PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

UMA ARQUITETURA PARA BANCOS DE DADOS DISTRIBUÍDOS APOIADA EM SERVIÇOS

Breno Mansur Rabelo

Belo Horizonte

2008

Breno Mansur Rabelo

UMA ARQUITETURA PARA BANCOS DE DADOS DISTRIBUÍDOS APOIADA EM SERVIÇOS

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Clodoveu Augusto Davis Junior

Belo Horizonte

2008

PUC Minas
Programa de Pós-graduação em Informática

FOLHA DE APROVAÇÃO

" Uma Arquitetura para Banco de Dados Distribuídos Apoiada em Serviços"

Breno Mansur Rabelo

Dissertação defendida e aprovada pela seguinte banca examinadora:

Prof. Clodoveu Augusto Davis Junior - Orientador (PUC Minas)
Prof. Alberto Henrique Frade Laender (UFMG)
Prof. Humberto Torres Marques Neto (PUC Minas)
Prof. Marco Túlio de Oliveira Valente (PUC Minas)

Prof. Clodoveu Augusto Davis Júnior - Orientador (PUC Minas)
Doutor em Ciência da Computação – UFMG

Prof. Alberto Henrique Frade Laender (UFMG)
Doutor em Ciência da Computação
University of East Anglia, Inglaterra.

Prof. Humberto Torres Marques Neto (PUC Minas)
Doutor em Ciência da Computação - UFMG

Prof. Marco Túlio de Oliveira Valente - Orientador (PUC Minas)
Doutor em Ciência da Computação - UFMG

Belo Horizonte, 11 de agosto de 2008.

AGRADECIMENTOS

Um trabalho acadêmico é, geralmente, extenso e complexo. Tecnicamente é um trabalho individual, mas na prática é necessária uma sinergia muito grande entre diversas pessoas que direta e indiretamente interagem até sua conclusão. Portanto, agradeço a Deus e, nominalmente, a algumas pessoas:

Primeiramente a minha mãe Marise de Oliveira Mansur Rabelo e ao meu pai Marcelo de Paulo Rabelo, que dedicaram sua vida a minha educação e dos meus irmãos. Vocês investiram e confiaram em nós, portanto gostaria de compartilhar mais uma conquista. E tenho certeza que muitas outras ainda virão...

Especialmente ao meu tio Carlos Salomão de Oliveira Mansur, que sempre foi como um segundo pai pra mim e que me incentivou a ingressar nesse desafio. Sua opinião sempre foi e sempre será muito importante para mim.

Ao meu falecido avô Said Mansur, que na verdade nem cheguei a conhecer, mas que segundo a minha mãe, foi o grande idealizador de tudo isso. Ele quis oferecer aos seus filhos uma boa educação, mas não teve tempo suficiente para isso. Minha mãe seguiu seus pensamentos e deu sua vida por este ideal. Onde ele estiver, sei que está feliz por ver resultados como esse. Obrigado por ser um homem de valor inestimável e por transmitir esses valores através de nossas gerações. Saiba que eles continuarão a ser transmitidos, em seu nome!

A minha namorada, companheira e futura esposa, Vanêssa Mariane Alves que sempre esteve comigo, presente em cada minuto, mesmo que em pensamento e que me ajudou como pode, esfriando minha cabeça nos momentos mais difíceis, me incentivando em momentos de descrença. Somente você poderia estabilizar meus sentimentos e sem você, meu amor, nada disso teria acontecido.

Ao senhor Ubirajara Campos Filho e ao senhor Marcelo Calonge, respectivamente Diretor Administrativo e Financeiro e Superintendente da Mendesprev Sociedade Previdenciária, meus grandes tutores profissionais, que viabilizaram a conclusão deste trabalho, em paralelo às minhas atividades na Mendesprev.

Aos meus amigos Nilson Zeferino Ribeiro, Tiago Franco Martins e Vander José Resende de Oliveira, companheiros da Mendesprev, que sempre fizeram força para entender meu trabalho e escutar minhas dificuldades, mesmo sem entender muito do que eu estava falando.

A outros dois grandes amigos Victor Hugo Portela Campos e Thiago Marotta Couto, que foram muito importantes principalmente na última fase do trabalho. Sem a ajuda de vocês ele poderia ter ficado sem algumas contribuições interessantes, gerada pela soma de nossas conversas e discussões. Muito obrigado meus amigos!

E enfim, meus sinceros agradecimentos ao meu "caríssimo" professor e orientador Dr. Clodoveu Augusto Davis Júnior. Obrigado pela força, incentivo e atenção, sobretudo nos momentos em que tudo parecia não funcionar mais. Sem a sua ajuda e sua flexibilidade teria sido impossível conciliar Mendesprev, UEMG e o mestrado.

RESUMO

A crescente disseminação de sistemas de informação distribuídos em rede tem reativado o interesse em sistemas de gerenciamento de bancos de dados distribuídos (SGBDD). Uma revisão na arquitetura de tais sistemas é hoje motivada pela ampla disponibilidade de redes de computadores, em particular a Internet, através das quais é possível reduzir o custo de comunicação de dados entre os nodos de um banco de dados distribuído. Além disso, o surgimento de padrões de interoperabilidade sintática, tais como o XML, e de recursos padronizados para serviços em rede viabilizam uma concepção alternativa para o gerenciamento de bancos de dados distribuídos. Esta dissertação apresenta uma revisão dos componentes clássicos de SGBDD sob o ponto de vista tecnológico, e propõe a adoção de recursos típicos de arquiteturas orientadas por serviços para a implementação da conexão entre eles, formando, portanto, uma arquitetura de bancos de dados distribuídos apoiada em serviços. Um protótipo foi projetado, implementado, testado e avaliado, demonstrando que a idéia proposta funciona na prática. O trabalho é concluído com a apresentação de pontos fortes e fracos da proposta e com apontamentos para possíveis trabalhos futuros que ajudariam a tornar viável a utilização da arquitetura em aplicações reais.

Palavras-chave: Bancos de dados distribuídos, arquitetura orientada a serviços, serviços Web, sistema gerenciador de bancos de dados, integração de dados, processamento de consultas distribuídas.

ABSTRACT

The increasing dissemination of information systems over computer networks has reinforced the interest on distributed database management systems (DDBMS). A review on the architecture of such systems is currently motivated by the wide availability of networking resources, especially the Internet, through which the cost of communication among the nodes of a distributed database can be reduced. Besides, the coming of age of syntactic interoperability standards, such as XML, and of service-based networking allow for an alternative set of ideas for the implementation and deployment of distributed databases. This work presents a review of the classical distributed database management architecture from a technological standpoint, and proposes the adoption of elements from service-oriented architectures for the implementation of the connections among such components, thus configuring a service-oriented distributed database architecture. A prototype was designed, implemented, tested and evaluated, demonstrating that the proposed idea works in practice. The work is concluded with the presentation of strengths and weaknesses of the proposal and notes for possible future work that would help to make viable the use of the architecture in real applications.

