estado inicial até o objetivo. Esta solução é conhecida por *plano*. A busca por uma solução pode ser obtida pela idealização de um objetivo como, por exemplo, a minimização das ações tomadas. Não há, em problemas clássicos de planejamento, necessidade de minimização ou maximização de funções lineares e é normal se utilizar o procedimento de *branch-and-bound* para localizar a solução, mesmo em caso de restrições.

Apesar de haver estudos que indicam a necessidade de se criar uma infraestrutura para o fluxo de trabalho para Grid [Cybok, 2004], não se tem visto a aplicação de soluções específicas.

4.1 Inteligência Artificial e Banco de Dados em Grid

O uso da Inteligência Artificial é muito importante para a integração de um SGBD com a Grid. Atualmente a Grid é capaz de localizar componentes para atingir os objetivos desejados. É possível localizar dados de entrada, réplicas e combinar requerimentos dos componentes com os recursos disponíveis. É necessário, porém, estender essas funcionalidades com representações que capturem o conhecimento atual do ambiente e disponibilize esta informação para que planejadores possam coordenar as ações através da Grid.

4.1.1 Sistema Multi-agente ou Grid de agentes

Conforme visto, as características que identificam a tecnologia de Grid são: habilidade de coordenar recursos através de controle não centralizado, utilização de padrões abertos para protocolos e interface e entrega com alta qualidade de serviço [Foster, 2002].

As diferenças entre sistemas multi-agentes e a Grid de agentes é que este último disponibiliza serviços de diretório, balanceamento de carga e distribuição de serviço (com todas regras para inferir na distribuição de serviço e negociação com outros agentes), endereçamento e protocolo de comunicação, roteamento de mensagens, segurança e visualização [Assunção et al, 2004].

Uma Grid de agentes possui quatro camadas: a primeira são os recursos, a segunda os agentes que realizam a interface com os recursos, a terceira são os serviços oferecidos e na quarta estão os agentes que utilizam os recursos da primeira camada. A ação das aplicações do cliente se realiza na quarta camada [Assunção et al, 2004].

Esta estrutura está coerente com a adoção da OGSA e com a integração através de *Web Services*.

4.1.2 Escalonamento

Segundo [Dantas, 2005], escalonamento é a atribuição de tarefas aos recursos computacionais para minimizar o tempo de execução das aplicações.

Seguindo a abordagem proposta por [Casavant e Kuhl, 1988], os métodos de escalonamento são divididos em local e global. Neste caso de estudo, contudo, interessa o escalonamento global porque se podem aplicar seus métodos em sistemas distribuídos. O escalonamento global divide-se em:

- **Estático**: a atribuição dos processos é realizada no início do planejamento e não é possível alterá-lo durante a execução do mesmo. Apesar de ser possível realizar buscas em todo espaço de solução, tanto em amplitude como em profundidade, o mais comum é utilizar heurísticas para encontrar uma solução quase ótima.
- **Dinâmico**: os processos são alocados e acompanhados, o que permite que se façam modificações durante a execução dos mesmos. É possível realizar o balanceamento de cargas entre os processadores. As decisões quanto ao balanceamento podem ser tomadas de forma centralizada ou distribuídas. O escalonamento pode ser cooperativo ou não. A forma cooperativa se dá quando as decisões a respeito do balanceamento se derem de forma não independente. Essa abordagem cooperativa é observada quando todos os elementos interagem com objetivo de atingir uma meta global para o sistema.

4.1.3 Alternativas de Planejamento

Técnicas de restrição tem sido amplamente pesquisadas para solucionar problemas de planejamento de Inteligência Artificial [Freuder et al., 2005]. Ainda segundo [Freuder et al, 2005], as unidades fundamentais dos *frameworks* baseados em restrições são as variáveis e as restrições (entidade que restringe o valor das variáveis). Em geral não se deve misturar a especificação do problema com o controle de busca. Isso porque as condições de busca que influenciam o planejamento podem mudar de um momento para outro e, por isso, nem sempre podem ser definidas antecipadamente.

Os principais *frameworks* são [Freuder et al., 2005]:

- **SAT (*Satisfiability*)**: utiliza fórmulas proposicionais e restrição de variáveis booleanas. Desta forma, utiliza-se duas variáveis relacionadas com operadores *not*, *or* e *and*. Para situações que envolvam tempo não é adequado, pois, por utilizar variáveis booleanas, mostra apenas diferenças qualitativas.
- **IP (*Integer Programming*)**: utiliza funções lineares de valores inteiros para armazenar resultados de decisão. Acha-se a solução com o procedimento de *branch-and-bound*, como em uma árvore de busca. Uma busca em toda árvore causaria problemas, mas com o uso de problemas subdivididos em partes menores e com as respectivas restrições e soluções, normalmente acha-se a solução utilizando uma pequena parte dos nós. Mesmo assim, em problemas mais complexos, pode haver um estouro na necessidade de memória e tempo de execução. Pode indicar resultados quantitativos e qualitativos. Não permite a definição de ações concorrentes.
- **CP (*Constraint Programming*)**: os problemas são descritos como problema de satisfação de restrições que consiste na utilização de variáveis associadas a um domínio e a um conjunto de restrições. Desta forma não se restringem os tipos de restrições, mas sim as variáveis de domínio finito. A semântica utilizada é mais natural e prática do que as anteriores. A satisfação de restrição é uma busca por variáveis que satisfaçam as restrições definidas. Caso se queira otimizar o processo, basta acrescentar uma variável que defina a qualidade da solução. A busca pela maximização desta variável tentará localizar uma alternativa de maior qualidade. Devido às características anteriores, este *framework* é adequado para planejamento que envolva tempo e recursos, pois permite a identificação de resultados qualitativos, quantitativos e ações concorrentes.

4.1.4 Planejadores baseados em Restrições

Há três categorias desse tipo de planejador [Freuder et al., 2005]:

- Planejamento com a indicação da restrição: utiliza as restrições em árvores de subplanos e possui uma interação limitada entre a solução de restrições e o processo de planejamento. A tarefa é reduzida a um pequeno número de atividades no nível mais baixo seguinte [Russell e Norvig, 2004]. Isso faz com que o custo computacional seja menor. É útil para planejamentos táticos, onde os procedimentos de solução são conhecidos antecipadamente. Quando utilizado em conjunto com restrições baseadas em ontologias, oferece condições satisfatórias de entendimento por parte dos usuários.
- Planejamento com grafos máximos: tenta identificar todas as possibilidades de execução das ações e coloca em um grafo. São criados subplanos que podem ser interpretados com a sintaxe booleana do SAT. A vantagem é ter condições de avaliar todo o plano, mas fica naturalmente limitada a possibilidade de solução para problemas complexos. É útil para problemas que necessitam de um número pequeno de variáveis.
- Captura do plano com a programação de restrições: não cria um estrutura de planejamento que verifique todas as possibilidades de ação. São definidas as restrições estruturais que definem os pontos de em que o plano deve tomar uma decisão e a busca inclui a melhor alternativa ao grafo como parte da solução. A busca pode variar entre a satisfação ou otimização dos valores das variáveis e a estrutura do grafo. Desta forma o plano pode ser criado com base em todos os aspectos do plano de busca.

4.1.5 Fatores importantes para programação de recursos em Grid

De acordo com Ranganathan [Ranganathan e Foster, 2003] alguns dos fatores mais importantes para programar recursos em Grid Computacional são: utilização do recurso, tempo de resposta, política global e local de acesso e escalabilidade.

Neste trabalho foram identificados quatro parâmetros importantes que afetam o desempenho em ambiente de Grid: banda de rede, padrões de acesso de usuários, qualidade da informação utilizada para realizar o trabalho e a relação entre o tempo de processamento e tamanho dos dados para determinar a importância da localização dos dados.

A contribuição mais importante desse trabalho está na constatação que o processamento dos dados devem ser realizados no próprio recurso onde os dados se encontram. Segundo os autores, outros trabalhos tratavam isso de maneira independente. Com

isso o volume de tráfego de réplica de dados é consideravelmente diminuído. Isso traz benefícios para a rede como um todo e aumenta a velocidade de processamento dos dados.

4.1.6 As atividades como um problema de Planejamento

Podem-se identificar dois possíveis ambientes para aplicação das técnicas de Inteligência Artificial para banco de dados em Grid: o planejamento clássico e o não-clássico. Considera-se o planejamento clássico para ambientes observáveis, determinísticos, finitos, estáticos e discretos. O planejamento não-clássico está relacionado aos ambientes parcialmente observáveis ou estocásticos [Russel e Norvig 2004].

De acordo com Blythe [Blythe et al, 2004], atribuir uma série de atividades em um fluxo de trabalho e alocar recursos a cada atividade é um problema de planejamento de Inteligência Artificial. Cada componente que faz parte do fluxo de trabalho é modelado como um operador de planejamento. Os efeitos e precondições dos operadores refletem a dependência de dados entre as entradas e saídas do programa e as necessidades e restrições dos recursos em termos de equipamento e *software*. O plano determinará a ordem de execução das atividades através das entradas e saídas esperadas.

Em um ambiente de Grid Computacional, pode-se imaginar estes dois cenários. Se houver uma organização virtual com recursos identificados e limitados, por exemplo, é possível ter um ambiente de aplicação para o planejamento clássico. Atualmente o que se tem realizado em Grid é este tipo de planejamento com algumas variações. Em um ambiente de Grid é possível se deparar com informações incompletas e incorretas, o que indicaria um planejamento não-clássico [Russell e Norvig, 2004]. Contudo as experiências têm se restringido a realizar o planejamento com o ambiente que se pode observar no momento em que este é elaborado.

Desta forma, as definições da arquitetura de planejamento são:

- **Estado Inicial ou Representação de estados**: deve indicar quais recursos estão disponíveis para os usuários, estimar a largura de banda entre os recursos e disponibilizar a descrição do metadado. Em geral, contudo, são condições lógicas que representam um estado com literais positivos. Um recurso poderia estar representado por Disponível (ou Indisponível), Com Programação Futura (ou Sem Programação Futura), Com Dados (ou Sem Dados), etc.
- **Representação de objetivos**: estado parcialmente representado por literais básicos positivos. Em um recurso pode-se ter como objetivo Com_Dados ˆ Programação_Futura ou ainda utilizando funções, como Programado(banco A,

serviço A, data A). Esta representação pode ser estabelecida com a indicação da atividade e o recurso que deverão ou poderão ser utilizados.

- **Representação de ações**: uma ação é especificada com base nas precondições e nos resultados obtidos depois da execução.

  Exemplo:

  Ação(Programar(b1, s1, d1),

  PRECOND: Ter(b1) ^ Ter(s1) ^ Disponível(b1,t1)

  EFECT: ¬Disponível(b1,t1) ^ Programado(b1,s1,d1))

Onde, b1 é um banco de dados, s1 é um serviço a ser executado e t1 é a data de programação do recurso.

Em Blythe [Blythe et al, 2004] pode-se encontrar uma solução implementada em três fases: preparar a especificação do problema inicial através da solicitação de informações da Grid, planejamento das atividades utilizando técnicas de Inteligência Artificial e interpretar o plano de saída gerado pelo sistema. Para realizar o planejamento foram utilizadas heurísticas locais com objetivo de maximizar a execução do plano.

Em resumo, para uma Grid Computacional deve-se definir como objetivo o comando de busca e indicar como operadores os recursos disponíveis, com suas respectivas heurísticas. O plano retorna a seqüência de atividades a serem executadas, com vinculação dos componentes e respectivos recursos. Este plano deve conter comentários sobre os motivos de utilização do recurso [Gil et al 2004]. Dessa forma, facilita-se a eventual troca de seqüência por parte do usuário.

4.1.6 Planejamento de Ordem Parcial

De acordo com Russel e Norvig [Russell e Norvig, 2004], o planejamento de ordem parcial é realizado a partir da inserção de pelo menos duas ações em um plano, sem especificar qual ação deve ser executada primeiro. É implementado através de uma busca no espaço de planos de ordem parcial. As ações serão, portanto, executadas sobre os planos através da adição de novos passos.

Ainda assim pode haver uma restrição na ordenação dos planos, sem prejuízo na paralelização das ações. Isso irá gerar um conjunto de vínculos entre os planos estabelecidos através de precondições e resultados.

A grande vantagem deste planejamento está em se atribuir ações básicas e independentes que, quando analisadas, podem ser estabelecidos pontos fictícios de início e de

fim. A paralelização da ação do plano pode ocorrer sem prejuízo para o plano como um todo. Com a finalização das ações, alcança-se o objetivo final do plano.

Abre-se também a possibilidade de se utilizar heurísticas para determinar qual *melhor* plano de execução. Em um ambiente de recursos heterogêneos e limitados, é importante que se possa considerar possíveis restrições.

4.1.8 Gerenciamento de trabalho

O problema de programar recursos em ambiente de Grid Computacional é diferente da programação de recursos em ambientes multiprocessados ou *clusters* [Ranganathan e Foster, 2003]. A localização dos recursos podem ser comprometida pela distância entre eles e a velocidade da rede pode ser sensivelmente pior do que em um ambiente restrito. Cada recurso pode ter políticas de acesso completamente diferentes e que devem ter maior precedência em relação aos serviços da Grid. As características da Grid envolvem um ambiente dinâmico com flutuações na utilização e disponibilidade dos recursos. Por este motivo o planejador deve realizar o planejamento no exato momento em que chegam as solicitações, com objetivo de utilizar e maximizar os recursos disponíveis. Esta mesma constatação foi utilizada por [Blythe et al, 2004] para implantar sua solução e está de acordo com os princípios do planejamento clássico.

A arquitetura proposta em [Ranganathan e Foster, 2003] prevê a utilização de três módulos para realizar a programação dos recursos:

- **Programador Externo**: o usuário submete as solicitações e o programador decide para qual site remoto enviará a atividade. Para isso deve utilizar informações específicas do local que enviará a solicitação.
- **Programador Local**: decide como programar as atividades alocadas nos recursos locais. Neste momento pode-se priorizar ou recusar alguns tipos de atividades.
- **Programador de Dados**: deve manter os dados que são mais acessados e eventualmente replicá-los para outros sites.

Por outro lado, em uma situação considerada ideal, as principais responsabilidades de um sistema gerenciador da Grid são [Gil et al 2004]: fiscalizar o desenvolvimento e execução das atividades programadas, coordenar atividades que tenham sido submetidas por diversas programações e aplicar as regras que garantam a execução das atividades dentro do tempo necessário.

Ainda de acordo com Gil [Gil et al, 2004], em um ambiente genérico de Grid Computacional, o gerenciador deve ser um agente dinâmico que controla as mudanças em

função de uma eventual queda de servidores ou atraso na divulgação de resultado em função de um aumento imprevisto da carga de trabalho. Isso abre a possibilidade de utilização de algoritmos específicos que utilizem o planejamento não-clássico. A programação pode ser modificada durante o processamento das demais informações. O próprio gerenciador deve ser capaz de prever algumas dessas ocorrências (desde que haja informações disponíveis sobre a indisponibilidade e utilização sazonal em níveis mais altos) e criar buscas distribuídas. Durante a execução do serviço, como em alguns processos o tempo consumido é muito grande, deve ser possível ao usuário localizar o estágio das atividades programadas. A partir dessa informação, o usuário mais uma vez poderá interferir no processo e modificar a seqüência ou utilização de recursos. Desta forma, uma solução que se apresenta viável é analisar periodicamente a submissão dos serviços a cada recurso. Quando um recurso estiver disponível e ainda houver serviços a submeter, este será utilizado. Naturalmente isto pode gerar uma alteração no planejamento inicial, mas, devido às circunstâncias e dinâmica do processo, pode ser uma alternativa viável.

Esta técnica, denominada Monitoramento e Replanejamento de Execução [Russell e Norvig, 2004], permite que se utilize qualquer técnica de planejamento para se construir um plano, mas, através do monitoramento da execução, verifica constantemente se este está sendo executado da melhor maneira possível. Quando algo sai errado ou simplesmente sai do previsto inicialmente, é possível utilizar outra alternativa que se apresente viável.

Um fato que está estabelecido é que não se devem programar recursos simplesmente através da localização de processadores ociosos, em especial quando for necessária a replicação de dados. Pode-se, por outro lado, analisar os dados que são mais requisitados e propor uma política de replicação o que permite que em outra programação a informação mais requisitada esteja disponível em outros sites.

4.1.9 Técnica de Planejamento aplicada a bancos de dados da Grid

Como as ações atômicas de pesquisas em um banco de dados podem ser descritas pelo comando SELECT em uma única tabela, utilizar o Planejamento de Ordem Parcial para definir um plano básico de ação é uma alternativa viável. Isso faz com que cada busca seja submetida a um único banco de dados, o que otimiza o processo de pesquisa. Este particionamento de uma busca complexa em diversas buscas simples e em diversos bancos de dados pode se dar em paralelo, o que agiliza o resultado da busca final.