Keywords: Distributed database, service-oriented architecture, web services, database management system, data integration, distributed query processing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CDE – Command Decomposition Engine

CP – Command Part

DEE – Distributed Execution Engine

DO – Distribution Optimizer

DP – Distribution Plan

DPN – Distribution Plan Node

DTC – Distributed Transaction Coordinator

JDBC – Java Database Connectivity

ODBC – Open Data Base Connectivity

OLEDB – Object Linking and Embedding Database

QDE – Query Decomposition Engine

QP – Query Part

RDM – Replicated Data Manager

SOA – Service Oriented Architecture

SGBD – Sistema Gerenciador de Banco de Dados

SGBDD – Sistema Gerenciador de Banco de Dados Distribuídos

SODA – Service Oriented Database Architecture

SODDA – Service Oriented Distributed Database Architecture

TST – Transaction Status Table

SUMÁRIO

1 INTRODUÇÃO

Os sistemas de informação vêm se tornando presença comum nas empresas e, cada dia mais, essas empresas optam em integrar seus sistemas com os de seus parceiros, visando melhorias em suas cadeias produtivas, incentivadas pela agilização do processo de logística. É cada vez mais comum a necessidade de gerenciamento de dados distribuídos fisicamente, armazenados em servidores distintos e independentes. Na medida em que o volume de dados distribuídos cresce e aumenta a necessidade de integração entre essas diversas fontes, torna-se importante o uso de técnicas de gerenciamento de bancos de dados distribuídos para, virtualmente, formar um grande banco de dados a partir de segmentos isolados ou independentes. Dentre os motivos para essa tendência estão limitações computacionais, custos dos servidores de grande porte e a descentralização operacional de vários tipos de organização.

Kossman (2000) lembra que, apesar de terem sido apresentadas boas idéias nas pesquisas realizadas sobre bancos de dados distribuídos, os protótipos desenvolvidos não chegaram a se tornar ferramentas comerciais reconhecidas. No mercado existem opções de sistemas gerenciadores de bancos de dados (SGBD) com suporte a distribuição, como as soluções IBM e Oracle. Porém, além de não implementar o modelo conceitual completamente, como lembram Elmasri e Navathe (2005), por não serem utilizados em larga escala essas implementações comerciais também não conseguiram estabelecer novos padrões para sistemas gerenciadores de bancos de dados distribuídos (SGBDD). Como um padrão para gerenciamento de dados distribuídos não foi estabelecido, os SGBDD aparentemente não conseguiram se firmar no mercado.

Apesar de os pesquisadores confirmarem as possibilidades relacionadas ao uso de bancos de dados distribuídos, as pesquisas não tiveram o alcance comercial esperado, provavelmente devido ao momento histórico desfavorável. Para Kossman (2000), as pesquisas foram efetuadas antes de seu tempo, principalmente pela inexistência de uma estrutura de comunicação suficientemente estável para transferência de dados, como a Internet.

A perspectiva de implementação de bancos de dados distribuídos na época em que esse conceito surgiu era bastante diferente da atual. Com a crescente disponibilidade de recursos para comunicação em rede, usando a Internet, aliada à crescente qualidade e a uma forte tendência para redução de custos, algumas das inibições naturais contra o

desenvolvimento de bancos de dados distribuídos desapareceram. No entanto, as bases teóricas dessa área não consideram as peculiaridades relativas ao uso dos padrões e facilidades hoje associadas ao tráfego de dados na Internet, tais como XML (Bray *et al.* 2006), serviços Web (Ferris e Farrel 2003; Alonso *et al.* 2004), SOAP (*Simple Object Access Protocol*) (Curbera *et al.* 2002) e arquiteturas orientadas por serviços (Endrei *et al.* 2004; Huhns e Singh 2005) entre outras. Com isso, podem-se reunir elementos tecnológicos novos, que motivam uma reavaliação dos SGBDD, considerando SOA (*Service-Oriented Architectures*, arquiteturas orientadas por serviços) para suportar a comunicação de dados via Internet. As novas necessidades de uso de bancos de dados distribuídos ampliam as possibilidades de uso das idéias anteriores em um contexto tecnológico mais favorável, o que certamente pode gerar alternativas interessantes aos atuais sistemas de bancos de dados centralizados.

Trabalhos recentes, como o de Campbell (2005) e o de Tok e Bressan (2006), apresentam iniciativas relacionadas à integração entre arquiteturas apoiadas em serviços e bancos de dados, propondo o surgimento da arquitetura SODA (*Service Oriented Database Architecture*). Este trabalho propõe uma extensão da arquitetura SODA, na qual é proposta uma metodologia para o gerenciamento de bancos de dados distribuídos através da Internet, com comunicação apoiada em SOA. A arquitetura referencial de bancos de dados distribuídos descrita por Elmasri e Navathe (2005) foi adotada como premissa básica para adaptação dos trabalhos anteriores. Para a comunicação e interoperabilidade entre os nodos do banco de dados distribuído são utilizados serviços Web, que possibilitam trabalhar com sistemas fisicamente distribuídos, através da Internet. A contribuição deste trabalho é uma arquitetura de bancos de dados distribuídos apoiados em serviços, que denominamos SODDA (*Service Oriented Distributed Database Architecture*), e é uma junção entre os conceitos clássicos de SGBDD e as recentes iniciativas das arquiteturas SOA e SODA.

1.1 Objetivo

O objetivo geral deste trabalho é propor uma nova arquitetura para bancos de dados distribuídos apoiada em serviços e demonstrar sua viabilidade prática. O uso de tal arquitetura deverá garantir que um sistema cliente seja capaz de se acoplar a nodos do banco de dados distribuído através de serviços Web, e obter uma visão integrada dos dados, com transparência de localização.

A arquitetura proposta nesta dissertação define funções que garantem ganhos de desempenho e tolerância a falhas, sem perda da transparência de uso do banco de dados para os sistemas. Dessa maneira, espera-se que o desenvolvedor de aplicações baseadas na arquitetura proposta não precise se preocupar em como a arquitetura funciona internamente.

A arquitetura proposta foi implementada em um protótipo, apresentado no Capítulo 4, de modo a demonstrar sua viabilidade.

1.2 Organização do trabalho

Esta dissertação está organizada da seguinte maneira. O Capítulo 2 apresenta trabalhos relacionados ou utilizados como base da proposta apresentada, e aponta deficiências que são tratadas no trabalho de pesquisa. O Capítulo 3 apresenta a arquitetura SODDA (*Service Oriented Distributed Database Architecture*) e seu funcionamento. O Capítulo 4 descreve a implementação de um protótipo da arquitetura proposta e termina com a realização de um teste de funcionamento. O Capítulo 5 apresenta a conclusão do trabalho e define diretrizes para trabalhos futuros.

2 TRABALHOS RELACIONADOS

Este capítulo apresenta uma revisão de trabalhos relacionados aos objetivos da dissertação, envolvendo conceitos clássicos de bancos de dados distribuídos e algumas tentativas documentadas na literatura de integrar bancos de dados a arquiteturas orientadas por serviços. Além desses aspectos, são apresentadas iniciativas de gerenciamento de bancos de dados em outro tipo de ambiente distribuído, as grades (*grids*) computacionais.

2.1 Bancos de dados distribuídos

Ozsu e Valduriez (1999) definem bancos de dados distribuídos (BDD) como uma coleção de bancos de dados relacionados e localizados em diversos nós de uma rede de computadores. Sistemas gerenciadores de bancos de dados distribuídos (SGBDD) são pacotes de software que permitem o gerenciamento de bancos de dados distribuídos de forma transparente, ou seja, de forma que o acesso ao BDD não seja diferente do acesso a um banco de dados centralizado convencional.

Para materializar a conexão entre os bancos de dados espalhados em uma rede, é necessário ter disponível um mecanismo completo para gerenciar a distribuição dos dados e viabilizar o acesso ao conjunto dos dados por meio de uma interface comum. A diferença entre um banco de dados centralizado e um banco de dados distribuído pode ser vista na Figura 1a onde é apresentado esquematicamente um sistema composto por múltiplos bancos de dados centralizados. Nesse ambiente, apesar de existirem várias fontes de dados e alguma forma de comunicação entre elas, todas as operações devem ser submetidas individualmente aos nodos detentores das fontes. A Figura 1b apresenta um bancos de dados distribuídos, onde as fontes de dados estão dispersas pelos nodos, mas o acesso é realizado de modo que, para os clientes, existe aparentemente um banco de dados “virtual”, cujo conteúdo reflete a união do conteúdo dos bancos de dados existentes em cada nodo.

Elmasri e Navathe (2005) afirmam que ainda não existem implementações comerciais completas da arquitetura referencial de bancos de dados distribuídos. Isso pode ser confirmado ao se analisar a documentação de alguns dos mais importantes pacotes de software comerciais atuais (Microsoft 2007; MySQL AB 2008; Oracle 2008; PostgreSQL 2008). O SQL Server 2005, o MySQL 5.3 e o PostgreSQL 8.3.3 se preocupam apenas alguns aspectos de distribuição, como replicação e balancemento de carga, visando quase que exclusivamente resistência a falhas e aumento de disponibilidade. O Oracle 11g incorpora

algumas funções adicionais de bancos de dados distribuídos, como o particionamento de tabelas, mas peca em questões de transparência.

Para implementar um SGBDD, Ozsu e Valduriez (1999) afirmam ser necessária a construção de algumas camadas de transparência. Tais camadas são necessárias para esconder das aplicações e usuários finais algumas características de bancos de dados distribuídos, como a fragmentação e a replicação dos dados, além dos aspectos de comunicação e rede e a heterogeneidade dos componentes dessa rede.

Uma deficiência comum e que comprova as afirmações dos autores sobre as limitações das atuais implementações de SGBDD é a ausência da transparência de rede em ferramentas comerciais. Devido a essa limitação, o desenvolvedor precisa conhecer nome e localização exata dos fragmentos de bancos de dados envolvidos em um processo de comunicação qualquer. Isso traz inconvenientes tanto na hora da comunicação, quanto na hora da manutenção, pois a cada mudança de localização de um determinado nodo de rede é necessário reconstruir toda a ligação entre os bancos de dados relacionados. O Oracle 11g é o principal exemplo dessa deficiência. Apesar de ser provavelmente a mais completa implementação comercial da arquitetura referencial de banco de dados distribuídos, a localização de nodos é claramente não transparente. Para realizar consultas é necessário usar uma expressão SQL modificada como: "*SELECT * FROM tabela@nome_banco.dominio.com"*, onde é necessária a informação do nome da tabela, o nome do banco de dados e o domínio em qual o banco de dados está contido. É possível simular a transparência de localização através de uso de sinônimos (*CREATE SYNONYM tabela FOR tabela@banco.dominio*), porém sinônimos não são instruções SQL padrão.

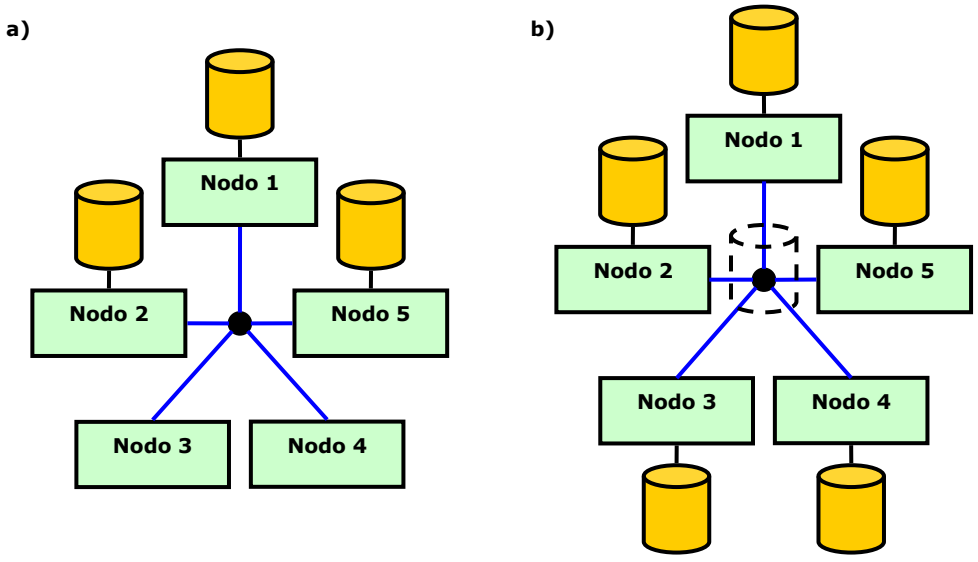

FIGURA 1. Bancos de dados centralizados vs. distribuídos.