É possível, também, monitorar o ambiente após a submissão da busca em todos os bancos de dados disponíveis. Quando o primeiro banco de dados encerrar o processamento

das linhas da tabela, recebe uma nova busca, mesmo que isso contrarie o plano inicial proposto. Quando não houver mais buscas a serem submetidas, basta aguardar a finalização dos processos em todos os bancos de dados para, em seguida, reunir o resultado das buscas simples.

O plano pode utilizar heurísticas específicas da Grid ou do próprio banco de dados para definir a ordem de execução do plano, mas deve-se considerar que o plano poderá ser modificado em caso de monitoramento dos recursos.

No final da execução do plano deve-se unir as diversas buscas em uma única para fornecer o resultado ao usuário.

4.2 Auto-gerenciamento em SGBD

Outra questão relevante para tomada de decisão do sistema planejador de banco de dados em Grid é a necessidade de um repositório com a situação atual do recurso além de dados armazenados que indiquem a utilização passada.

Os principais SGBDs comerciais possuem um padrão ótimo de desempenho. É necessário um mecanismo que ajuste a situação atual com estes padrões. O histórico de utilização é importante para informar ao gerenciador qual é e quando ocorre o pico de utilização do recurso. Isto não deve impedir que usuários experientes possam otimizar manualmente o desempenho dos SGBDs.

Os próprios SGBDs devem fornecer, via metadados, informações relevantes dessa utilização, permitindo que o gerenciador decida como e quando utilizá-lo. Mecanismos de backup e recuperação de falhas devem estar automatizados para diminuir a possibilidade de perda de informações.

4.3 Solução DQP

A solução proposta por Gounaris [Gounaris et al, 2004] indica uma forma de escalonar o uso de SGBDs de uma Grid Computacional com base no Processamento de Buscas Paralelas (DQP).

A solução recupera buscas escritas em OQL (Object Query Language) e transforma em um plano de busca que gera um grafo em formato de árvore. Os vértices da árvore representam os operadores básicos da busca e as margens o fluxo de dados (figura 4.1). As três formas clássicas de paralelismo em banco de dados são:

- **Independente**: ocorre quando há pares de subplanos de busca independentes, ou seja, os subplanos não dependem um do outro.

- ***Pipelined***: ocorre quando a saída de um operador é utilizada por outro operador conforme aquele é gerado. Neste caso ambos os operadores são executados em concorrência.
- **Particionada**: um operador físico do plano de busca tem várias cópias sendo que cada uma delas processa um subconjunto de dados.

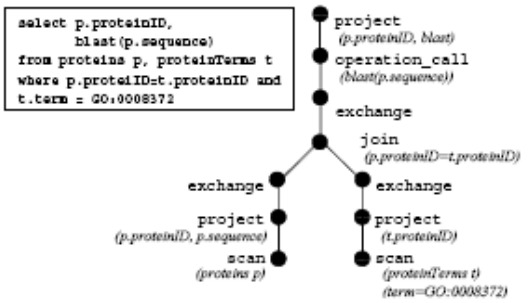


Figura 4.1: Exemplo de Plano de Busca [Gounaris et al, 2004]

Segundo os autores, mesmo em sistemas homogêneos, achar o maior grau de paralelismo diminui a eficiência da utilização dos recursos e diminui o desempenho da aplicação como um todo.

Eles identificam que o problema de escalonamento da Grid está no fato de não limitar o grau de paralelismo, em não determinar quantas e quais máquinas utilizar e qual parte do plano de busca cada recurso pode realizar.

Entende-se a eficiência da paralelização de buscas a possibilidade de realizar o paralelismo de buscas em ambientes heterogêneos com inúmeros recursos e balancear o desempenho e eficiência na utilização desses recursos.

A busca de uma heurística eficiente compreende a combinação de recursos, a distribuição da carga de trabalho e a otimização dos planos de busca.

O algoritmo proposto está representado na figura 4.2.

O algoritmo é composto por dois laços. O mais externo poderá ser repetido até o limite do número de operadores de busca. O interno poderá ser repetido até o limite do número de computadores disponíveis na Grid.

Através deste algoritmo, um plano de busca otimizado de um único nó particionado, resultado da transformação da busca especificada via OQL pelo Polar*, os computadores candidatos para realizarem a operação e uma variável que tem o objetivo de estabelecer o limite para a melhoria do desempenho são os parâmetros de entrada. As heurísticas utilizadas para selecionar os computadores candidatos envolvem: poder de processamento (CPU),

memória, velocidade de I/O, velocidade de conexão, proximidade. Essas heurísticas são extraídas da Grid Computacional através do [GrADS, 2006].

O processo se inicia com o plano de busca. Este plano deve preferencialmente não ter paralelismo. O plano é separado em cada um dos subplanos de busca, o que gera um particionamento da busca original. Cada operador identificado no plano de busca é atribuído a um computador. Os operadores podem ser:

- Operadores de troca (que encapsulam o paralelismo e envolvem comunicação e estão representados na figura 1 por *Exchange*);
- Chamadas de operações (que encapsulam funções definidas pelo usuário e estão representados na figura 1 como *operation_call*); varredura (que herdam os custos de entrada e saída e estão representados na figura 1 como *scan*);
- Projeto (que indicam corte nos dados e estão representados como *Project*) e
- Demais operadores como uniões (*joins*), utilização de CPU e comunicação entre outros que indicam custo para realizar a operação.

Depois de particionada a busca, aumenta-se um grau para cada parte do subplano de busca em um determinado tempo de execução. Passe-se a estimar o custo do subplano de busca: aloca-se um recurso com base na localização do dado. O recurso somente será alocado onde os dados estejam disponíveis, ou seja, não há previsão de utilização de réplicas no algoritmo.

A análise de viabilidade de particionamento da busca é feita para cada operador da busca. Inicia-se pelo operador de maior custo e o computador disponível para execução desta parte da busca. Se o aumento do desempenho for maior que o limite estabelecido no parâmetro de entrada do algoritmo, aumenta-se um grau de paralelismo e estima-se novamente o custo da busca com a nova formatação do plano. Este processo é feito sucessivamente até que se extraia um plano de busca dentro dos limites impostos pelo parâmetro do algoritmo.

Com esse algoritmo é possível localizar um plano de busca viável e que não demande muito tempo de planejamento. Quanto menor for o limite informado como parâmetro, mais perto do ótimo estará o resultado e maior será o tempo de preparação do plano. De acordo com os autores é improvável que este algoritmo funcione bem em processamentos massivos.

```
repeat
   Operator Op = getCostlierParallelisableOp()
   Criteria[] L = getCriteria(Op)
   float a = getThreshold()
   repeat
      Machines[] M = getAvailMachines()
      checkMoreParallelism(Op, M, L, a)
   until no changes
until no changes
```

Figura 4.2: Algoritmo proposto em [Gounaris et al, 2004]

Os resultados alcançados na implementação do algoritmo estão relacionados à identificação do custo da busca:

- Atribuição de um custo para cada plano de busca paralelo
- Atribuição de um custo para cada operador físico do plano de busca
- Diminuição da influência entre o custo e o algoritmo de programação

Ao especificar melhor os custos de uma busca, é possível estabelecer heurísticas que permitam aprimorar o modelo.

4.4 Crítica

A solução proposta pelo DQP apresenta relevância considerável dentro do que se propõe a realizar. Contudo, no sítio DQP [OGSA-DQP, 2006] encontra-se uma série de limitações para o projeto. Dentre os quais destacam-se:

- Não pesquisam os metadados dos computadores envolvidos na busca e estas informações não são utilizadas durante o escalonamento. Com isso toda heurística fica prejudicada.
- Suporte apenas ao MySQL.
- As buscas estão restritas a alguns tipos de dados (long, integer, string, double e float) e não suporta tipos de dados como: date, time, etc.
- Suporte a buscas escritas em OQL. Esta não é uma linguagem padrão de busca utilizada pelo público não-acadêmico. Naturalmente há uma dificuldade em reescrever todas as buscas dentro desse novo padrão.
- As buscas devem ter todas as tabelas na cláusula FROM e unidas pela cláusula WHERE. Só há suporte ao uso de AND na cláusula WHERE.
- Não se pode utilizar o OGSA-DAI para indicar serviços de dados do OGSA-DQP.
- Está limitado aos SGBDs do computador onde está instalado o OGSA-DQP.

Estas limitações estão relacionadas basicamente a não utilização do padrão proposto pelo OGSA-DAI. A comunidade acadêmica e até comercial tem dado preferência à utilização de padrões para evitar limitações e necessidade de adaptação a novos modelos. O OGSA-DQP está baseado no projeto Polar* que é responsável pelo particionamento da busca. Isso aumenta a complexidade de instalação e configuração do ambiente.

O OGSA-DAI não é um trabalho concluído, mas tem sido aprimorado ao longo dos anos de acordo com o grupo de trabalho OGSA-DAIS que pretende estabelecer o padrão de acesso e manipulação em banco de dados.

A própria documentação do projeto incentiva que se estendam os padrões para necessidades complementares. Com isso, é possível fazer com que os avanços realizados a cada nova versão possam ser incorporados ao processo de escalonamento e planejamento dos recursos que o *middleware* é capaz de identificar.

Ao se idealizar uma solução com a utilização de técnicas de IA, o sistema de planejamento deve receber como entrada o objetivo a ser alcançado, representado por um comando de busca e uma biblioteca de recursos que se pode utilizar para modificar o estado. Os recursos devem ser capazes de indicar a respectiva representação do estado atual, como nível de utilização, velocidade e metadados de cada SGBD disponível na Grid. Antes mesmo de planejar a execução da atividade, é necessário subdividir o comando de busca (*query*) em partes que possam ser executadas em paralelo, extraindo, desta forma, o máximo dos recursos disponíveis. É necessário saber os recursos que se pode contar antes da submissão da tarefa e quais informações estão disponíveis em cada um dos recursos (metadados).

Cada ação deve ter uma descrição do estado (pré-condição) e uma descrição das mudanças de estado (efeitos).

Após estas atividades, a utilização de técnicas de planejamento de distribuição de trabalho é necessária para: determinar qual SGBD realizará o trabalho, localizar e utilizar eventuais réplicas dos dados necessários, determinar quando pode ser esperado o retorno da informação, recuperá-la, uni-la às diversas subconsultas para produzir um resultado único e determinar alternativas em caso de falha de um ou mais recursos. Estas informações devem estar disponíveis para o planejador. O sistema deve ser desenvolvido sob uma arquitetura orientada a serviços para que se maximize a adoção e integração com diversos tipos de sistema.

Tecnicamente, ao adotar esta estrutura, é possível combinar os SGBDs em um espaço de estados com heurística.

Parte II: Proposta

5 Especificação do Problema

O problema de coordenar as atividades para gerar um fluxo de ações a serem executadas e a vinculação das ações aos recursos que irão executá-las através de um *middleware* pode ser formulado como um problema de planejamento que pode ser solucionado com algoritmos de Inteligência Artificial. O objetivo é criar um mecanismo que possa estar entre as solicitações do usuário e a execução nos diversos bancos de dados disponíveis na Grid Computacional. Portanto evita-se fixar em um único tipo de busca ou problema.

Basicamente sabe-se que o planejador deve se basear em um *middleware* que já exista e tenha suporte às principais atividades de um banco de dados além de não ter vínculo exclusivo com uma única implementação comercial de banco de dados. Isso faz com que haja necessidade de estabelecer uma interface única para qualquer tipo de banco de dados. Sabe-se que a melhor forma de realizar trabalhos em uma Grid Computacional é estabelecer o desenvolvimento baseado em serviços. A integração com os *middlewares* disponíveis é importante para que não se limite e se permita a evolução da aplicação da solução.

Em aplicações relacionadas a um SGBD é possível identificar duas maneiras básicas para especificar o que deve ser feito: a execução de comandos DML/DQL (*Data Manipulation Language* e *Data Query Language*) e execução de procedimentos armazenados.

No caso de um procedimento armazenado a especificação deve ser mais rigorosa, visto que deverá definir a lógica do processo. Esta definição foge ao escopo deste trabalho porque ainda não há um padrão definido para linguagens de programação para banco de dados em Grid Computacional. Uma alternativa que se mostra viável é a utilização da linguagem Java em SGBD. Já é possível escrever códigos em Java no banco de dados Oracle, no PostGreSQL e no DB2 da IBM. No SQL Server 2005 é possível utilizar o C# para criar funções estendidas. De qualquer forma, este não é o problema principal, visto que é necessário manter uma biblioteca de funções e procedimentos vinculados aos SGBD para que estes possam executar as funções. O problema principal está em estabelecer a comunicação com diversos SGBD e permitir que se localize e manipule dados em qualquer dos recursos disponíveis da Grid. Uma vez estabelecida a comunicação e ao permitir a manutenção e recuperação de dados na Grid, o problema central estará equacionado.

Por outro lado, o grupo de trabalho envolvido na definição dos serviços da Grid Computacional (OGSA-DAIS) já definiu a estrutura para manipulação de dados o que viabiliza a solução apresentada. A implementação de parte das soluções idealizadas já é uma realidade e existe um *middleware* (OGSA-DAI) que possibilita estabelecer a integração entre solicitações e recursos.

5.1 Justificativa do uso de IA

O problema de planejamento do uso de banco de dados em um ambiente distribuído pode ser especificado a partir da disponibilidade das informações. Em um banco de dados relacional, como as informações são extraídas de tabelas e estas tabelas estão, potencialmente, presentes em diversos bancos de dados através de mecanismos de replicação, pode-se realizar buscas que visem maximizar o uso dos recursos. Contudo a solução para este problema em um ambiente completamente observável, como é o caso de *clusters*, tem sido amplamente discutida na comunidade acadêmica e os resultados, representados por produtos comerciais, tem sido satisfatórios.

Isto se dá porque em *clusters*, diferentemente do que ocorre em Grid Computacional, o ambiente detém completo controle sobre a distribuição dos dados. Em uma Grid Computacional o desafio está em estabelecer o uso dos recursos de maneira dinâmica, sem enxergar todos os bancos de dados como extensões de um único ambiente.

Em bancos de dados relacionais, a unidade básica de pesquisa é o comando SELECT. Consultas complexas e filtros exigem maior esforço do banco de dados. Paralelizar as buscas tem se mostrado um mecanismo eficiente em ambientes de produção.

Mesmo que se considere uma Organização Virtual como um ambiente completamente observável, nem sempre sabemos a localização exata das tabelas visto que podem estar em diversos bancos de dados. O espaço de estados deve ser observado sempre que se propor uma submissão de trabalhos (recursos disponíveis para realizar as buscas). Pode-se utilizar o Planejamento de Ordem Parcial para realizar e separar as ações básicas do plano, visto que cada consulta simples é tratada como uma ação independente e paralelizável. O fato de se paralelizar as buscas particionadas permite que se maximize o uso dos recursos dentro de uma Organização Virtual. Definir as etapas básicas do processo, como verificar o espaço de estados, apurar métricas, vincular bancos de dados e tabelas e, por fim, identificar os comandos básicos de busca para determinar pré-condições e efeitos, são tarefas que podem ser resolvidas através de algoritmos de IA.

O monitoramento se faz necessário para acompanhar os resultados e retornar a informação ao usuário. O monitoramento pode, inclusive, fazer com que eventuais falhas do sistema planejador sejam corrigidos em tempo de processamento.

Como se pode inferir, o uso dos algoritmos de planejamento desenvolvidos ao longo do tempo com as técnicas de IA contribuem para resolver o problema da submissão das buscas. Algoritmos de monitoramento e até mesmo algoritmos de planejamento condicionais podem ser utilizados para potencializar a eficiência do sistema planejador.

5.2 Arquitetura Orientada a Serviços

Para que o desenvolvimento siga o padrão de arquitetura orientada a serviços, as necessidades do usuário devem ser descritas através *do que* deve ser feito. Uma vez que a linguagem SQL também é baseada no mesmo princípio, pois esconde a complexidade da operação de *como* será manipulada a informação nos arquivos físicos, a própria estrutura do comando SELECT poderá ser utilizada.

Ao formular o comando SELECT, o usuário define:

a.) Em qual(ais) tabela(s) a informação está
b.) Quais colunas ele precisa
c.) Quais linhas ele precisa
d.) Indica o relacionamento entre as tabelas (quando for mais de uma)
e.) Indica a ordem pela qual se quer enxergar o resultado
f.) Indica eventuais agrupamentos da informação
g.) Impor condições para filtrar as linhas que devem retornar

5.3 Fases do Planejador

O sistema planejador será dividido em quatro partes principais:

- Interceptar as requisições do usuário que são enviadas para o *middleware*
- Solicitar ao *middleware* as informações dos recursos disponíveis na Grid Computacional
- Especificar o domínio do problema de acordo com os padrões da Inteligência Artificial e planejar a execução
- Programar os recursos e distribuir as solicitações fruto do planejamento para que o *middleware* execute nos SGBDs.

A figura 1 demonstra as entradas e saídas esperadas pelo planejador. Deve haver um processo que identifique o estado atual dos recursos e outro para definir as necessidades do usuário. Ambos servem para iniciar o processo de planejamento. A identificação dos recursos e a forma de comunicação com os SGBDs serão delegadas ao OGSA-DAI. As necessidades do usuário deverão estar especificadas dentro do formato utilizado na OGSA-DAI.

O usuário informa sua necessidade de busca complexa dentro da estrutura proposta na OGSA-DAI. O planejador solicita ao *middleware* informações dos recursos disponíveis e verifica quais recursos possuem as tabelas necessárias para execução do comando. O planejador particiona a busca em comandos que possam ser executados paralelamente em diversos SGBD e vincula aos recursos disponíveis. Após o estabelecimento do fluxo de trabalho, o serviço será enviado à Grid Computacional para ser processado nos gerenciadores de bancos de dados. Terminado o processo, o planejador une as linhas retornadas e devolve ao usuário que solicitou a busca.

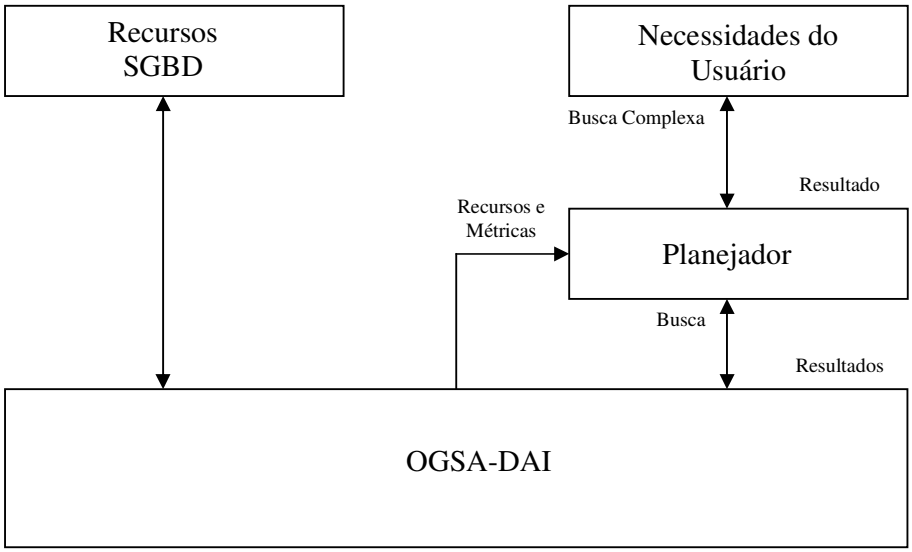


Figura 5.1 – Entradas e Saídas do sistema planejador

Para isso é necessário especificar os objetivos, pré-condições e efeitos esperados, estabelecer uma heurística válida e consistente com o objetivo, excluir as restrições identificadas pelo usuário e implementar regras de busca. A heurística deve ser estabelecida com base nas informações dos recursos e são representadas pelas métricas na figura 5.1.

5.4 Objetivo

Cada elemento do comando SELECT será um componente que demandará uma ação pelo planejador. Cada elemento será, portanto, um operador do planejamento. O objetivo será o conjunto de dados definidos nas colunas do comando.

5.5 Pré-condições e Efeitos

Para cada uma das ações propostas no sistema planejador haverá pré-condições que, após o processamento, gerarão os efeitos para a nova etapa. Desta forma, ao se começar a ação baseado na consulta complexa, obtém-se as consultas particionadas. Ao unir as consultas particionadas e os recursos disponíveis, é possível extrair as métricas e vincular os bancos de dados que possuem as tabelas para realizar as buscas. Com base nas métricas, bancos de dados e buscas, pode-se efetuar o planejamento de ordem parcial.

As pré-condições e os efeitos de cada busca serão definidos pela localização dos dados nas tabelas. Note-se que as tabelas, por fazerem parte de um ambiente tipicamente distribuído, poderão estar replicadas em alguns servidores de banco de dados. A localização da informação deve se dar através do *metadados* da Grid.

O planejador indicará a ordem na qual as ações deverão ser realizadas para atingir o objetivo. Com o resultado final de cada uma das buscas particionadas, é possível unir o resultado para apresentação final ao usuário.

5.6 Heurística

A transferência de dados pela rede deve ser modelada como um operador do planejamento e deve compor o custo da ação, juntamente com o custo estimado do processamento no banco de dados, com o poder de processamento do computador onde está instalado o SGBD, memória disponível e espaço no HD.

O custo da transferência de dados pode ser medida pela banda e proximidade dos nós da rede. Quanto maior a banda menor o custo de transferência. Quanto mais próximo, menor o custo. Deve ser definido um operador para indicar qual limite mínimo de banda deverá ser descartada em função do volume de tráfego de dados esperado.

O custo do processamento da consulta no banco de dados deve basear-se, sempre que possível, no padrão de cada recurso. Deve-se priorizar a utilização de CBO (*Cost-Based Optimizer*) em favor do RBO (*Rule-Based Optimizer*) pela maior confiabilidade que o primeiro confere.

O custo do banco de dados é composto por uma série de parâmetros disponíveis nas implementações comerciais dos produtos. Mede-se basicamente por três aspectos:

a.) Percentual utilizado da CPU

b.) Velocidade de entrada e saída de dados

c.) Uso e limitação de memória volátil

Para que se tenha maior segurança na qualidade do recurso, será utilizado, sempre que a informação estiver disponível no banco de dados, o MTBF (*Mean Time Between Failure*) que indica o tempo médio de falhas.

Estas informações serão mantidas por processos específicos de cada SGBD sempre que possível e comporá o pacote de instalação da solução. A intenção é criar um mecanismo independente de outros serviços da Grid (como MDS), pois que estes trazem informações muito mais vinculadas aos servidores e nada sobre o banco de dados. A simplificação da estrutura de busca é fundamental para agilizar o processo de escalonamento e planejamento de utilização dos recursos.

5.7 Restrições

Um planejador deve prever as restrições de *hardware* e *software* [Blythe et al, 2004]. No caso de bancos de dados relacionais a maior parte destes requisitos estão encapsulados. As limitações de *hardware* normalmente envolvem um tipo específico de equipamento, memória ou espaço em disco e as de *software* estão vinculadas à utilização da linguagem SQL, um padrão para gerenciadores de bancos de dados. O sistema operacional também é escondido pelo SGBD.

Portanto, para um planejador de ações em gerenciadores de banco de dados, basta que se defina qual ou quais implementações serão utilizadas.

O estado inicial do ambiente será definido pela informação dos gerenciadores de bancos de dados disponíveis, pelas métricas extraídas dos recursos e pelo custo estimado do processamento da consulta.

5.8 Regras de Busca

O sistema planejador deverá verificar todos os espaços de estados possíveis dadas as restrições do usuário e as condições identificadas na Grid Computacional. A escolha será sempre baseada na solução que demande menor tempo de processamento e permita maior possibilidade de desvio em caso de falhas.

5.9 Proposta de Implementação

Utilizou-se a UML porque é uma linguagem adequada para modelar sistemas orientados a serviços. A figura 5.2 identifica cada uma das atividades que o sistema planejador deve realizar e os atores que fazem parte do processo.

Optou-se por utilizar um caso de uso para a execução do planejamento recebendo apenas as requisições do usuário e casos de uso que obrigatoriamente serão executados para: descobrir os recursos presentes na Grid Computacional, recuperar métricas dos bancos de dados e realizar o particionamento do comando SELECT. Como estes casos de uso poderão ser utilizados em outras funções do ambiente, eles foram modelados separadamente. A Grid Computacional, por ser o próprio ambiente, não foi modelada como um ator ou mesmo caso de uso.

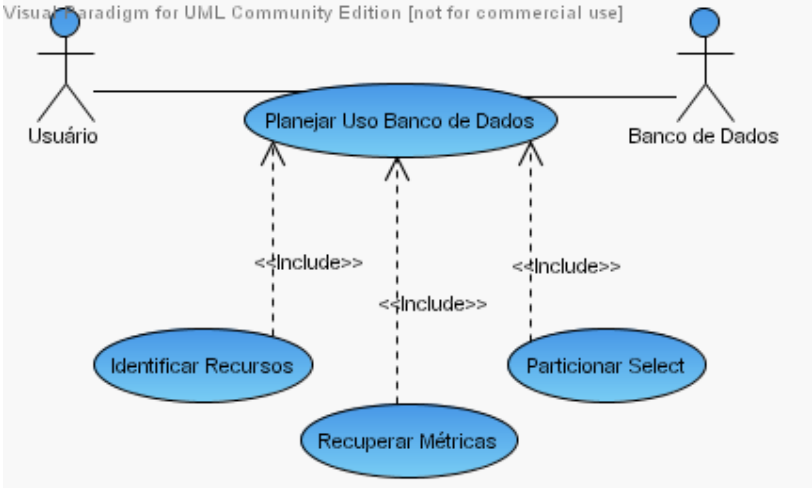


Figura 5.2 – Diagrama de Caso de Uso da proposta

5.9.1 Algoritmos e Heurística

O algoritmo utilizado para criar o plano será o planejamento de ordem parcial com monitoramento. Isso porque cada um dos algoritmos possui características que poderão ser utilizadas isoladamente ou em conjunto, dado que um planejamento é um exercício de análise das possíveis combinações de ações para atingir o resultado esperado [Russel e Norvig 2004].

No planejamento de ordem parcial é possível indicar a seqüência de ação sobre os planos. Assim é possível dividir o planejamento em tópicos (planos) independentes e que podem ser executados internamente ou em paralelo em qualquer ordem. Naturalmente é necessário estabelecer as restrições de ordenação e os vínculos causais entre os planos.

Normalmente é necessário buscar uma série de informações dispersas em diferentes tabelas no banco de dados antes de se consolidar os resultados para apresentação. A busca pode ser realizada em diversas tabelas, em diversos bancos de dados, simultaneamente, sem necessidade de uma ordem pré-estabelecida. A única restrição é que isso seja feito antes da consolidação de dados.

No caso específico da Grid Computacional, onde é possível dividir o problema em uma série de pequenos objetivos (sub-objetivos serializáveis [Russel e Norvig 2004]), acredita-se que o agente será suficientemente rápido para controlar as ações dispersas na Grid.

O grafo de planejamento é estabelecido em níveis que representam períodos de tempo. Cada nível possui um conjunto de literais e ações relacionados e será possível extrair uma heurística do resultado do grafo [Russel e Norvig 2004].

Em um planejamento de banco de dados em Grid, é possível utilizar o grafo de planejamento para identificar se uma determinada seqüência de ações irá ou não produzir o efeito desejado (se é ou não possível executar o plano) e determinar se há ações ou situações que são mutuamente exclusivas.

Além da heurística que pode ser extraída do grafo de planejamento, é importante estabelecer restrições para os recursos computacionais. Estas restrições determinarão a ordem e a prioridade no uso dos bancos de dados.

Um banco de dados relacional depende basicamente de quatro situações: processador, rede, memória e meio de armazenamento físico (E/S). A heurística deverá conter estes elementos em uma proporção que cada implementação de banco de dados deverá determinar. A proposta é que cada fornecedor de banco de dados forneça a proporção de utilização de cada recurso, inclusive com a limitação imposta pelas necessidades dos usuários internos da organização [Foster et al 2001].

Uma medida importante para tomada de decisão do planejador é o MTBF (Mean Time Between Failures). Este número indica o tempo médio de falha de um determinado recurso (disco rígido, por exemplo). Alguns gerenciadores de bancos de dados possuem métricas que permitem identificar o nível de falha de periféricos, a última falha ocorrida e o nível de utilização histórico do banco de dados. Estas informações permitirão ao planejador inferir sobre a confiabilidade de determinado recurso e, eventualmente, propor caminhos alternativos de execução do plano.

Com estas informações dos SGBDs aliadas ao grafo de planejamento será possível otimizar a seqüência tendo como base o tempo estimado de execução. A solução poderá ser formulada em um problema de satisfação de restrições booleano e poderá ser resolvido com o

algoritmo de conflito mínimo. A codificação proposicional envolve a definição do estado inicial, objetivo, axiomas sucessores, de precondições e de exclusão de ações (ou restrição de estados). Dessa forma, qualquer modelo que atenda à sentença proposicional será um plano válido para o problema [Russel e Norvig 2004].

As codificações proposicionais são complexas e podem consumir grande volume de recursos [Russel e Norvig 2004]. Os símbolos de ação seriam suficientemente grandes para inviabilizar a aplicação desta técnica. Portanto é necessário reduzi-lo e, no caso do banco de dados em Grid, utilizar redes semânticas com predicados binários. Dessa forma, cada argumento será descrito e analisado separadamente [Gil et al 2004].

A identificação de uma única variável de cada SGBD para indicar quanto há de disponibilidade do recurso, simplificará a utilização da lógica proposicional na elaboração do plano de ação.

5.9.2 Algoritmo proposto

Com base nas diretrizes expostas, o algoritmo proposto para criar o plano de trabalho está estabelecido como proposto na figura 5.3.

```
Entrada (estado inicial):
        Consulta Complexa
Processo:
        LER recursos para objetos (operadores de planejamento)
        LER consulta
        << Separar itens da consulta em consultas simples >>
                - tabelas da cláusula FROM
                - joins
        PARA cada item de consulta
                <<LER quais recursos possuem tabela>>
                        BUSCAR no dicionário de dados dos recursos
                        ARMAZENAR métricas do recurso (métrica_recurso)
                PARA cada tabela (pré-condição)
                        PARA cada recurso
                                SE tabela está no dicionário de dados
                                        CRIA recurso_tabela
        PARA cada recurso_tabela
                PARA cada métrica_recurso
                        SE métrica(recurso 1) > métrica(recurso 2)
                                CRIA base de consulta no recurso
        SUBMETE buscas
        UNE resultados
Saída (objetivo):
        Resultado da consulta complexa
```

Figura 5.3: Algoritmo proposto.

O algoritmo contém três laços: o primeiro é executado com base na separação dos itens da busca; o segundo em cada uma das tabelas especificadas na busca; e por fim, após o

vínculo do recurso com a tabela, verifica o recurso com melhor métrica para realizar o planejamento.

Como parâmetro de entrada, será interceptado recebida a consulta complexa no *middleware* (OGSA-DAI). Serão solicitadas ao *middleware* informações sobre quais são os recursos (bancos de dados) disponíveis na Grid.

Com base nestas informações, serão lidos os recursos para os objetos e importados os objetos de consulta. Um método irá separar cada uma das tabelas e respectivas colunas de busca. Deste método retornarão, além das tabelas e colunas, as uniões entre as tabelas e as eventuais funções a serem utilizadas no banco de dados.

Para cada item da consulta, serão identificados quais bancos de dados estão disponíveis para busca. Será possível extrair métricas de desempenho do banco de dados. Em seguida para cada tabela necessária para realizar a consulta serão identificados os recursos disponíveis. Isso será feito através de uma busca no dicionário de dados do banco de dados.

Para cada uma das tabelas já vinculadas a cada um dos possíveis recursos, deverá ser verificada a métrica extraída do banco de dados. Os recursos com melhor métrica receberão o comando de busca para se submeter ao *middleware*.

Cada item do processo possui como pré-requisito o resultado da etapa anterior. O efeito de cada etapa, por sua vez, apontará para um estado específico necessário para etapa seguinte, conforme pode ser observado na figura 5.4. A submissão das buscas particionadas, por serem atômicas, podem se realizar em paralelo.

Iniciar(Em(Busca Complexa) ^ Em(Recursos Disponíveis))
Objetivo(Em(Resultado Busca))
Ação(Separar(Busca Complexa))
PRECOND: Em (Busca Complexa)
EFFECT: Em (Buscas Simples)
Ação(ApurarMétrica(Recursos Disponíveis))
PRECOND: Em (Recursos Disponíveis)
EFFECT: Em (Métrica)
Ação(VincularBusca(Buscas Simples, Recursos Disponíveis))
PRECOND: Em (Busca Simples, Métrica)
EFFECT: Em (Busca, Recurso)
Ação(SubmeterBusca(Busca, Recurso))
PRECOND: Em (Busca, Recurso)
EFFECT: Em (Resultado Busca)
Ação(MostrarResultado (Resultado Busca))
PRECOND: Em (SubmeterBusca)
EFFECT: Em (Resultado Final)

Figura 5.4 Descrição do Problema de Planejamento de Banco de Dados em Grid

No final do processo haverá uma série de tabelas temporárias que serão unificadas no comando complexo para oferecer o resultado final ao usuário.