Outra deficiência comum é a ausência de mecanismos de distribuição de dados que dêem suporte a fontes de dados heterogêneas. Como as opções que existem no mercado são em sua totalidade proprietárias, os fabricantes em geral não se preocupam em adaptar os mecanismos de distribuição visando acomodar fontes de dados alternativas à que eles desenvolvem. Com isso, não se consegue de fato obter SGBDD heterogêneos, ou federados, que possam funcionar adequadamente, e permanece a inconveniência de se forçar o usuário a adotar sempre o mesmo produto em todos os nodos do SGBDD. Com isso, usuários e desenvolvedores ficam sujeitos à cobrança de licenças adicionais de software mesmo em situações em que determinados nodos poderiam usar SGBDD livres ou de menor porte, por exemplo.

Apresentadas algumas deficiências das ferramentas comerciais atuais, Elmasri e Navathe (2005) afirmam ainda ser primordial a existência de alguns mecanismos para construção de um SGBDD completo. Tais mecanismos são os responsáveis pela funcionalidade básica de uma arquitetura de BDD e são apresentados brevemente a seguir.

Esquema de fragmentação. O esquema de fragmentação descreve como os dados serão divididos entre os nodos. Este esquema fica armazenado no Catálogo Global do BDD, que é consultado a cada interação com o banco de dados. O catálogo por sua vez indica, ao mecanismo que solicitou uma operação sobre o banco de dados, quais são os nodos que possuem os fragmentos de dados necessários para cumprir uma determinada tarefa. Podem existir fragmentos horizontais, formados por tuplas inteiras, e verticais, contendo um subconjunto das colunas de uma tabela. Em ambos os casos, a fragmentação deve ser feita de tal maneira que seja possível, usando operações relacionais, produzir uma versão integrada da tabela prevista no projeto lógico. Eventualmente, podem existir cópias total ou parcialmente redundantes de certos fragmentos. A fragmentação também pode ser mista, ou seja, produzir simultaneamente fragmentos horizontais e verticais. As informações necessárias sobre o esquema de fragmentação são armazenadas no catálogo do SGBD, que no caso do SGBDD é usualmente denominado “catálogo global”, por incluir dados sobre fragmentos situados em outros SGBD.

Processador de consultas distribuídas. O processador de consultas distribuídas é o mecanismo responsável pelo particionamento de consultas. Quando uma consulta é executada, o processador consulta o catálogo global e descobre os nodos do BDD que devem ser envolvidos para obter o resultado esperado. Após essa fase de descoberta, o processador trabalha na otimização da consulta, levando em consideração os custos relativos à distribuição dos dados. Além de utilizar, como critério de otimização, o número esperado de acessos a

disco, o processador de consultas distribuídas deve considerar também os custos referentes ao tráfego de dados na rede, que provavelmente serão o componente de custo dominante para consultas distribuídas.

Gerenciador de transações distribuídas. Este é um mecanismo utilizado em consultas que envolvem diversos nodos de um BDD. Ele acompanha os estados das transações demandadas a cada nó para que as alterações de estado no banco de dados sejam propagadas para todos os nodos ou, em caso de falha, não tenham efeito sobre nenhum deles. Ele é um importante mecanismo para manutenção da consistência do BDD.

Gerenciador de dados replicados. Como em um BDD podem existir diversas cópias de itens de dados, é necessário que o SGBDD possua um mecanismo capaz de garantir a consistência entre essas cópias, sincronizando-as. De nada adiantaria ter cópias redundantes dos dados, se essas estivessem desatualizadas, pois no momento em que fossem necessárias estariam em estado inválido. O gerenciador de dados replicados é o mecanismo responsável por essa sincronização em um SGBDD.

Recuperador de banco de dados. Este mecanismo é o responsável por descobrir quando um nodo perde conexão com o resto do BDD e por determinar qual cópia pode substituí-lo no período de indisponibilidade do serviço, caso existam fragmentos redundantes aos armazenados no nó que falhou. Trata-se, portanto de um mecanismo de tolerância a falhas do SGBDD.

Catálogo global. Por último, o BDD deve possuir um catálogo global, que armazena metadados sobre o BDD. Como já comentado anteriormente, o catálogo é acessado por diversos outros mecanismos do SGBDD, pois armazena informações de localização dos dados dispersos pelos nodos de um BDD. Este é um mecanismo de importância primordial para um BDD e, devido à dependência que os outros mecanismos apresentam em relação a este, pode ser considerado o mais importante.

Existem algumas vantagens na adoção de BDD em lugar de BD centralizados, tais como a melhoria de desempenho pela distribuição das consultas, e a possibilidade de replicação de dados em diferentes servidores, o que ajuda a evitar a interrupção do acesso caso ocorra falha em algum componente do BDD. Também ocorrem ganhos em escalabilidade, devido à possibilidade de expansão do ambiente pelo simples acréscimo de nodos.

No entanto, esse ganho potencial ainda não pôde ser plenamente alcançado, pois a arquitetura de BDD foi idealizada antes da grande expansão da Internet. Ao rever esse tema, pode se notar que a necessidade de soluções para troca de mensagens e dados entre nodos da

arquitetura constituía uma barreira à sua disseminação, já que os recursos de comunicação eram mais raros e envolviam custos econômicos mais altos do que atualmente. As novas tendências da orientação por serviços e a constante evolução das tecnologias de redes podem ser importantes ferramentas para retomada da idéia original de SGBDD, desta vez contando com a Internet e outros recursos padronizados, amplamente acessíveis e com custos bem menores.

As principais iniciativas recentes relacionadas a bancos de dados em sistemas distribuídos estão relacionadas ao uso de *clusters* e *grids* de computadores. No próximo capítulo é demonstrado como os bancos de dados são gerenciados nesse tipo de arquitetura.

2.2 Bancos de dados em grid

Um *cluster* (Buyya 1999a; Buyya 1999b) ou *grid* (Kacsuk, Fahringer e Németh 2007) é um sistema de processamento distribuído formado por dois ou mais computadores, que trabalham em conjunto para solução de problemas que exigem grande escala de processamento.

Os clusters se diferenciam dos *grids* em relação à coordenação do processamento. Nos *clusters* existe um computador coordenador, que delega trabalho aos outros computadores. Já nos *grids* não existe o papel de coordenador e o processamento é dividido entre os computadores de forma natural: quando a quantidade de trabalho ultrapassa um determinado limite, a tarefa é particionada e enviada a outro computador.

Com o crescente uso de arquiteturas orientadas a serviços, tornou-se interessante construir sistemas desse tipo, que apóiam sua comunicação em serviços Web, o que impulsionou o surgimento da arquitetura OGSA (*Open Grid Service Architecture*) (Foster *et al.* 2002), um padrão para criação de *grids* distribuídos pela Web e baseados em SOA.