O usuário continuará criando o programa para processamento no *middleware* dentro dos padrões propostos pelo ambiente e receberá o resultado final sem a necessidade de modificar o que é feito atualmente. Toda complexidade de transformar o comando e extrair o resultado ficará sob a responsabilidade do sistema.

Não se corre o risco de estourar o tempo de planejamento, visto que o limite do algoritmo está no número de tabelas da busca e no número de recursos disponíveis. A menos que haja um volume extremamente grande de recursos, o tempo de processamento deve ser coerente com o ganho na distribuição do processamento.

Uma outra vantagem do algoritmo é que, caso não haja recurso disponível para tratar a solicitação como, por exemplo, uma ou mais tabelas não estarem presentes nos recursos disponíveis, este será interrompido.

6 Metodologia

Com base na especificação proposta, foi elaborado um sistema que atende mais aos requisitos de planejamento do que de escalonamento, utilizando a técnica de planejamento de ordem parcial, com decomposição de ações, mas sem a restrição de recursos. Um segundo sistema foi desenvolvido para monitorar o ambiente e aguardar o resultado da primeira submissão em todos os recursos disponíveis para, assim que o primeiro recurso encerrar o processo, receber novas submissões, se houver.

Visto que em sistemas gerenciadores de banco de dados o tempo de resposta de uma busca submetida pode variar em função da quantidade de linhas e colunas processadas, memória disponível e carga de processamento do servidor e, que, quando a busca é processada, pode-se obter uma resposta completa positiva ou negativa, não há como se determinar prazos a para execução deste tipo de tarefa. Portanto o que se pretende realizar está restrito a uma tarefa de planejamento e submissão das buscas.

Não há restrições aplicáveis ao modelo, visto que ou a informação pode estar disponível ou não. Como a busca pode ser realizada em um ambiente completamente observável, optou-se por esta forma de planejamento. Antes de iniciar o planejamento, é realizada uma busca para identificar a existência ou não das tabelas em cada um dos bancos de dados disponíveis. Quando houver a tabela no banco, a busca será realizada e, a princípio, as linhas serão retornadas. A partir desta premissa, foram identificados dois cenários distintos:

- Planejamento de Ordem Parcial: a partir da decomposição do comando SELECT em seus elementos básicos, é possível realizar simultaneamente tantas instruções quantas forem possíveis sem que haja necessidade de ordem pré-estabelecida. Ao final de cada comando SELECT, as linhas serão unidas para retornar o resultado final da busca, conforme pode ser visto na figura 6.1.

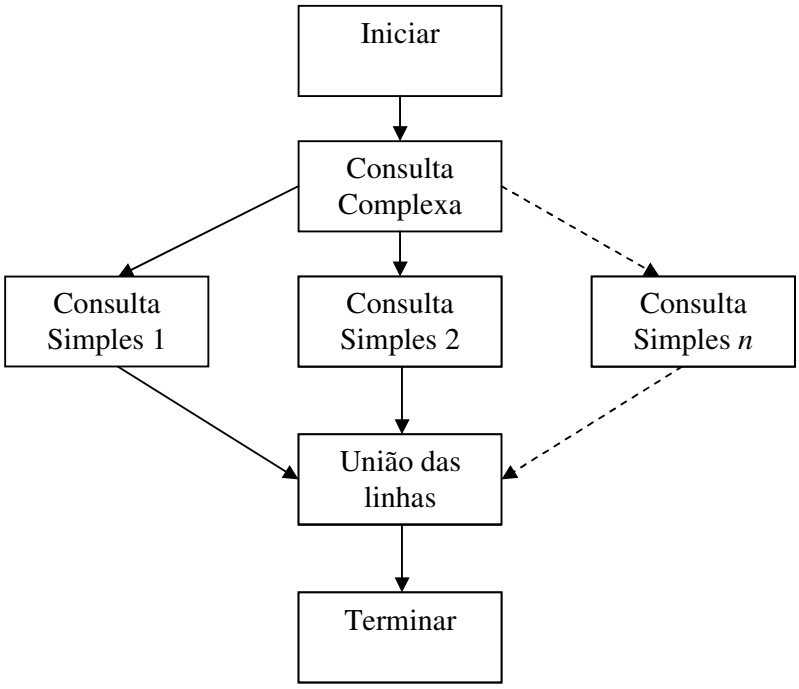


Figura 6.1: Planejamento de Ordem Parcial de Consultas Complexas

- Planejamento Condicional com Restrição de Recursos e Monitoramento: parte-se do Planejamento de Ordem Parcial, mas verifica-se a possibilidade de haver menos recursos disponíveis do que o número de buscas a serem submetidas. Isso ocorrerá somente caso seja uma busca com mais de 2 tabelas. Quando isto acontecer, o planejador monitorará as buscas submetidas a todos os bancos de dados e o primeiro que encerrar o processo receberá novas buscas para execução, conforme mostra a figura 6.2.

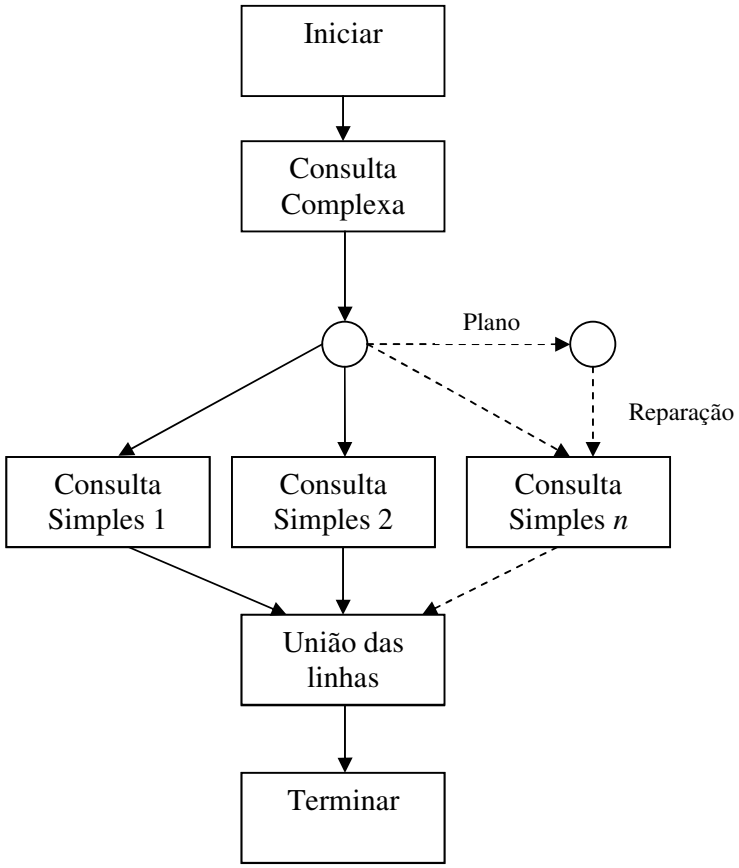


Figura 6.2: Planejamento de Ordem Parcial com Restrição de Recursos e Monitoramento

No primeiro teste (Planejamento de Ordem Parcial), assume-se que, mesmo que o sistema gerenciador de banco de dados venha a realizar mais de uma busca ao mesmo tempo, isso não representa uma restrição. Os gerenciadores de banco de dados atuais são capazes de executar as buscas submetidas sem ter que interromper a execução de uma busca para implementar a outra. No segundo teste (Planejamento Condicional com Monitoramento), submetem-se todas as buscas simples em todos bancos disponíveis e aguarda-se o primeiro banco a encerrar a busca. O banco que encerrou a busca inicia, então, a próxima busca do planejamento.

6.1 Módulo Desenvolvido

A partir da instalação do ambiente, iniciou-se a criação do módulo para realizar o planejamento da utilização dos recursos e com a utilização dos dois bancos de dados disponíveis. A plataforma de desenvolvimento escolhida foi o Java. Por haver uma série de exemplos disponíveis no ambiente Java para a realização de buscas, foram analisadas todas as

possibilidades e APIs disponíveis para criação do módulo. Assim, o ambiente de desenvolvimento escolhido foi o Eclipse versão 3.0, por ter total compatibilidade com o Java e não ser dependente de um distribuidor específico.

O *middleware* OGSA-DAI não possui um mecanismo de paralelização das buscas em diferentes bancos de dados (*DataServices*). É possível, através da versão atual, submeter as buscas em paralelo, desde que se utilize um único banco de dados. Quando o processo é encerrado e controle retorna à atividade que, então, pode submeter nova busca.

Como isso não atendia o que era proposto realizar, foi necessário criar um mecanismo que estabelecesse as buscas através de *Threads*. Uma classe foi criada para implementar a *Thread* e reter algumas informações importantes para acompanhar o andamento das consultas. Desta forma foi possível notar que, mesmo em *DataServices* diferentes o sistema conseguia acrescentar dados em todas as tabelas temporárias ao mesmo tempo.

Por haver uma série de exemplos disponíveis no ambiente para realização de buscas, foram analisadas todas as possibilidades e APIs disponíveis para criação do módulo. O módulo foi desenvolvido em uma única classe, cujos métodos são:

- **getResources**: localiza e retorna os recursos disponíveis no *middleware*.
- **splitSelect**: divide o comando SELECT com base nas tabelas de busca.
- **checkDataResources**: verifica a existência de cada uma das tabelas de busca em cada um dos recursos disponíveis.
- **getMetrics**: verifica a métrica de cada recurso disponível.
- **isWhereAlias**: verifica a existência de apelidos nas tabelas de busca.
- **tableMetadata**: verifica a existência da tabela no metadados dos recursos envolvidos.
- **metric**: avalia a métrica dos recursos envolvidos na busca.
- **DTOutputStream** e **performBulkLoad**: para unir as buscas e fornecer o resultado final.

6.1.1 Identificação dos recursos disponíveis

O método **getResources** localiza, a partir da *Universal Resource Locator* (URL) do serviço de dados da OGSA-DAI quais são os recursos que estão disponíveis naquele momento. São utilizadas funções específicas da API do *middleware* para localizar e identificar os recursos.

6.1.2 Separação dos elementos do comando SELECT (*parse*)

O método **splitSelect** recebe o comando SELECT que será executado, exatamente como os usuários e as aplicações normalmente submetem ao banco de dados. Foi criado um pequeno mecanismo de transformação de consultas complexas em pequenas consultas, cada uma com uma única tabela. Desta forma cada comando SELECT poderá ser submetido a um banco de dados para execução.

Não houve necessidade de se criar um mecanismo complexo de *parse*, visto que o objetivo deste trabalho foi testar algoritmos de planejamento do uso dos bancos de dados (recursos).

A primeira etapa do processo é separar as colunas do comando SELECT. Depois são separadas as tabelas da cláusula FROM. As tabelas são separadas com e sem os apelidos, para facilitar a criação das buscas isoladas. Neste mesmo instante são acrescentados os nomes das tabelas temporárias que serão criadas antes de fornecer o resultado final da busca, após a execução paralela nos diferentes bancos de dados.

Na seqüência verifica-se a existência ou não da cláusula WHERE. Caso exista, procura-se inicialmente identificar e separar a(s) cláusula(s) de união das tabelas. Com isso é possível acrescentar outras cláusulas que não interfiram na união das tabelas, como comparações através de operadores relacionais e outras cláusulas como AND e OR. Para este processo, utiliza-se o método **isWhereAlias**.

Cumprida esta etapa, inicia-se a criação dos comandos SELECT já particionados com objetivo de se obter apenas um comando para cada tabela. Em paralelo a este processo, utiliza-se o comando para criar a tabela temporária que será utilizada para execução e transporte dos dados de cada comando SELECT. Caso no comando SELECT criado existam variáveis a serem substituídas (identificadas com o símbolo de interrogação), o comando recebe uma variável booleana para, antes da submissão da busca no banco de dados, realizar a substituição.

No final do método, obtém-se uma coleção com as tabelas, seus respectivos comandos de busca, existência ou não de parâmetros, o comando de criação da tabela temporária e, quando for o caso, a cláusula de união entre as tabelas envolvidas na busca.

6.1.3 Vinculação dos recursos com as tabelas

O método **checkDataResources** recebe como parâmetros as tabelas, os recursos e a URL que fazem parte da busca.

Para cada uma das tabelas envolvidas na busca, identificam-se quais bancos de dados a possui. Para isto é realizada uma busca simples no dicionário de dados de cada banco de dados. A última versão da OGSA-DAI possui algumas funções que retornam metadados dos bancos de dados disponíveis no ambiente, mas ainda com informações muito precárias. Ao consultar a documentação da versão, nota-se que há uma intenção do grupo para criar um mecanismo completo de extração das informações de metadados nos diversos bancos de dados. Enquanto estas funções não são disponibilizadas pelo ambiente, foi criado um método **tableMetadata** para realizar a busca das informações para este sistema no dicionário de dados de cada banco de dados utilizado. Este método pode ser facilmente adaptado para todos os demais bancos de dados que se utilize na Organização Virtual.

Ao final do método, uma coleção será retornada com cada uma das tabelas que fazem parte da busca e com os respectivos recursos que permitem sua localização.

6.1.4 Verificação de Métricas dos recursos

O método **getMetrics** recebe a relação de recursos que podem fazer parte da busca e a URL do ambiente.

As principais métricas que podem ser extraídas de um banco de dados, conforme mencionado anteriormente, são: memória disponível (volátil e não volátil), velocidade e disponibilidade do processador e velocidade de rede. Cada banco de dados possui mecanismos próprios para apurar estas métricas. Como a opção deste trabalho foi utilizar o próprio banco de dados para extrair as métricas e não utilizar métricas da Grid Computacional (como MDS) por se acreditar que cada gerenciador dispõe de condições mais eficazes para extrair esta informação, criou-se uma tabela em cada um dos gerenciadores de banco de dados para armazenar e facilitar a busca das métricas. A estrutura desta tabela está no Anexo A deste trabalho.

Com base nisso, atribui-se um valor de referência, o maior valor possível (quando for o caso) e um peso específico. Caso o valor de referência já seja um índice, como percentual de uso do processador, não há necessidade de especificar o maior valor possível, visto que este será 100%. Contudo, a banda de rede pode ter um valor de referência e ter um valor limite (largura da banda). Este valor de referência será transformado em um índice através da fórmula:

$$1 - (x/y)$$

onde: **x** é o valor de referência

y é o valor limite.

Ao se atribuir um peso a cada um dos índices, pode-se identificar uma métrica específica de cada recurso disponível na Grid. Este índice é calculado através do método **metric**.

A vantagem de se utilizar uma tabela para armazenar a métrica é a possibilidade de utilizar recursos do próprio banco de dados, como a criação de gatilhos, para automatizar o processo de atualização dos dados do servidor. Isso faz com que estes dados reflitam a disponibilidade do banco de dados com maior precisão. Outra vantagem é a possibilidade de se utilizar métricas adequadas a cada tipo de recurso, sem forçar uma padronização que poderia desvirtuar a escolha de um recurso em detrimento de outro. Por exemplo, caso o uso de memória seja mais eficiente em um banco de dados do que em outro, ou haja mecanismos de agilização de rede, pode-se atribuir um peso maior para um recurso em particular.

Esta idéia contribui com o conceito de Organização Virtual, visto que cada membro da comunidade é responsável pela disponibilização e organização dos seus próprios recursos.

No final do processo, tem-se um índice que indica a métrica de cada recurso que pode ser utilizado na busca.

6.1.5 Seleção dos recursos que executarão as buscas

Com base nas informações capturadas nos processos anteriores é possível realizar a seleção dos recursos mais adequados para realizar as buscas particionadas. No final deste processo tem-se o plano de execução da busca.

Deve-se notar que o algoritmo prevê a utilização de todos os recursos envolvidos na busca. Com isso, a métrica será importante para determinar quais bancos de dados serão priorizados na submissão das buscas. Caso o número de comandos SELECTs retornados depois do paticionamento (consultas) seja inferior ao número de bancos de dados disponíveis, somente os bancos com maior métrica serão utilizados. Caso o número de consultas seja superior ao número de bancos de dados, todos serão utilizados, sem importar a disponibilidade (métrica) do banco de dados.

6.1.5.1 Plano de Ordem Parcial (POP)

Como a busca está separada na menor unidade possível de busca (uma única tabela), inicia-se o processo de planejamento com a utilização do conceito de Ordem Parcial. Desta forma é possível paralelizar as ações que não possuem dependência entre elas. Isto é o que acontece quando se submete cada uma das buscas em diferentes bancos de dados. Ao final das

consultas e com o resultado das linhas recuperadas, é possível unir as ações isoladas para compor o resultado final.

O Plano de Execução será armazenado em uma coleção com três colunas: o recurso, o comando e a tabela temporária que deverá ser criada. Parte-se da coleção com os recursos que podem atender às buscas. Para cada recurso envolvido na busca onde é possível se executar o mesmo comando, verifica-se qual o recurso com maior disponibilidade (métrica). O módulo evita que se concentre todas as buscas em um único recurso (o de maior disponibilidade) ao forçar a troca do recurso utilizado a cada comando. Desta forma, o número mínimo que será obtido na paralelização da busca será o mesmo número de servidores de banco de dados disponíveis. Assim, caso haja uma busca com duas tabelas e apenas dois servidores, cada servidor realizará uma busca, independente da disponibilidade (métrica). Quando há mais buscas do que recursos disponíveis, o algoritmo tende a indicar a busca no recurso que tenha maior disponibilidade. Quando houver mais recursos do que buscas, o sistema irá priorizar os que tiverem melhores métricas.

A figura 6.3 demonstra o algoritmo utilizado.

```
mDistrib= checkDataResources()
mMetric = getMetrics()
Para cada mDistrib
 Se Busca Diferente
     Atualiza Recursos Utilizados
 Fim-Se
 Para cada mMetric/Recurso
  Se (Métrica do Recurso é melhor) e
     (Recurso não Utilizado)
     mSubmit <- Recurso
  Fim-Se
 Fim-Para
Fim-Enquanto
```

Figura 6.3: Algoritmo POP para elaborar o Plano de Execução

Baseado no Plano de Execução, o primeiro algoritmo proposto submete as buscas com base nos recursos disponíveis. A figura 6.4 demonstra o algoritmo de submissão.

```
Baseado no Plano de Execução
Para cada Recurso
 Baseado no Plano de Execução
 Para cada Busca
  Submete Busca
  Inicia Thread
 Fim-Para
Fim-Para
Enquanto Threads em Andamento
Fim-Enquanto
```

Figura 6.4: Algoritmo para Submissão

Com a seqüência de recursos e comandos para execução, o módulo submete as buscas a cada um dos recursos através de uma atividade de *Bulk Load* presente no *middleware*. Foi escolhida esta opção, pois é necessário criar tabelas temporárias para realizar a união das linhas retornadas em cada um dos comandos submetidos. Esta união acontecerá no recurso mais disponível (melhor métrica). O processo de *bulk load* assume a responsabilidade de executar a busca e alimentar as tabelas temporárias. Para isso utilizam-se os métodos **DTOutputStream** e **performBulkLoad**.

Antes de submeter as buscas, eventuais parâmetros são associados aos comandos para que estes possam filtrar as linhas que deverão ser retornadas.

No final deste processo as tabelas temporárias estão alimentadas com o resultado das buscas em cada um dos gerenciadores de banco de dados.

6.1.5.2 Plano de Ordem Parcial com Restrição de Recursos e Monitoramento

Partindo do Plano de Ordem Parcial já estabelecido, muda-se o foco da atribuição para as buscas e não para os recursos. Com isso enquanto houver recursos disponíveis, submete-se as buscas em paralelo. Quando o número de recursos se esgotar, o módulo aguarda a liberação de qualquer recurso para então submeter ao recurso disponível. Isso implica, na maior parte das vezes, em uma mudança no plano original estabelecido.

O motivo da mudança no plano é que a submissão original obedece ao critério da disponibilidade do recurso (métrica) o que não quer dizer que o recurso mais disponível irá encerrar a consulta mais rapidamente. Outros fatores influenciam o tempo de duração da busca: volume de dados analisados, transferência de dados entre os bancos de dados, etc.

A figura 6.5 demonstra o algoritmo utilizado.

```
Baseado no Plano de Execução
Enquanto houver Busca no Plano
 Para cada nó disponível
  Monta Busca para recurso disponível
  Submete Busca
  Inicia Thread
  Retira Busca do Plano de Execução
  Indica nó como não disponível
 Fim-Para
 Enquanto Threads em Andamento
 Fim-Enquanto
 Indica nó como disponível
Fim-Enquanto
```

Figura 6.5: Submissão de buscas com Algoritmo POP com Restrição de Recursos e Monitoramento

A forma de acompanhamento da submissão, criação de tabelas temporárias, união de resultados são realizados da mesma forma que no algoritmo anterior.

6.1.6 União das buscas e apresentação do resultado

Com as tabelas temporárias criadas e disponíveis no recurso de melhor métrica, o comando completo definido pelo usuário no início do processo é submetido. O resultado da busca fica disponibilizado para tratamento pelo ambiente ou pelo usuário final.

No final do processo as tabelas temporárias são excluídas do gerenciador de banco de dados.

6.2 Bases de Dados utilizadas

As bases de dados de testes foram:

- Base de dados exemplo que vem com o *middleware*
- Base de dados relativamente complexa de uma seqüência de DNA com quatro tabelas

6.2.1 Base de dados Exemplo do OGSA-DAI

Inicialmente composto por uma única tabela LITTLEBLACKBOOK com um milhão de linhas, acrescentou-se uma segunda tabela DEPEND com cerca de quinhentas linhas. Ambas tabelas continham uma coluna em comum (ID).

As duas tabelas estavam disponíveis no banco de dados Oracle e somente a tabela LITTLEBLACKBOOK estava disponível no MySQL.

Por não conter dados complexos e não haver um rigor de exigência na qualidade dos dados extraídos, esta base foi utilizada na fase de testes iniciais [OGSA-DAI, 2006].

As estruturas das tabelas estão no Anexo B deste trabalho.

6.2.2 Bio-informática

A escolha deste modelo deveu-se à complexidade e volume dos dados armazenados em uma seqüência de DNAs. Há um grande número de atributos envolvidos em cada uma das tabelas o que representa um desafio de análise do resultado dos dados, conforme pode ser observado no Anexo B deste trabalho.

Os dados foram extraídos do trabalho de Shipp [Shipp et al, 2002] e podem ser obtidos através de um arquivo-texto e uma planilha Excel na Internet. O arquivo-texto contém valores de expressões genéticas. São 7.129 atributos e 58 modelos. Como há muitos atributos, estes são representados como linhas e os modelos como colunas. Na planilha Excel há informações clínicas dos modelos de Linfoma utilizados no estudo.

Há três tabelas básicas: LYMPH_OUTCOME, LYMPH_EXP e LYMPH_MAP. A primeira armazena o resultado de dados clínicos. Um resultado positivo indicado que houve cura e o negativo que não houve. Nesta tabela há 58 linhas.

A segunda tabela armazena valores genéticos de expressão para cada modelo e tem origem no processamento das seqüências de DNA. Nesta tabela há 7.129 linhas.

A terceira armazena as descrições da identificação dos genes criadas na importação dos dados. Nesta tabela há 7.129 linhas.

A quarta tabela do modelo (LYMPH_TEMP) é a que possui o maior número de linhas (413.482) e é resultado do cruzamento dos dados das tabelas anteriores para armazenar os dados originais da expressão genética.

Como há ligação entre cada uma das tabelas, esta base de dados foi escolhida por permitir que se possa testar as buscas com duas, três ou quatro tabelas simultaneamente.

Parte III: Resultados

7 Testes Realizados

Instalou-se o *middleware* OGSA-DAI versão WSI-2.2 no TomCat 5.0.28.

A infra-estrutura de instalação foi: um servidor Pentium IV, 3.2 Ghz HT, com 1 GB de memória RAM e 80 GB de disco rígido. O sistema operacional é Windows XP Professional, *service pack* 2. Inicialmente foi utilizado um sistema gerenciador de banco de dados Oracle 10*g* (10.0.2) e o sistema gerenciador de banco de dados MySQL 5.0. Um segundo servidor, Pentim IV, 1.1 Ghz, com 750MB de memória RAM e 80 GB de disco rígido e com Windows XP Professional instalado foi utilizado. Neste servidor foi instalado um banco de dados Oracle 10*g* (10.2.0). Todos foram instalados no mesmo servidor e disponibilizados ao *middleware* através dos mecanismos especificados no ambiente.

O roteiro de instalação do ambiente consta no Anexo C deste trabalho.

Para descrever os resultados alcançados com os testes, as duas fases de teste foram separadas de acordo com a base de dados utilizada, quais sejam:

- Base de Dados OGSA-DAI
- Base de Dados Bio-informática

A métrica dos recursos, para efeitos de testes, foi estabelecida com os valores apresentados nas tabelas 6.5, 6.6 e 6.7:

Servidor 1:

Oracle (orclCPoderoso):

METRIC_ID	METRIC_NAME	METRIC_DESC	REFVALUE	MAXVALUE	MATCHVALUE
1	Network Traffic Volume Per Sec	Bytes Per Second	123,35	10000	,2
2	Physical Read Total IO Requests Per Sec	Requests Per Second	0	10000	,1
3	Physical Write Total IO Requests Per Sec	Requests Per Second	1,36666667	10000	,1
4	Database CPU Time Ratio	% Cpu/DB_Time	,964314625		,3
5	Shared Pool Free %	% Free/Total	,625472534		,3

Figura 6.6:Métrica do banco de dados orclCPoderoso.

Como resultado, a métrica calculada para este recurso foi 0.8744554811491747.

MySQL (MySQLCPoderoso):

Resultset 1

metric_id	metric_name	metric_desc	refvalue
1	Network Traffic Volume Per Sec	Bytes Per Second	66.4400
2	Physical Read Total IO Requests Per Sec	Requests Per Second	0.0000
3	Physical Write Total IO Requests Per Sec	Requests Per Second	3.8860
4	Database CPU Time Ratio	% Cpu/DB_Time	0.9700
5	Shared Pool Free %	% Free/Total	0.9700

Figura 6.7:Métrica do banco de dados MySQLCPoderoso.

Como resultado, a métrica calculada para este recurso foi 0.9806323399999999.

Servidor 2:

Oracle (orclFamilia):

METRIC_ID	METRIC_NAME	METRIC_DESC	REFVALUE	MAXVALUE	MATCHVALUE
1	Network Traffic Volume Per Sec	Bytes Per Second	18,9	10000	,2
2	Physical Read Total IO Requests Per Sec	Requests Per Second	105	10000	,1
3	Physical Write Total IO Requests Per Sec	Requests Per Second	15,525	10000	,1
4	Database CPU Time Ratio	% Cpu/DB_Time	,987651		,3
5	Shared Pool Free %	% Free/Total	,586545		,3

Figura 6.8:Métrica do banco de dados orclFamilia.

Como resultado, a métrica calculada para este recurso foi 0.8706755500000001.

7.1 Base de dados OGSA-DAI

O ambiente de teste limitou-se aos servidores onde foi instalado o OGSA-DAI , de acordo com a descrição anterior.

Após a execução do sistema com a utilização dos dados desta base, os resultados foram obtidos a partir das seguintes buscas:

i) `select a.id, a.name from littleblackbook a`

ii) `select a.id, a.name from littleblackbook a where a.id>=? and a.id<?`

iii) `select a.id, a.name, b.lbb_id, b.name from littleblackbook a, depend b where a.id = b.lbb_id and a.id>=? and a.id<?`

A primeira consulta teve o objetivo de checar o retorno das linhas de dados, sem haver preocupação com a paralelização ou distribuição da consulta. O resultado obtido foi de acordo com o esperado, visto que as linhas deveriam ser retornadas como uma busca usual em bancos de dados.

A segunda consulta procurou incluir e testar a submissão de comandos de busca com parâmetros. O padrão OGSA-DAI utiliza o símbolo de interrogação para indicar quais parâmetros deverá ser submetido ao banco de dados. Mais uma vez, os resultados estiveram dentro do esperado, visto que o ambiente conseguiu identificar o momento da substituição dos parâmetros e retornou somente as linhas dentro do intervalo especificado.

A terceira consulta, por envolver duas tabelas, permitiu observar o comportamento do módulo frente a uma situação de paralelização de busca. No primeiro teste desabilitou-se um dos dois bancos de dados disponíveis na grid. Com isso o escalonador não teve alternativa senão utilizar o único banco de dados disponível. O sistema não foi capaz, contudo, de

submeter a consulta como era originalmente. A busca foi dividida em dois comandos SELECT, um para cada tabela, e submeteu ambas ao mesmo banco de dados.

7.1.1 Resultado do Planejamento na terceira consulta

Este teste possibilitou analisar o desempenho do algoritmo de planejamento. Neste teste foi habilitado o segundo banco de dados, portanto a busca poderia ser submetida aos dois recursos disponíveis na grid. Em uma primeira etapa a tabela LITTLEBLACKBOOK estava presente em ambos bancos de dados e a tabela DEPEND estava somente no banco de dados Oracle.

Durante a execução da busca, o planejador utilizou para o primeiro comando de busca o MySQL, por possuir melhor métrica, e o Oracle para o segundo comando.

Quando se acrescentou o terceiro recurso, não houve modificação no plano de submissão, visto que o terceiro recurso possui uma métrica inferior às dos demais recursos.

Foi possível constatar que o módulo buscou o tabela existente no MySQL, mesmo quando a métrica não era favorável. Isso se deve ao fato de que o algoritmo procura dividir a busca entre diversos recursos (maximização do uso dos recursos através da paralelização da busca). Como uma das tabelas não estava presente em todos os recursos, esta tabela teria que ser localizada no outro gerenciador de banco de dados. A busca da única tabela presente nos dois bancos de dados era submetida ao MySQL e a outra no Oracle.

Ao se colocar a tabela DEPEND em ambos os recursos, o resultado do planejamento priorizou o MySQL para a primeira busca e em seguida o Oracle, por ser o servidor com menor disponibilidade.

7.2 Base de dados Bio-informática

O ambiente de teste também limitou-se aos servidores onde foi instalado o OGSA-DAI e os bancos de dados descritos anteriormente.

As buscas utilizadas para os testes foram:

iv) `select a.gene_id, a.accession, b.gene_id, b.sample_id from lymph_map a, lymph_exp b where a.gene_id = b.gene_id`

v) `select a.gene_id, a.accession, b.gene_id, b.sample_id, c.sample_id, c.status from lymph_map a, lymph_exp b, lymph_outcome c where a.gene_id = b.gene_id and b.sample_id = c.sample_id`

vi) `select b.gene_id, b.sample_id, c.sample_id, c.status from lymph_exp b, lymph_outcome c where b.sample_id = c.sample_id`

vii) `select a.gene_id, a.accession, d.gene_id, d.dlbc1 from lymph_map a, lymph_temp d where a.gene_id = d.gene_id`

viii) `select a.gene_id, a.accession, b.gene_id, b.sample_id, d.dlbc1 from lymph_map a, lymph_exp b, lymph_temp d where a.gene_id = b.gene_id and a.gene_id = d.gene_id`

ix) `select a.gene_id, a.accession, b.gene_id, b.sample_id, c.sample_id, c.status, d.dlbc1 from lymph_map a, lymph_exp b, lymph_outcome c, lymph_temp d where a.gene_id = b.gene_id and b.sample_id = c.sample_id and a.gene_id = d.gene_id`

Todas as buscas tiveram o objetivo de verificar o comportamento do planejamento que foi realizado pelo módulo. Para isso utilizou-se apenas consultas que envolvessem união regular de tabelas (*joins*). Os testes dividiram-se em duas etapas:

- Dois recursos habilitados (banco de dados Oracle e MySQL do servidor principal)
- Três recursos habilitados (além dos anteriores o banco de dados Oracle do outro servidor)

7.2.1 Resultado do Planejamento com dois recursos em POP

Como se pode notar, os planejador se comportou conforme esperado, visto que sempre que houve o mesmo número de recursos e de buscas, todos foram utilizados. Sempre que havia mais buscas do que recursos, aquele que possuía melhor métrica foi utilizado no Plano.

Todos os planos foram submetidos em paralelo em cada recurso disponível e os tempos utilizados são os fornecidos pela própria linguagem durante a execução do código.