O grupo de pesquisa que estuda a arquitetura OGSA foca seus esforços em aspectos de processamento e distribuição. No entanto, existem pesquisas paralelas sobre extensões aplicáveis à arquitetura. Uma dessas extensões é o *Data Access and Integration Services* (DAIS) (Paton *et al.* 2002). O DAIS é uma extensão que possibilita coordenação de unidades de bancos de dados, heterogêneas, distribuídas através de um *grid* baseado em OGSA. Essa extensão permite acesso padronizado a fontes de dados, construindo uma visão de banco de dados federado. Funções acessórias como replicação e fragmentação são herdadas do OGSA, e com isso a extensão se aproxima bastante do que a literatura define como bancos de dados distribuídos.

A herança de funções da arquitetura OGSA é interessante e inteligente, pois "terceiriza" uma série de funções dos bancos de dados distribuídos a outra camada do *grid*. Isso, porém, faz com que a extensão DAIS sofra algumas restrições em relação ao modelo conceitual de bancos de dados distribuídos, implementando de forma limitada algumas funcionalidades, como a fragmentação de dados. São ignoradas, por exemplo, as possibilidades de fragmentação vertical.

A restrição de fragmentação se agrava com a ausência de transparência de localização. Isso acontece porque o desenvolvedor poderá planejar a criação de bancos de dados isolados, simulando a fragmentação vertical, mas não consegue tratar a tabela como uma só, pois precisará informar onde estão os fragmentos de dados. Dessa forma, para retornar dados de uma tabela distribuída em cinco nodos, será necessário escrever cinco junções para obtenção do resultado desejado, o que em um banco de dados distribuído baseado no modelo conceitual seria possível com um comando apenas, como *"select * from tabela"*.

Enfim, questões de otimização de consultas não são bem resolvidas. As questões de comunicação, como tempo de acesso e escolha de nodos em um ambiente com replicação são tratadas pela arquitetura OGSA de forma genérica. Dessa maneira, os custos de comunicação também são tratados de forma pouco realista, pois ignoram aspectos como o volume de dados nas tabelas e a necessidade de realizar buscas redundantes, o que pode gerar *overhead* desnecessário na busca de informações no banco de dados distribuído.

Recentemente, novas iniciativas relacionadas à integração entre SOA e bancos de dados têm surgido. Na próxima seção são apresentadas algumas dessas iniciativas e são introduzidas algumas novas oportunidades que permanecem em aberto.

2.3 Arquitetura de bancos de dados apoiada em serviços

A arquitetura SODA (*Service Oriented Database Architecture*) é uma iniciativa recente que busca integração entre os SGBD e SOA (*Service Oriented Architecture*). A arquitetura SODA utiliza serviços Web, um dos pilares da SOA, para publicar funções e procedimentos capazes de manipular o conteúdo de bancos de dados distribuídos pela Internet.

Campbell (2005), criador da arquitetura SODA, apresenta em seu trabalho detalhes sobre a implantação da arquitetura no núcleo do Microsoft SQL Server. São expostas informações sobre a introdução de um *HTTP Daemon* no núcleo do sistema de banco de dados para garantir a total independência entre o servidor de banco de dados e o servidor web.

Tal iniciativa comprova que tem se tornado cada vez mais comum o uso de bancos de dados distribuídos através da Internet, ao ponto de tornar necessário um mecanismo nativo para resposta de requisições HTTP pelo próprio banco de dados. O autor acredita que a iniciativa pode, futuramente, padronizar e facilitar a integração entre bancos de dados distribuídos através da Internet. Avaliando a documentação do SQL Server 2005, percebe-se que a arquitetura SODA já faz parte da versão comercial produto.

Para Tok e Bressan (2006), muitos SGBD comerciais começam a apresentar funções de integração entre serviços Web e *stored procedures*, a exemplo do SQL Server. No entanto, o problema é que essa integração é apenas superficial e o processador de consultas do SGBD acaba desconhecendo a comunicação através do serviço Web em questão e despreza o custo dessa comunicação no processo de otimização da consulta.

Segundo Kiely (2006), a arquitetura SODA oferece suporte a cenários ocasionalmente conectados, onde os dados são armazenados em *cache* enquanto o cliente está desconectado e são sincronizados quando ele se conecta. Para o autor, a arquitetura SODA é uma alternativa aos servidores de aplicativos tradicionais e habilita uma nova classe de aplicativos focados em serviços, o que é claramente uma subutilização dos recursos disponíveis na adoção pela orientação a serviços.

Campbell (2005) mostrou pela primeira vez uma integração mais profunda entre o núcleo de um SGBD e serviços Web, porém Tok e Bressan (2006) acreditam que a nova arquitetura continua presa aos bancos de dados convencionais e poderia ser mais explorada.

De qualquer forma, o trabalho de Campbell (2005) traz grande contribuição. A arquitetura SODA não trata de aspectos de distribuição, mas se suas idéias forem exploradas mais a fundo, pode-se resolver boa parte dos problemas documentados em trabalhos anteriores sobre bancos de dados distribuídos e que permanecem sem solução, como por exemplo, a padronização da comunicação através de um protocolo comum, a padronização do formato das mensagens e o encapsulamento de fontes de dados heterogêneas.

Até onde foi possível determinar, não existem trabalhos de fato semelhantes a esta dissertação na literatura atual. Além das referências clássicas sobre SGBDD, apenas as iniciativas de gerenciamento de BD em *grid* tratam do problema de distribuição de dados.

No capítulo a seguir apresentamos a arquitetura SODDA (*Service Oriented Distributed Database Architecture*), que surge de uma das principais dificuldades da arquitetura referencial dos SGBDD: a comunicação.

3 A ARQUITETURA SODDA

Basicamente, a arquitetura SODDA reúne características da arquitetura referencial de SGBDD às novas tecnologias e iniciativas relacionadas à arquitetura SODA e à Internet. Nossa proposta para a definição da arquitetura SODDA é utilizar a Internet aliada à orientação a serviços como canal de comunicação em um SGBDD. Com isso, torna-se necessário adaptar as idéias e conceitos originais de SGBDD para utilizar as características de uma nova arquitetura de rede, e para aproveitar as características de serviços Web na comunicação entre componentes do BDD. As seções a seguir descrevem essas adaptações.

3.1 Orientação a serviços na arquitetura SODDA

A arquitetura SODDA utiliza serviços Web para coordenar operações entre os nodos do BDD. Cada nodo possui um serviço Web coordenador do banco de dados local, chamado *SODDA Hub*[1], capaz de responder a um provedor de dados no cliente quando o nodo recebe requisições de consultas e outras operações sobre o banco de dados. O SODDA Hub pode ser visto como um *middleware* comum de conectividade, como o OLEDB[2], JDBC[3] ou ODBC[4], que são alternativas conhecidas, porém voltadas para implementações monolíticas. O SODDA Hub roda no lado do cliente e é responsável por todas as interações com os componentes SOA da arquitetura SODDA. Conseqüentemente, na visão do desenvolvedor de aplicações, a arquitetura SODDA é um provedor de dados completamente transparente, e que pode ser usado como qualquer uma das opções monolíticas apresentadas anteriormente.