Para a busca (i), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.ACCESSION FROM LYMPH_MAP A
orclCPoderoso SELECT B.GENE_ID,B.SAMPLE_ID FROM LYMPH_EXP B
Início: 12:22:46
Executando Query em Paralelo
Executando (MySQLCPoderoso)
Executando (orclCPoderoso)
Executada Query em Paralelo
Resposta: COMPLETO
Término: 12:32:56
```

Para a busca (ii), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.ACCESSION FROM LYMPH_MAP A
orclCPoderoso SELECT B.GENE_ID,B.SAMPLE_ID FROM LYMPH_EXP B
```

```
MySQLCPoderoso SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
Início: 12:09:43
Executando Query em Paralelo
Executando (MySQLCPoderoso)
Executando (orclCPoderoso)
Executando (MySQLCPoderoso)
Executada Query em Paralelo
Resposta: COMPLETO
Término: 12:19:45
```

Para a busca (iii), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT B.GENE_ID,B.SAMPLE_ID FROM LYMPH_EXP B
orclCPoderoso SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
Início: 12:45:11
Executando Query em Paralelo
Executando (MySQLCPoderoso)
Executando (orclCPoderoso)
Executada Query em Paralelo
Resposta: COMPLETO
Término: 12:55:18
```

Para a busca (iv), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.ACCESSION FROM LYMPH_MAP A
orclCPoderoso SELECT D.GENE_ID,D.DLBC1 FROM LYMPH_TEMP D
Início: 13:43:53
Executando Query em Paralelo
Executando (MySQLCPoderoso)
Executando (orclCPoderoso)
Executada Query em Paralelo
Resposta: COMPLETO
Término: 13:44:14
```

Para a busca (v), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.SAMPLE_ID FROM LYMPH_EXP A
orclCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
MySQLCPoderoso SELECT D.DLBC1,D.GENE_ID FROM LYMPH_TEMP D
Início: 14:46:06
```

```
Executando Query em Paralelo
Executando (orclCPoderoso)
Executando (MySQLCPoderoso)
Executando (MySQLCPoderoso)
Executada Query em Paralelo
Resposta: COMPLETO
Término: 14:56:14
```

Para a busca (vi), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.ACCESSION FROM LYMPH_MAP A
orclCPoderoso SELECT B.GENE_ID,B.SAMPLE_ID FROM LYMPH_EXP B
MySQLCPoderoso SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
orclCPoderoso SELECT D.DLBC1 FROM LYMPH_TEMP D
Início: 15:03:38
Executando Query em Paralelo
Executando (orclCPoderoso)
Executando (orclCPoderoso)
Executando (MySQLCPoderoso)
Executando (MySQLCPoderoso)
Executada Query em Paralelo
Resposta: COMPLETO
Término: 15:13:50
```

7.2.2 Resultado do Planejamento com dois recursos em POP com Restrição de Recursos e Monitoramento

O algoritmo se comportou como o algoritmo anterior, mas o Plano de execução não foi submetido imediatamente. Optou-se por realizar o teste com duas tabelas uma única vez, visto que não acrescentaria muito para esta pesquisa. Como se pode constatar no tempo de execução da busca (i), o tempo de execução foi praticamente o mesmo. Isso se deve ao fato de a busca ser realizada em paralelo em dois diferentes bancos de dados em qualquer uma das situações.

Como foi realizado o monitoramento dos recursos, pode-se notar que nas buscas (ii), (v) e (vi) os resultados foram consideravelmente menores. Nota-se que o Plano de Execução sofreu alterações durante o processamento das consultas (v) e (vi). Isso se deveu ao planejador inicialmente ter a informação somente sobre quais são os recursos disponíveis e suas respectivas métricas. Conforme o recurso encerrava seu trabalho, ele estava disponível para realizar nova busca, independente da métrica do recurso.

Outro fato observador foi que, quanto maior o volume de linhas processadas pelo recurso e quanto mais cedo a busca for submetida, maior o ganho no tempo de processamento. Isso pode ser notado na busca (v) em relação à busca (vi).

Todos os planos foram submetidos em paralelo em cada recurso disponível e os tempos utilizados são os fornecidos pela própria linguagem durante a execução do código.

Para a busca (i), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
orclCPoderoso SELECT D.GENE_ID,D.DLBC1 FROM LYMPH_TEMP D
Início: 15:30:34
Executando MySQLCPoderoso
Executando orclCPoderoso
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 15:40:43
```

Para a busca (ii), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.SAMPLE_ID FROM LYMPH_EXP A
orclCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
MySQLCPoderoso SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
Término: 16:29:21
Executando orclCPoderoso( LYMPH_EXP )
Executando MySQLCPoderoso( LYMPH_MAP )
Encerrei recurso MySQLCPoderoso
Executando MySQLCPoderoso( LYMPH_OUTCOME )
Encerrei recurso MySQLCPoderoso
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 16:39:01
```

Para a busca (v), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
orclCPoderoso SELECT A.GENE_ID,A.SAMPLE_ID FROM LYMPH_EXP A
MySQLCPoderoso SELECT D.DLBC1,D.GENE_ID FROM LYMPH_TEMP D
Início: 15:55:08
```

```
Executando orclCPoderoso( LYMPH_EXP )
Executando MySQLCPoderoso( LYMPH_MAP )
Encerrei recurso MySQLCPoderoso
Executando MySQLCPoderoso( LYMPH_TEMP )
Encerrei recurso MySQLCPoderoso
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 16:04:06
```

Para a busca (vi), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT B.GENE_ID,B.SAMPLE_ID FROM LYMPH_EXP B
orclCPoderoso SELECT A.GENE_ID,A.ACCESSION FROM LYMPH_MAP A
MySQLCPoderoso SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
orclCPoderoso SELECT D.DLBC1 FROM LYMPH_TEMP D
Início: 16:17:53
Executando orclCPoderoso( LYMPH_MAP )
Executando MySQLCPoderoso( LYMPH_EXP )
Encerrei recurso orclCPoderoso
Executando orclCPoderoso( LYMPH_OUTCOME )
Encerrei recurso orclCPoderoso
Executando orclCPoderoso( LYMPH_TEMP )
Encerrei recurso orclCPoderoso
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 16:27:37
```

7.2.3 Resultado do Planejamento com três recursos em POP com Restrição de Recursos e Monitoramento

O algoritmo se comportou como esperado quando se adicionou mais um recurso. Algumas considerações importantes dizem respeito à não utilização do recurso de menor métrica quando havia menos buscas do que recursos (i). Por outro lado, utilizou todos os recursos quando havia três consultas (ii) e (v). Para estas buscas não houve um ganho significativo no desempenho porque havia uma grande disparidade entre o número de linhas das tabelas. Como há uma grade tabela e as outras são menores, o tempo total da busca gira em torno do tempo da maior busca.

Contudo, para a consulta (vi) pode-se notar que houve um ganho ao se esperar pelo recurso mais disponível para submeter a última busca. Neste mesmo teste pode-se notar uma alteração no plano de execução que normalmente não seria o mais provável. Como o recurso com menor métrica encerrou o trabalho mais rapidamente, ele recebeu a responsabilidade de processar as linhas da última tabela da busca.

Para a busca (i), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.SAMPLE_ID FROM LYMPH_EXP A
orclCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
Início: 13:42:37
Executando MySQLCPoderoso( LYMPH_EXP )
Executando orclCPoderoso( LYMPH_MAP )
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 13:52:45
```

Para a busca (ii), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.SAMPLE_ID FROM LYMPH_EXP A
orclCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
orclFamilia SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
Início: 13:59:18
Executando MySQLCPoderoso( LYMPH_EXP )
Executando orclCPoderoso( LYMPH_MAP )
Executando orclFamilia( LYMPH_OUTCOME )
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 14:08:56
```

Para a busca (v), os resultados foram:

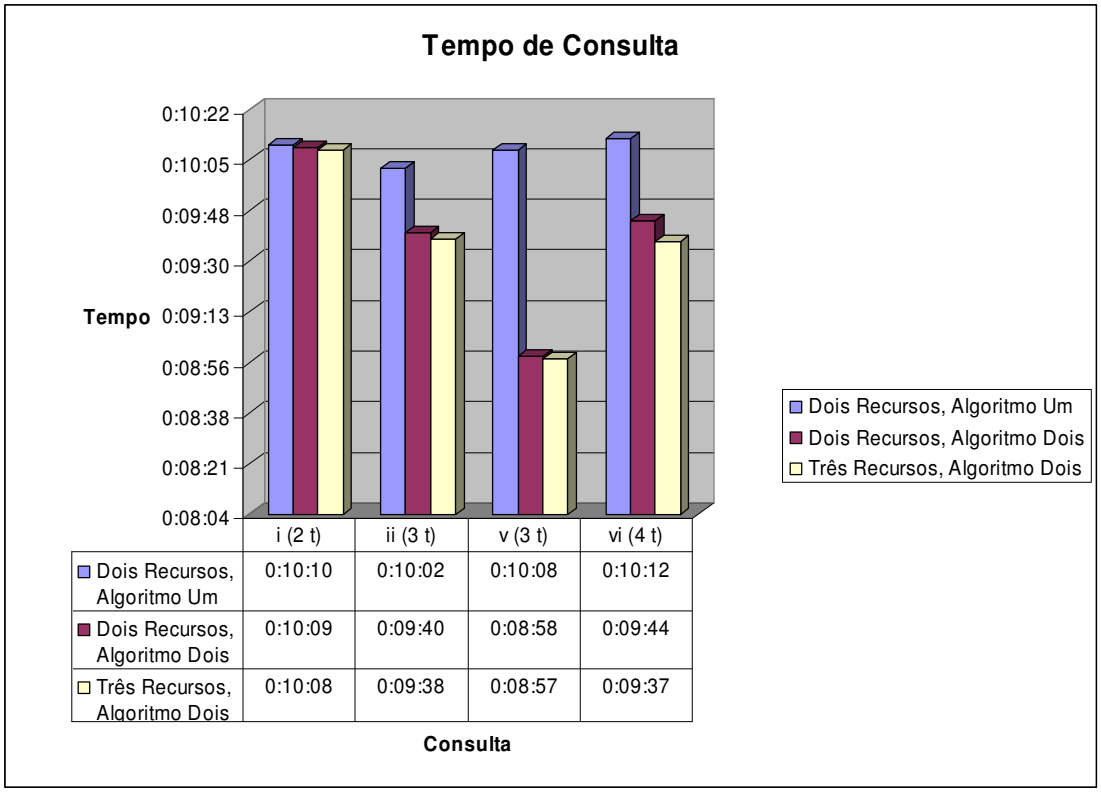
```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.SAMPLE_ID FROM LYMPH_EXP A
orclCPoderoso SELECT B.GENE_ID,B.ACCESSION FROM LYMPH_MAP B
orclFamilia SELECT D.DLBC1,D.GENE_ID FROM LYMPH_TEMP D
Início: 18:14:20
Executando MySQLCPoderoso( LYMPH_EXP )
Executando orclCPoderoso( LYMPH_MAP )
```

```
Executando orclFamilia( LYMPH_TEMP )
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 18:23:17
```

Para a busca (vi), os resultados foram:

```
Plano Original:
MySQLCPoderoso SELECT A.GENE_ID,A.ACCESSION FROM LYMPH_MAP A
orclCPoderoso SELECT B.GENE_ID,B.SAMPLE_ID FROM LYMPH_EXP B
orclFamilia SELECT C.SAMPLE_ID,C.STATUS FROM LYMPH_OUTCOME C
MySQLCPoderoso SELECT D.DLBC1 FROM LYMPH_TEMP D
Início: 18:44:09
Executando MySQLCPoderoso( LYMPH_MAP )
Executando orclCPoderoso( LYMPH_EXP )
Executando orclFamilia( LYMPH_OUTCOME )
Encerrei orclFamilia
Executando orclFamilia( LYMPH_TEMP )
Executando Query em Paralelo
Executada Query em Paralelo
Resposta: COMPLETO
Término: 18:53:46
```

Os resultados comparativos das buscas, utilizando dois recursos com o primeiro algoritmo e dois e três recursos com o segundo algoritmo estão comparados na figura 7.1.



	i (2 t)	ii (3 t)	v (3 t)	vi (4 t)
Dois Recursos, Algoritmo Um	0:10:10	0:10:02	0:10:08	0:10:12
Dois Recursos, Algoritmo Dois	0:10:09	0:09:40	0:08:58	0:09:44
Três Recursos, Algoritmo Dois	0:10:08	0:09:38	0:08:57	0:09:37

Figura 7.1: Comparativo da Implementação dos Algoritmos

Fica claro que o segundo algoritmo possui um desempenho melhor em todas as situações e apresenta melhores resultados conforme aumentam o número de recursos envolvidos na busca. Isto pode ser comprovado pela figura 7.2.

Na figura 7.2 pode-se notar que houve um ganho percentual em todas as consultas analisadas quando se comparou a utilização de três recursos com o segundo algoritmo com dois recursos no primeiro algoritmo. O ganho não foi maior na busca com quatro tabelas porque a quarta tabela da busca possui poucas linhas (vi).

Na consulta (v) as tabelas pesquisadas foram as que possuem maior número de linhas, isso fez com que o ganho fosse maior do que na busca (ii), mesmo com o mesmo número de tabelas.

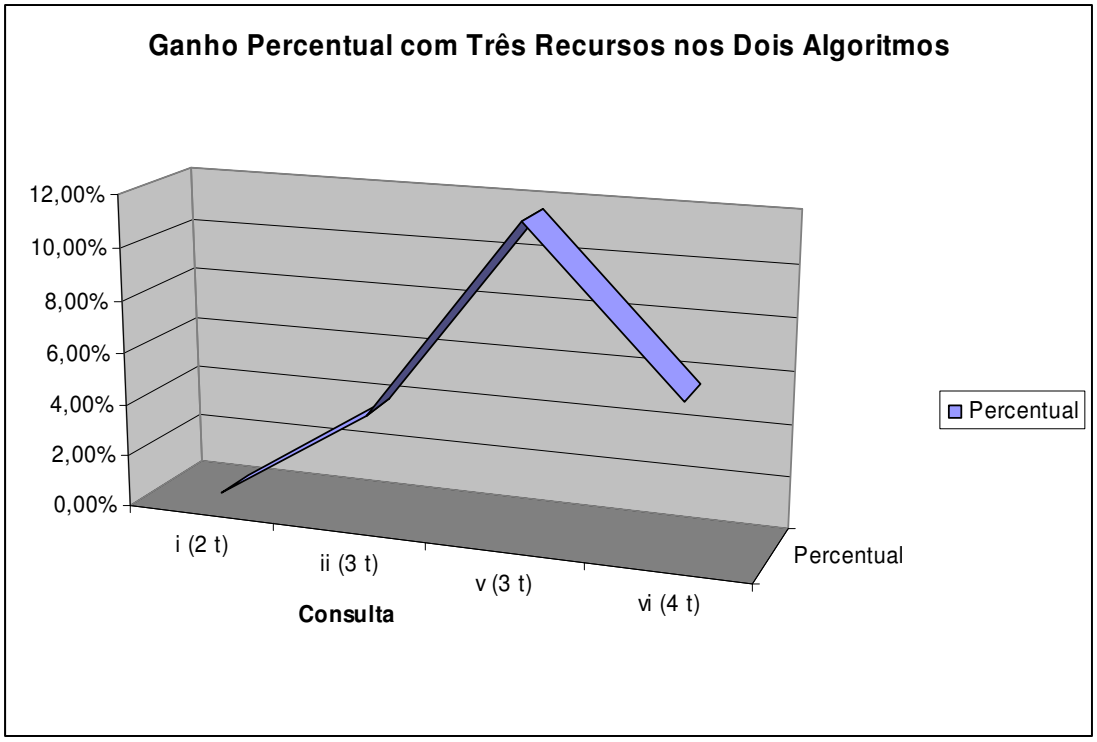


Figura 7.2: Ganho percentual entre as buscas com três recursos no segundo algoritmo e dois recursos no primeiro algoritmo

8 Conclusão

O trabalho apresentou a comparação entre os algoritmos propostos e demonstrou a viabilidade do uso de qualquer um dos dois, com uma tendência a utilização do segundo algoritmo para sistemas que utilizem um maior número de buscas e recursos.

Os testes mostraram que as Grids Computacionais são uma alternativa para otimizar o uso de recursos em um ambiente distribuído. Os Bancos de Dados, por serem importantes recursos computacionais, começam a serem utilizados, ainda que de maneira insipiente, na Grid.

Com novas pesquisas envolvendo Bancos de Dados e Grids, será possível melhorar o atendimento aos processos que envolvem estes recursos. Tanto o meio acadêmico quanto o empresarial poderá se beneficiar da união destas tecnologias.

Atualmente é possível utilizar o OGSA-DAI, *middleware* desenvolvido para prover os serviços de banco de dados, que funciona agregado ao Globus, o principal produto de Grid Computacional. Apesar dos esforços para atender às demandas, ainda há muitas atividades de banco de dados que não são realizadas. Isso abre um campo enorme para aprimorar o produto e realizar novas pesquisas. O OGSA-DQP é um destes produtos. Ele permite realizar buscas particionadas em banco de dados com a utilização do OGSA-DAI.