Diferentemente da arquitetura SODA, onde os serviços Web são usados apenas para exposição de *stored procedures* que podem manipular dados do BD, a arquitetura SODDA usa serviços Web como componentes fundamentais da comunicação entre nodos, como seria de se esperar em uma arquitetura orientada por serviços.

Como pode ser visto na Figura 2, o uso de serviços Web padroniza todo acesso aos bancos de dados. Todas as operações submetidas aos nodos do banco de dados passam pelo SODDA Hub, que é a única entidade da arquitetura SODDA que possui acesso ao catálogo global do BDD.

---

[1] Os nomes dos componentes da arquitetura SODDA apresentados dessa parte em diante estão em inglês para manter a compatibilidade com a implementação e com publicações relacionadas à dissertação e escritas naquele idioma.

[2] http://msdn.microsoft.com/en-us/library/ms722784(VS.85).aspx

[3] http://java.sun.com/products/jdbc/overview.html

[4] http://msdn.microsoft.com/en-us/library/ms710252.aspx

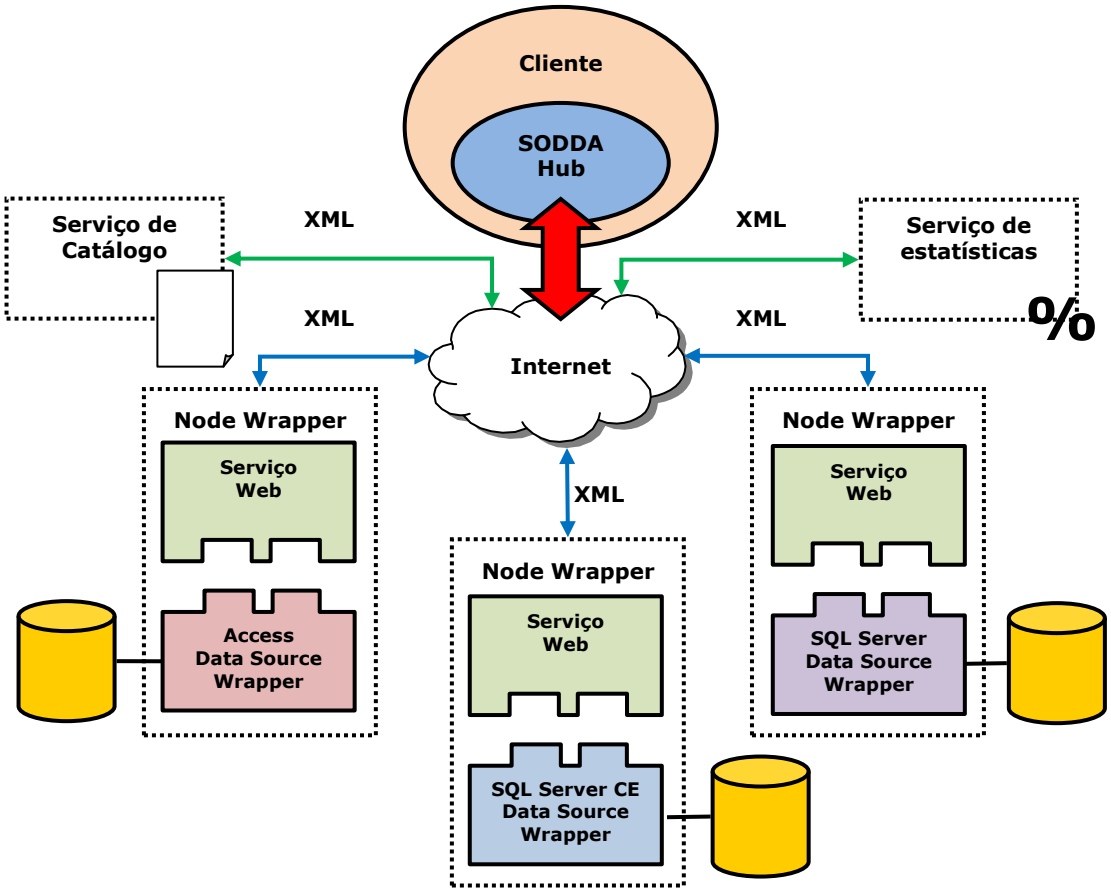

FIGURA 2. Funcionamento básico da arquitetura SODDA.

O acesso direto aos bancos de dados locais é encapsulado por dois elementos da arquitetura: os *Node Wrappers* e os *Data Source Wrappers*. Os serviços Web funcionam como *Node Wrappers*. Em todos os nodos de um ambiente distribuído deve existir um serviço Web para responder a invocações do cliente. Para executar comandos no contexto local de um nodo, o *Node Wrapper* deve usar um *Data Source Wrapper*.

Para o protótipo da arquitetura SODDA, foram desenvolvidos *Data Source Wrappers* para alguns SGBD (como SQL Server, SQL Server Compact e Access). Esses *wrappers* implementam métodos para adaptar e converter comandos SQL padrão em comandos de consulta específicos para cada fonte de dados. Se a fonte de dados é um banco de dados relacional, o comando SQL pode ser adaptado para uma variante específica de SQL, se necessário. Se a fonte de dados não for um SGBD, o comando SQL é convertido em um método de consulta associado à fonte de dados específica. Em fontes XML, por exemplo, comandos SQL podem ser convertidos em comandos XPath. Além disso, qualquer desenvolvedor pode criar outros *wrappers*, para outros tipos específicos de fontes de dados. O único requisito para isso é que eles devem seguir a interface disponível no provedor de dados da arquitetura SODDA, apresentado na Seção 4.1.

Em um SGBDD, o catálogo global armazena informações de localização dos fragmentos de dados. O gerenciamento e sincronização de suas cópias é um dos principais problemas na operação de um BDD. Na arquitetura SODDA, o catálogo global é substituído por um serviço Web, que mantém dados de localização de todos os *Node Wrappers* do ambiente de banco de dados distribuído. O uso de um serviço como catálogo global do BDD transforma a arquitetura SODDA em uma solução SOA completa, com transparência de localização.

O serviço de estatísticas é um repositório para o armazenamento centralizado de metadados relacionados a custos dos diversos bancos de dados envolvidos no ambiente BDD. Este serviço disponibiliza ao SODDA Hub parâmetros que serão utilizados no momento do processamento de consultas. Esses parâmetros incluem custos relacionados aos dados, tais como tamanho das tuplas e número de linhas nas tabelas, e custos de comunicação, como tempo médio de resposta e largura de banda da rede.

Após consultar o serviço de catálogo e o serviço de estatísticas para obter metadados sobre os nodos envolvidos em alguma operação sobre o BDD, o SODDA *Hub* precisa trocar documentos XML (envelopes SOAP comuns na comunicação com serviços Web) diretamente com os *Node Wrappers*, com o objetivo de recuperar dados para responder a uma requisição do cliente. Antes de realizar a consulta, no entanto, é necessário saber onde estão os fragmentos de dados. Em um ambiente distribuído, além dos dados estarem fragmentados, muitas vezes poderão existir cópias redundantes desses fragmentos. Portanto, é necessário manter um esquema conciso de fragmentação e replicação, apresentado na seção a seguir.

3.2 Fragmentação e replicação de dados na arquitetura SODDA

Independentemente das tecnologias envolvidas na construção de uma arquitetura de bancos de dados distribuídos, existe sempre um aspecto básico a ser tratado: os dados podem estar fragmentados. Essa fragmentação acontece para que seja possível dividir a carga de dados entre os vários nodos do ambiente distribuído, com intenções diversas, como a melhoria do desempenho no acesso a dados, a redução de períodos de inatividade do banco de dados e principalmente aumento de escalabilidade, pois os dados poderão ser acessados por mais clientes ao mesmo tempo.

Esses ganhos são potencializados quando, associada à funcionalidade de fragmentação, existe suporte a replicação de dados. A replicação permite que existam cópias idênticas de fragmentos de dados específicos e, dessa maneira, mesmo que um determinado

nodo que possui um fragmento de dado se torne indisponível, é possível manter o banco de dados funcionando, a partir de uma de suas cópias.

Na arquitetura SODDA, o cliente possui uma visão virtual do ambiente integral, como em um banco de dados centralizado convencional, como pode ser visto na Figura 3. Essa visão virtual é formada pela união dos fragmentos de dados (sejam eles horizontais, verticais ou mistos), que são encapsulados através dos *NodeServices*.

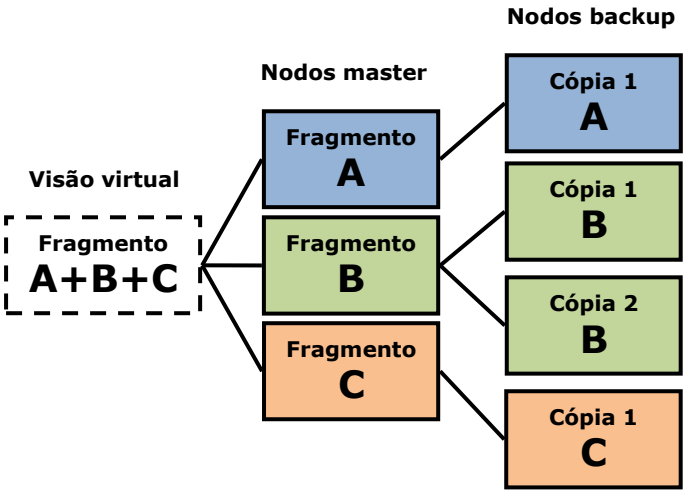

FIGURA 3. Esquema de fragmentação de dados da arquitetura SODDA.

Para cada fragmento de dado existe um nodo *master* e podem existir diversos nodos de *backup*. A cada iteração com o banco de dados, o *Replicated Data Manager* (RDM) é consultado para escolha dos nodos que participarão ativamente do processo. O RDM permite três estratégias de escolha de nodos: *AlwaysMaster*, *BestMatch* e *LoadBalance*.

Os nodos de *backup* são chamados em caso de falha de acesso ao nodo *master* ou em casos onde a estratégia de seleção do nodo privilegie o tempo de acesso. Dessa maneira, o esquema de fragmentação e replicação de dados da arquitetura SODDA é dinâmico, capaz de modificar a prioridade dos nodos no serviço de catálogo. Tal modificação pode ocorrer caso algum dos nodos se torne temporariamente indisponível ou apresente um aumento considerável no tempo de acesso, devido à sobrecarga de processamento ou outro problema qualquer.

De acordo com um temporizador pré-estabelecido, o serviço de estatísticas varre os nodos componentes do ambiente distribuído para atualizar sua tabela de tempo de acesso aos nodos, o que reclassifica automaticamente a lista contida no serviço de catálogo.

Na Figura 4, é apresentada a estratégia *AlwaysMaster*. Nessa estratégia o nodo *master* é sempre selecionado para atender à interação com o banco de dados. Se algum dos nodos *master* não puder atender a interação, ele é marcado como *offline* e substituído pela primeira

cópia disponível que esteja em estado *online*. Com isso, um nodo *backup* é automaticamente promovido a *master* e a interação é executada. Se ocorrerem mudanças no estado do banco de dados, os nodos *backup* são marcados como *outdated* e os nodos *master* ficam responsáveis por propagar as alterações. Assim que a alteração é registrada no nodo *backup*, ele passa ao estado *online* e se torna disponível para próxima interação.

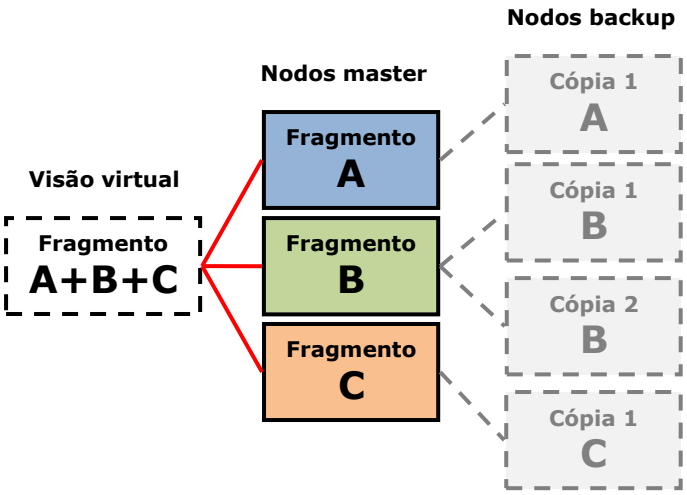

FIGURA 4. Estratégia de replicação *AlwaysMaster* da arquitetura SODDA.