Contudo há diversas limitações no ambiente proposto. Novas práticas puderam ser testadas neste trabalho e mostraram que se pode utilizar a mesma infra-estrutura do OGSA-DAI, com pequenas alterações, e alcançar resultados satisfatórios.

O objetivo deste trabalho foi o de criar um sistema que interceptasse as requisições do usuário e, após verificar quais recursos há disponíveis no ambiente, subdividir as buscas complexas em buscas particionadas, mais simples. O plano de execução deveria prever a paralelização de cada uma das buscas particionadas.

Foi necessário criar um interpretador de comandos SELECT (*parse*) para realizar o particionamento da busca. Como a arquitetura utilizada na Grid é Orientada a Serviços, pode-se utilizar qualquer outro particionador que atenda aos requisitos do módulo, sem prejuízo do algoritmo proposto.

Esta pesquisa demonstrou a aplicação de técnicas de planejamento que envolve o campo da Inteligência Artificial. O objetivo de utilizar a Inteligência Artificial foi achar a melhor solução possível dentro do espaço de estados disponível, neste caso representado pelos gerenciadores de bancos de dados. A partir dos conceitos de Planejamento de Ordem Parcial e depois abrindo mão da Restrição de Recursos e Monitoramento da Submissão do Plano de

Execução, pode-se implementar um algoritmo que maximiza o uso dos recursos disponíveis na Organização Virtual.

Este trabalho trouxe a contribuição de levantar algumas considerações sobre a utilização de métricas específicas para bancos de dados. As métricas são importantes heurísticas que podem ser utilizadas para tomar a melhor decisão no momento da criação do plano de execução. Foi dado um enfoque diferente do utilizado na Grid Computacional que é a utilização de um serviço de informações sobre o volume de processamento e disponibilidade de armazenamento (MDS). Como os bancos de dados possuem uma característica própria, esta característica foi explorada para propor uma alternativa ao MDS.

De qualquer forma, mesmo que se opte por utilizar o MDS, o algoritmo não perde suas características. A métrica utilizada pode ser substituída pelo MDS sem prejuízo do planejamento proposto.

Como foi utilizado o padrão da OGSA-DAI, pode-se utilizar todos os bancos de dados que estiverem disponíveis na Grid e não há limitação no uso de OR ou AND na cláusula WHERE.

Com o teste realizado nos dois algoritmos implementados, observou-se que a utilização do Plano de Ordem Parcial é adequada para realizar o plano de execução das buscas. Como em um ambiente de Grid há restrição no uso de recursos, o monitoramento mostrou-se satisfatório para melhorar o plano de execução criado inicialmente.

Com isso, o algoritmo não só maximizou o uso dos recursos como também implementou o plano da melhor maneira possível. Isso faz com que uma das premissas do planejamento em Grid fosse atendida: acompanhamento dos serviços submetidos ao ambiente.

9 Trabalhos Futuros

O presente trabalho sugere diversas opções de prosseguimento. A seguir lista-se uma série de elementos que podem ser explorados por pesquisadores:

- Criar um repositório para manter dados históricos de utilização dos recursos. Isso irá colaborar com a heurística do planejador e permitirá criar réplicas de dados para as informações mais solicitadas.
- Criar um planejador para manter um repositório com o resultado do planejamento.
- Criar um repositório do planejador para manter o histórico de utilização dos recursos. Desta forma será possível refinar o processo de seleção de recursos e ajustar o tempo de processamento das ações uma vez que é possível comparar o valor indicado pelo gerenciador de banco de dados com o tempo efetivamente despendido, conforme figura 8.1.

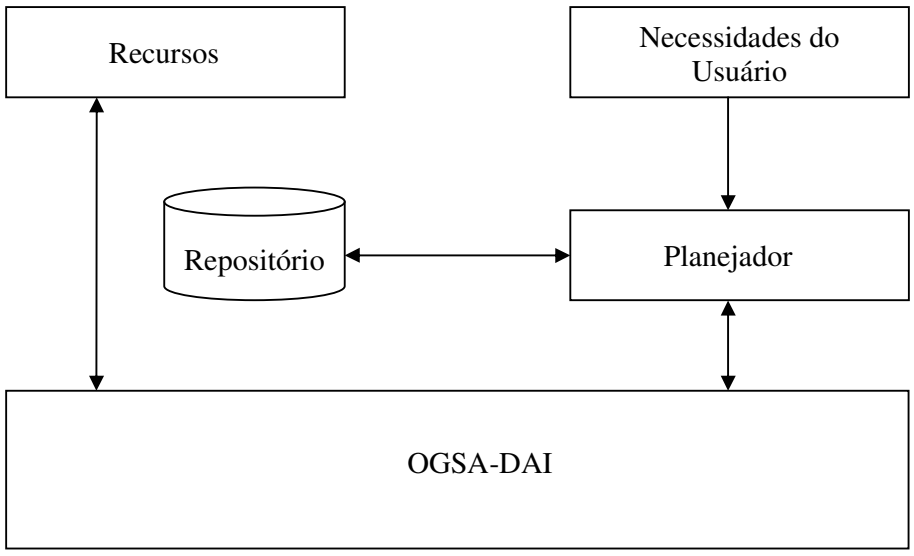


Figura 8.1: Proposta de Repositório de Planejamento

- Criar um metadados de todos os recursos disponíveis na OGSA-DAI.
- Permitir que se trabalhe com parte das linhas das tabelas replicadas.
- Estabelecer uma ontologia específica para descoberta e localização de gerenciadores de banco de dados.
- Aprimorar a heurística através da identificação da carga de trabalho atual e projeção da carga futura de utilização do banco de dados.

- Criar um agente de planejamento contínuo. Será necessário criar uma base de conhecimento e leitura constante do ambiente atual.
- Criar um sistema multiagente que permita agir de maneira cooperativa a partir do plano proposto.
- Estabelecer um Planejamento Condicional, onde se possa submeter as consultas a um recurso sem utilizar o artifício de criar tabelas temporárias. Desta forma, pode-se testar a hipótese de que, ao não se criar tabelas temporárias no recurso que será responsável pela união dos dados para apresentação final, a consulta será executada mais rapidamente. Deve ser possível checar a quantidade de consultas que podem obter este benefício, pois, visto que pelo menos uma tabela estará obrigatoriamente no recurso que unirá as consultas, pode ser viável submeter no mínimo duas tabelas a este recurso. Isso fará com que apenas a partir de três tabelas seja utilizado a grid computacional.

Com estas implementações, será possível realizar o planejamento de consultas de forma mais completa e próxima das necessidades da área acadêmica e profissional, além de complementar os serviços disponibilizados pelo OGSA-DAI.

10 Referências

[Alchemi, 2005] Alchemi. http://www.alchemi.net. Disponível em outubro de 2005.

[Alpdemir, 2003] Alpdemir, M. N.; Mukherjee, A.; Paton, N. W.; Watson, P.; Fernandes, A. A. A.; Gounaris, A.; Smith, J. Service-Based Distributed Querying on the Grid. : 2003.

[Alpdemir, 2003b] Alpdemir, M. N.; Mukherjee, A.; Paton, N. W.; Watson, P.; Fernandes, A. A. A.; Gounaris, A.; Smith, J.; Sakellariou, R.; Li, Peter. Using OGSA-DQP to Support Scientific Applications for the Grid. : 2003.

[Assunção et al, 2004] Assunção, M. D.; Koch, F. e Westphall, C. B.Building Complex Application Using Grid of Agents. Universidade Federal de Santa Catarina: 2004.

[Baker et al, 2005] Baker, M.; Apon, A.; Ferner, C. e Brown, J. Emerging Grid Standards. Computer, vol. 38, nº 4, pg. 43-50: 2005.

[Blythe et al, 2004] Blythe, J.; Deelman, E. e Gil, Y. "Planning and Metadata on the Computational Grid". Em: AAAI Spring Symposium on Semantic Web Services: 2004.

[Casavant e Kuhl, 1988] Casavant, T. L. e Kuhl, J. G. "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems" Em: IEEE Transactions on Software Engineering, v.14, n. 2, p. 141-154: 1988.

[Chede, 2005] Chede, C. T. Grid Computing: um novo paradigma computacional. Ed. Brasport. São Paulo: 2005.

[Cybok, 2004] Cybok, D. A Grid Workflow Infrastructure Em: Global Grid Forum 2004 Special Issue of Concurrency and Computation: Practice and Experience: 2004.

[Dantas, 2005] Dantas, M. "Computação Distribuída de Alto Desempenho". Ed. Axcel. Rio de Janeiro: 2005.

[Foster, 1999] Foster, I.; Kesselman, C. The Grid: Blueprint for a New Computing Infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishers: 1999.

[Foster, 2001] Foster, I. ; Kesselman, C. e Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, vol 15(3): 2001

[Foster, 2002] Foster, I. What is the Grid? A three point checklist. GRIDToday, Daily News And Information For the Global Grid Community: 2002.

[Foster, 2003] Foster, I.; Tuecke, S. e Unger, J. “OGSA Data Services”. Em: http://www.ggf.org: 2003. Acessado em julho de 2005.

[Freuder et al., 2005] Freuder, A. N.; Freuder, E. C.; Fourer, R.; Giunchiglia, E.; Goldman, R. P.; Kautz, H.; Rintanen, J. e Tate, A. Constraint Programming. Em: IEEE Intelligent Systems, março/abril de 2005 p. 62 a 72: 2005.

[Gil, 2004] Gil, Y. ; Deelman, E.; Blythe, J.; Kasselman, C. e Tangmunarunkit, H. Artificial Intelligence and Grids: Workflow Planning na Beyond. Em: IEEE Intelligent Systems vol 19, nº 1 janeiro/fevereiro de 2004 p. 26 a 33: 2004.

[Globus, 2005] Globus Toolkit. http://www.globus.org. Acessado em fevereiro de 2005.

[Gounaris et al, 2004] Gounaris, A.; Sakellariou, R.; Paton, Norman W.; Fernandes, Alvaro A. A. Resource Scheduling for Parallel Query Processing on Computational Grids. Em: 5th International Workshop on Grid Computing (GRID 2004). IEEE Computer Society: 2004.

[GrADS, 2006] GrADS Grid Analysis and Display System. Disponível em: http://grads.iges.org/grads/. Acessado em janeiro de 2006.

[GridBus, 2005] GridBus. http://www.gridbus.org. Acessado em outubro de 2005.

[IBM, 2005] IBM homepage. http://www.ibm.com. Acessado em fevereiro de 2005.

[Legion, 2005] Legion. http://www.cs.virginia.edu/~legion/. Acessado em outubro de 2005.

[OGSA, 2006] OGSA homepage. http://www.globus.org/ogsa/. Acessado em janeiro de 2006.

[OGSA-DAI, 2006] OGSA-DAI “OGSA Data Access and Integration”. Em: http://www.ogsadai.org.uk/. Acessado em janeiro de 2006.

[OGSA-DQP, 2006] OGSA-DQP “OGSA Database Query Processing”. Em http://www.ogsadai.org.uk/about/ogsa-dqp/. Acessado em janeiro de 2006.

OGSI. GGF (2003). Open Grid Services Infrastructure Version 1.0.

[Oliveira, 2002] Oliveira, Celso H. P. SQL Curso Prático. Ed. Novatec. São Paulo: 2002.

[Oliveira, 2004] Oliveira, Celso H. P. Tecnologia de Grid Computing utilizando Banco de Dados Oracle. IV CBCOMP. Itajaí: 2004.

[Oliveira e Almeida, 2005] Oliveira, Celso H. P. e Almeida, Maurício A. Padrão de Banco de Dados Virtual em Grade Computacional. 2º CONTECSI. São Paulo: 2005.

[Oracle, 2005] Oracle homepage. http://www.oracle.com. Acessado em fevereiro de 2005.

[Pernas e Dantas, 2004] Pernas, A. M. e Dantas, M. A. R. Ontologias Aplicadas à Descrição de Recursos em Ambientes de Grid. Revista Infocomp – Revista da Ciência da Computação, Volume 3, Número 2, pp 26-31: 2004.

[Ranganathan e Foster, 2003] Ranganathan, Kavitha e Foster, Ian Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. Em: Journal of Grid Computing, Volume 1 p. 53-62: 2003.

[Russel e Norvig 2004] Russell, Stuard e Norvig, Peter. Inteligência Artificial. Ed. Campus. Rio de Janeiro: 2004.

[Nieto-Santisteban et al, 2004] Nieto-Santisteban, M. A., Szalay, A., Thakar, A. R., O'Mullane, W. J. When Databases Systems Meet the Grid. Johns Hopkins University – Microsoft Research: 2004.

[Shipp et al, 2002] Shipp, M. A. et al. Diffuse Large B-Cell Lymphoma Outcome Prediction by Gene Expression Profiling and Supervised Machine Learning. Em: Nature Medicine, vol. 8, nº 1, pp 68-74: 2002.

[Smith, 2003] Smith, J.; Gounaris, A.; Watson, P.; Paton, N. W.; Fernandes, A. A. A.; Sakellariou, R. Distributed Query Processing on the Grid. Em: The International Journal of High Performance Computing Applications, vol. 17, nº 4, pp 353-367: 2003.

[SRB, 2005] SRB homepage. http://www.sdsc.edu/srb/. Acessado em fevereiro de 2005.

[UDDI, 2005] UDDI. http://www.uddi.org. Acessado em outubro de 2005.

[Wang et al, 2005] Wang, F.; Helian, N. Cluster Computing and Grid 2005 Works in Progress: Grid-Oriented Storage. IEEE Distributed Systems Online, vol. 6, nº 9: 2005

[Watson, 2002] Watson, P.; Paton, N.; Atkinson, M.; Dialani, V.; Pearson, D. e Storey, T. Database Access and Integration Services on the Grid. Em: Fourth Global Grid Forum (GGF-4): 2002. http://www.nesc.ac.uk/technical_papers/dbtf.pdf. Acessado em julho de 2005.

[Watson, 2003] Watson, P. Database and the Grid. Em: Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Inc: 2003.

Glossário

Alchemi: é um *framework* que permite criar um ambiente de grid.

API: *Application Program Interface* é um conjunto de rotinas, protocolos e ferramentas para desenvolvimento de sistemas.

CBO: *Cost-Based Optimizer* é uma forma de otimização de buscas presente em alguns SGBDs. Como a otimização é realizada com base no custo da busca, é possível obter resultados melhores do que quando se utiliza a otimização baseada em regras.

Clustering: é o processo de distribuição de tarefas entre servidores em um ambiente distribuído com restrição a um único e limitado ambiente físico. O controle e gerenciamento são realizados de maneira centralizada. Do ponto de vista do usuário, é tido como um único recurso independente da quantidade de servidores envolvidos.

CORBA: *Common Object Request Broker Architecture* possibilita que haja comunicação entre objetos, independente da linguagem de programação ou sistema operacional utilizados. Com isso a troca de dados entre sistemas distribuídos é simplificada. É muito utilizada em linguagens de programação orientadas a objetos.

CP: *Constraint Programming* consiste na solução de problemas de satisfação de restrições através de variáveis associadas a um domínio e a um conjunto de restrições. É um *framwork* adequado para planejamento que envolve tempo e recursos para identificar resultados qualitativos, quantitativos e ações concorrentes.

DAIS: *Data Access and Integration Services* é um grupo do *Grid Global Fórum* que tem trabalhado na definição de requisitos de integração dos serviços de bancos de dados com a grid computacional.

Framework: estrutura de sistema que dá subsídio para que outro sistema seja desenvolvido. Normalmente um framework inclui uma série de programas de apoio, bibliotecas de código e fragmentos de código que são utilizados para facilitar o desenvolvimento de sistemas.

GDS: *Grid Data Service* é o serviço responsável pela comunicação com o banco de dados.

GDSF: *Grid Data Service Factory* é responsável pela criação de instâncias dos serviços de dados da grid.

GDSR: *Grid Data Service Registry* é responsável pela publicação dos serviços de dados da grid.

GDT: *Grid Data Transport* realiza o transporte de dados entre instâncias dos serviços de dados da grid.

GGF: *Global Grid Forum* é uma comunidade formada por usuários, desenvolvedores e fornecedores que têm pesquisado as melhores práticas e especificações dos padrões para grid computacional.

Globus: principal *middleware* de grid disponível. É um conjunto de ferramentas que realiza o trabalho de conectar e gerenciar recursos distribuídos.

GRAM: *Globus Resource Allocation Manager* é o responsável pela interface do Globus com os recursos dos computadores locais.

Grid: é um grupo de recursos heterogêneos, distribuídos e integrados que são vistos como um único recurso computacional sem restrição para área de atuação. O controle e gerenciamento são realizados de maneira descentralizada.

GridBus: tem como objetivo integrar tecnologias que permitam a integração entre a grid e as empresas de negócios.

GridFTP: mecanismo de acesso e transferência de dados disponível no Globus.

GSF: *Grid Service Factory* é responsável pela criação de uma nova instância de serviço na grid.

GSH: *Grid Service Handle* é o identificador único de um serviço da grid. Não contém a localização e não permite, isoladamente, a comunicação com o serviço.

GSI: *Grid Security Infrastructure* é o conjunto de serviços responsáveis pela segurança da grid. Está integrado ao Globus e permite, entre outros, o controle de acesso aos recursos.

GSR: *Grid Service Reference* contém o protocolo e o endereço de cada serviço para que se estabeleça a comunicação entre os serviços da grid.

IP: *Integer Programming* é um *framework* que utiliza funções lineares para achar soluções em uma árvore de busca. É uma técnica de Inteligência Artificial para programação de restrições.

JDBC: *Java Database Connectivity* é uma API do Java para execução de comandos SQL em banco de dados.

Legion: é um sistema que funciona como um computador virtual, com desenvolvimento baseado em orientação a objeto e que tem alta escalabilidade, facilidade de programação e é tolerante a falhas.

MDS: *Metacomputing Directory Services* é o serviço de diretório da grid que permite a localização e recuperação das informações dos recursos disponíveis.

Middleware: software que conecta duas ou mais aplicações através da leitura e envio de dados necessários ao processamento.

MTBF: *Mean Time Between Failures* é o tempo médio entre falhas de um componente. É uma medida que determina o tempo médio em horas que se pode confiar em um *hard disk* ou uma impressora, por exemplo.

ODBC: *Open Database Connectivity* é um *driver* que permite às aplicações acessar os dados armazenados em SGBD.

ODMG: *Object Database Management Group* foi um grupo que até 2001 estabeleceu padrões para acesso a dados sob o paradigma da orientação a objeto. Atualmente o ODBMS (*Object Database Management Systems*) dá continuidade ao projeto.

OGSI: *Open Grid Services Infrastructure* é um conjunto de especificações WSDL que define padrões de interface e de ações para a OGSA.

OGSA: *Open Grid Services Architecture* é um conjunto de definições, conceitos e tecnologia de serviços *Web* que são construídos sobre a OGSI.

OGSA-DAI: *Data Access and Integration* é um projeto para desenvolver um *middleware* para integrar o banco de dados em uma grid computacional. O mesmo nome é dado ao produto que permite a utilização de recursos de banco de dados relacionais e XML em grid computacional.

OGSA-DQP: *Data Query Processor* é um produto criado para realizar o processamento de serviços distribuídos de busca. É baseado no projeto Polar*.

OQL: *Object Query Language* é uma linguagem de busca semelhante ao SQL, mas com recursos especiais para tratar objetos complexos, valores e métodos.

P2P: *Peer-to-peer* é um ambiente distribuído onde cada estação da rede tem as mesmas capacidades e responsabilidades. Permite o compartilhamento de recursos entre as estações.

Polar*: É um projeto derivado do Polar (*Parallel Object-oriented database*) que tem pesquisado meios de integrar serviços de busca distribuída na grid.

SAT: *Satisfiability* é um *framework* utilizado em Inteligência Artificial que, através de fórmulas proposicionais e restrição de variáveis, permite a programação através de restrições.

SDE: *Service Data Elements* descreve os parâmetros da virtualização de dados e as operações realizadas pelos serviços de dados da grid.

SDK: *Software Development Kit* é um pacote utilizado para desenvolver sistemas para uma determinada plataforma. É formado por uma ou mais APIs.

SGBD: Sistema Gerenciador de Banco de Dados é um conjunto de programas responsável pelo armazenamento, alteração e extração de informações em um banco de dados. São produtos, comercializados ou não, que utilizam a memória (volátil e não volátil) e a capacidade de processamento dos computadores para gerenciar os dados armazenados.

SOA: *Service-Oriented Architecture* é uma arquitetura onde os serviços são definidos através de uma linguagem de descrição e interfaces que têm como objetivo a realização de processos de negócio.

SOAP: *Simple Object Access Protocol* é um protocolo de troca de mensagens baseado em XML utilizado para transportar informações de *Web Services*.

SQL: *Structured Query Language* é a linguagem padrão para manipulação de dados em banco de dados relacionais e objeto-relacionais.

SRB: *Storage Resource Broker* é um *middleware* cliente-servidor projetado para permitir que as aplicações pudessem acessar informações distribuídas em recursos heterogêneos.

SSP: *Storage Service Providers* são provedores de serviços de compartilhamento de armazenamento de dados através de VPN.

UDDI: *Universal Description Discovery and Integration* é um diretório distribuído estabelecido na Internet e serve para descoberta de serviços. Pode ser comparado a um catálogo telefônico.

UML: *Unified Modeling Language* é uma linguagem de uso geral para especificação de sistemas de processamento de dados orientados a objeto.

VPN: *Virtual Private Network* é uma rede criada a partir de serviços públicos de comunicação de dados. Provê o acesso controlado e fornece mecanismos de segurança através de encriptação do tráfego de dados para garantir que somente pessoas autorizadas possam ter acesso à rede.

XML: *eXtensible Markup Language* é uma linguagem onde os desenvolvedores podem criar suas próprias marcações. Isso permite que se estabeleçam padrões, mecanismos de transmissão, interpretação e validação dos dados contidos no arquivo.

Web Service: é um padrão de integração de aplicações através da Internet. É utilizado em aplicações comerciais para comunicação entre classes de negócios e clientes sem a necessidade de um amplo conhecimento sobre a base tecnológica utilizada em qualquer um dos ambientes envolvidos.

WSDL: *Web Services Description Language* é uma linguagem em formato XML utilizada para descrever a troca de mensagens através de *Web Services*.

Anexo A – Estrutura da Tabela de Métrica

Os comandos para criação da tabela de métrica são:

```
/**
 * Tabela de Armazenamento de Métricas
 * ORACLE
 */

create table dai_metric
 ( metric_id number,
   metric_name varchar2(100),
   metric_desc varchar2(200),
   value number,
   maxvalue number,
   match number );

create sequence sqmetric;

/**
 * Tabela de Armazenamento de Métricas
 * MySQL
 */

create table dai_metric
 ( metric_id integer auto_increment,
   metric_name varchar(100),
   metric_desc varchar(200),
   value integer,
   maxvalue integer,
   match integer );
```

Anexo B – Estrutura das Tabelas de Teste

Os comandos para criação das tabelas de teste são:

```
/**
 Criação no Banco de Dados Oracle
*/
DROP TABLE lymph_outcome PURGE;
CREATE TABLE lymph_outcome (sample_id VARCHAR2(6),
                            full_ipi  VARCHAR2(20),
                            surtime   NUMBER,
                            status    VARCHAR2(20),
                            outcome   VARCHAR2(1));

DROP TABLE lymph_map PURGE;
CREATE TABLE lymph_map (gene_id     NUMBER,
                        description CLOB,
                        accession   VARCHAR2(4000));

DROP TABLE lymph_temp PURGE;
CREATE TABLE lymph_temp (gene_id NUMBER,
                         DLBC1   NUMBER,
                         DLBC2   NUMBER,
                         DLBC3   NUMBER,
                         DLBC4   NUMBER,
                         DLBC5   NUMBER,
                         DLBC6   NUMBER,
                         DLBC7   NUMBER,
                         DLBC8   NUMBER,
                         DLBC9   NUMBER,
                         DLBC10   NUMBER,
                         DLBC11   NUMBER,
                         DLBC12   NUMBER,
                         DLBC13   NUMBER,
```

```
DLBC14   NUMBER,
DLBC15   NUMBER,
DLBC16   NUMBER,
DLBC17   NUMBER,
DLBC18   NUMBER,
DLBC19   NUMBER,
DLBC20   NUMBER,
DLBC21   NUMBER,
DLBC22   NUMBER,
DLBC23   NUMBER,
DLBC24   NUMBER,
DLBC25   NUMBER,
DLBC26   NUMBER,
DLBC27   NUMBER,
DLBC28   NUMBER,
DLBC29   NUMBER,
DLBC30   NUMBER,
DLBC31   NUMBER,
DLBC32   NUMBER,
DLBC33   NUMBER,
DLBC34   NUMBER,
DLBC35   NUMBER,
DLBC36   NUMBER,
DLBC37   NUMBER,
DLBC38   NUMBER,
DLBC39   NUMBER,
DLBC40   NUMBER,
DLBC41   NUMBER,
DLBC42   NUMBER,
DLBC43   NUMBER,
DLBC44   NUMBER,
DLBC45   NUMBER,
DLBC46   NUMBER,
DLBC47   NUMBER,
```

```
                                DLBC48   NUMBER,
                                DLBC49   NUMBER,
                                DLBC50   NUMBER,
                                DLBC51   NUMBER,
                                DLBC52   NUMBER,
                                DLBC53   NUMBER,
                                DLBC54   NUMBER,
                                DLBC55   NUMBER,
                                DLBC56   NUMBER,
                                DLBC57   NUMBER,
                                DLBC58   NUMBER);

     /**
      Criação no Banco de Dados MySQL
     */

     CREATE TABLE IF NOT EXISTS LYMPH_EXP
        ( GENE_ID INTEGER,
          SAMPLE_ID VARCHAR(6),
          GENE_EXP INTEGER
        ) ;

     CREATE  TABLE  IF  NOT  EXISTS  lymph_outcome  (sample_id
VARCHAR(6),
                                   full_ipi  VARCHAR(20),
                                   surtime   INTEGER,
                                   status    VARCHAR(20),
                                   outcome   VARCHAR(1));

     CREATE TABLE IF NOT EXISTS lymph_map (gene_id     INTEGER,
                               description LONGTEXT,
                               accession   VARCHAR(4000));

     CREATE TABLE IF NOT EXISTS lymph_temp (gene_id INTEGER,
```

```
DLBC1   INTEGER,
DLBC2   INTEGER,
DLBC3   INTEGER,
DLBC4   INTEGER,
DLBC5   INTEGER,
DLBC6   INTEGER,
DLBC7   INTEGER,
DLBC8   INTEGER,
DLBC9   INTEGER,
DLBC10   INTEGER,
DLBC11   INTEGER,
DLBC12   INTEGER,
DLBC13   INTEGER,
DLBC14   INTEGER,
DLBC15   INTEGER,
DLBC16   INTEGER,
DLBC17   INTEGER,
DLBC18   INTEGER,
DLBC19   INTEGER,
DLBC20   INTEGER,
DLBC21   INTEGER,
DLBC22   INTEGER,
DLBC23   INTEGER,
DLBC24   INTEGER,
DLBC25   INTEGER,
DLBC26   INTEGER,
DLBC27   INTEGER,
DLBC28   INTEGER,
DLBC29   INTEGER,
DLBC30   INTEGER,
DLBC31   INTEGER,
DLBC32   INTEGER,
DLBC33   INTEGER,
DLBC34   INTEGER,
```

```
DLBC35   INTEGER,
DLBC36   INTEGER,
DLBC37   INTEGER,
DLBC38   INTEGER,
DLBC39   INTEGER,
DLBC40   INTEGER,
DLBC41   INTEGER,
DLBC42   INTEGER,
DLBC43   INTEGER,
DLBC44   INTEGER,
DLBC45   INTEGER,
DLBC46   INTEGER,
DLBC47   INTEGER,
DLBC48   INTEGER,
DLBC49   INTEGER,
DLBC50   INTEGER,
DLBC51   INTEGER,
DLBC52   INTEGER,
DLBC53   INTEGER,
DLBC54   INTEGER,
DLBC55   INTEGER,
DLBC56   INTEGER,
DLBC57   INTEGER,
DLBC58   INTEGER);
```

Anexo C – Roteiro de Instalação do Ambiente

Os comandos para instalação do ambiente são:

Opção 1: Instalação sob o Globus Toolkit versão 4:

```
ROTEIRO INSTALAÇÃO GT4
======================
- Descompactar GT4
- Adicionar variável de ambiente GLOBUS_LOCATION (c:\gt4)
- Instalar JDK 1.4 ou superior
- Instalar ANT
- Instalar TOMCAT
- Inicializar TOMCAT
- Deploy no TOMCAT:
c:
cd\gt4
ant -f share/globus_wsrf_common/tomcat/tomcat.xml
deploySecureTomcat -Dtomcat.dir="c:\Tomcat"
# Startup no TOMCAT
ant -f share/globus_wsrf_common/tomcat/tomcat.xml war -
Dwar.file=c:\gt4\gt4.war

- Copiar c:\GT4\LIB\AXIS-URL.JAR para
TOMCAT_DIR\COMMON\LIB

ROTEIRO INSTALAÇÃO ANT
======================
- Descompactar em um diretório (adicionar variável de
ambiente ANT_HOME - c:\ant)
- Adicionar o BIN no PATH (c:\ant\bin)
- JAVA_HOME=c:\java\jdk1.5.0
- Copiar JUNIT.JAR (dentro de JUNIT.ZIP) para f:\ant\lib

ROTEIRO INSTALAÇÃO TOMCAT
```

```
     ========================
     - Instalar em c:\Tomcat
     - Executável (admin/admin)
     - Se não der porta 8080, colocar 8081
     - CATALINA_HOME=C:\Tomcat

     TESTES:
     =======
     bin\globus-start-container -nosec
     ant -f share/globus_wsrf_test/runtests.xml runServer -
Dtests.jar = %GLOBUS_LOCATION%/lib/wsrf_test_interop.jar

     ant -f share/globus_wsrf_test/runtests.xml runServer -
Dtests.jar = %GLOBUS_LOCATION%/lib/wsrf_test_unit.jar -
DbasicTestsOnly = true
```

Opção 2: Instalação no Axis

```
     INSTALAÇÃO DO AXIS
     ==================
     - Copiar WEBAPPS/AXIS para C:\Tomcat\WebApps\
     - Descompactar (JAF-1_0_2-UPD2.ZIP) activation.jar no
C:\Tomcat\WebApps\Axis\WEB-INF\lib
     - Descompactar (JAVAMAIL-1_3_3_3_01.ZIP) mail.jar no
C:\Tomcat\WebApps\Axis\WEB-INF\lib
     - Descompactar (XML-SECURITY-BIN-1_3_0.ZIP) xmlsec-
1.3.0.jar no C:\Tomcat\WebApps\Axis\WEB-INF\lib
     - Copiar C:\Java\jdk1.5.0\lib\TOOLS.JAR para
C:\Tomcat\common\lib
     - Criar variáveis de ambiente para
     AXIS_HOME=c:\axis
     AXIS_LIB=%AXIS_HOME%\lib
     AXISCLASSPATH=%AXIS_LIB%\axis.jar;%AXIS_LIB%\jaxrpc.jar;%A
XIS_LIB%\commons-discovery-0.2.jar;%AXIS_LIB%\commons-logging-
```

```
1.0.4.jar;%AXIS_LIB%\activation.jar;%AXIS_LIB%\mail.jar;%AXIS_
LIB%\saaj.jar;%AXIS_LIB%\axis-ant.jar;%AXIS_LIB%\log4j-
1.2.8.jar;%AXIS_LIB%\log4j-properties.jar;%AXIS_LIB%\wsdl4j-
1.5.1.jar
```

Para ambos os casos, instalar e testar o OGSA-DAI (é necessário substituir o diretório correto, ou “wsrf” ou “axis” nos comandos a seguir).

```
INSTALAÇÃO DO OGSA-DAI
======================
- Descompactar binário fornecido no diretório
cd\ogsadai-axis
ant install -Ddai.container = %CATALINA_HOME%

ant deployService -Ddai.container = %CATALINA_HOME% -
Ddai.service.name = ogsadai/DataService -Ddai.service.config=y

-- RESTART no TOMCAT
ant listResourcesClient -Ddai.url =
http://localhost:8080/axis/services/ogsadai/DataService

-- Copiar JARs do BD para o diretório C:\OGSADAI\DRIVERS
-- Deploy Resource:
ant deployResource -Ddai.container = %CATALINA_HOME% -
Ddai.resource.file = data.service.resource.propertiesOracle

ant deployResource -Ddai.container = %CATALINA_HOME% -
Ddai.resource.file = data.service.resource.propertiesMySQL

-- Expose Resource:
ant exposeResource -Ddai.container = %CATALINA_HOME% -
Ddai.service.name = ogsadai/DataService -Ddai.resource.id =
orclCPoderoso
```

```
     ant exposeResource -Ddai.container = %CATALINA_HOME% -
Ddai.service.name = ogsadai/DataService -Ddai.resource.id =
MySQLCPoderoso

     -- RESTART no TOMCAT
     ant listResourcesClient -Ddai.url =
http://localhost:8080/axis/services/ogsadai/DataService
```