Quando o nodo que estava com estado *offline* voltar, ele passará ao estado *outdated* e entrará em modo de recuperação. Dessa forma, ele requisitará ao *master* atual que seja atualizado e o *master* por sua vez enviará uma cópia diferencial de seus dados ao nodo que está marcado como *outdated*. Ao fim dessa operação de recuperação, o estado do nodo é alterado para *online* e ele também se torna disponível para a próxima interação.

A Figura 5 representa as estratégias *BestMatch* e *LoadBalance*. Na estratégia *BestMatch*, os tempos de acesso para os nodos são usados unicamente como parâmetro de escolha do nodo e, dessa maneira, sempre o nodo mais rápido recebe o trabalho a fazer. Com isso ele também será o único a propagar modificações de estado no banco e com o tempo isso pode prejudicar seu desempenho. A estratégia *LoadBalance* também utiliza o tempo de acesso como parâmetro de escolha do nodo, mas distribui as chamadas entre os melhores nodos e não escolhe somente o melhor[1]. Dessa maneira, um único nodo não ficará sempre com o trabalho de receber e propagar as alterações e, com essa sobrecarga reduzida, o desempenho global do banco de dados tende a melhorar.

[1] Existem várias estratégias para balanceamento de carga aplicáveis a servidores. Vide, por exemplo, Cardellini, V., Colajanni, M. e Yu, P. S. (1999). A escolha de uma estratégia para este caso está fora do escopo desta dissertação, e é proposta como trabalho futuro.

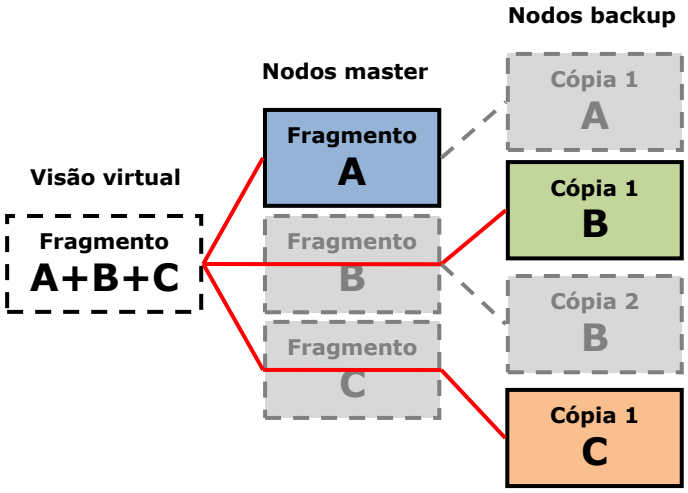

FIGURA 5. Estratégia de replicação *BestMatch* e *LoadBalance* da arquitetura SODDA.

Diferentemente da estratégia *AlwaysMaster*, o nodo que recebe a operação a ser realizada não é promovido a *master*. Isso evita que, ao efetuar operações simultaneamente, ocorram modificações desnecessárias no catálogo. Essas modificações poderiam gerar um *overhead* desnecessário, além de problemas diversos relacionados à sincronia de travamentos para leitura e modificação de entradas do catálogo.

A estratégia *BestMatch* pode exigir mais de um nodo, mas se usada em um ambiente de rede misto (LAN e WAN) pode oferecer ganhos interessantes. Naturalmente, as operações que estivessem dentro da LAN seriam realizadas mais rapidamente e o trafego na WAN seria reduzido durante a operação, pois o melhor nodo (ou seja, o nodo local) sempre teria melhor tempo de acesso e os nodos de *backup* só seriam contatados em outro momento, para propagação das modificações. Em um ambiente de rede unicamente WAN, a melhor solução provavelmente seria a estratégia *LoadBalance*, uma vez que o tráfego seria particionado e se daria em caminhos de dados diferentes.

Nos capítulos a seguir, serão apresentadas as funções de consulta e manipulação de dados distribuídos, que utilizam os esquemas de fragmentação e as estratégias de replicação da arquitetura SODDA.

3.3 Processamento distribuído de consultas na arquitetura SODDA

O processamento distribuído de consultas é uma das funções essenciais de um BDD, pois é utilizado em todas as situações de recuperação de dados que estão fragmentados entre diversos locais. O processamento distribuído de consultas é composto por diversas fases, que servem para dividir o trabalho em *tarefas locais*, que deverão ser executadas por cada nodo no

momento de uma consulta, e *tarefas globais*, que são executadas para consolidar a resposta a partir dos resultados parciais.

Na arquitetura SODDA, considera-se que os nodos do BDD são SGBD convencionais com autonomia de funcionamento local. Logo, esses nodos também possuem os mecanismos internos de um banco de dados convencional. Pensando nisso, e visando obter melhor desempenho, a arquitetura SODDA conta com um processador de consultas com otimização em dois níveis.

O primeiro nível de otimização trata dos aspectos de distribuição, baseado nos metadados do catálogo global e do serviço de estatísticas. O otimizador obtém estatísticas sobre as tabelas envolvidas em uma consulta a partir do serviço de estatísticas, de modo a traçar um plano de execução que evite o trafego de grandes massas de dados na rede. Isso é possível através de uma decomposição da consulta original, visando executar o mais cedo possível filtros que potencialmente diminuam o volume de dados que serão envolvidos nos estágios subseqüentes do processamento. Neste ponto, cabe observar que a elaboração do plano deve procurar fazer com que cada SGBD local possa avançar o máximo possível na execução da consulta usando recursos localmente disponíveis, em uma tentativa de diminuir o tráfego de resultados parciais na rede.

O segundo nível otimiza a consulta local enviada a um nodo, utilizando o mecanismo nativo de otimização de consultas do SGBD utilizado em cada nodo. Dessa maneira, parte do trabalho de otimização é aproveitado dos próprios SGBD envolvidos no ambiente BDD e a arquitetura SODDA pode se concentrar somente em aspectos de distribuição e consolidação.

O mecanismo de processamento de consultas da arquitetura SODDA é baseado no modelo básico exposto por Kossman (2000), mas devido à particularidade da otimização em dois níveis ele apresenta algumas variações. Na arquitetura SODDA, o mecanismo de consulta se divide em três partes, que são apresentadas a seguir.

Query Decomposition Engine (QDE). Como já comentado, a idéia do mecanismo de processamento de consultas da arquitetura SODDA não é baseada em um processador completo, mas sim em um *decompositor de consultas*, que conta com os mecanismos de processamento de consultas dos SGBD envolvidos no BDD. O decompositor é responsável pelo particionamento das consultas. O QDE deve descobrir os nodos envolvidos na operação, decompor a consulta em subconsultas com base no catálogo global, levando em consideração a forma de fragmentação e distribuição dos dados. O produto final do QDE é a criação de *query parts* (QP) que serão executadas em outra fase, nos contextos locais dos nodos do BDD.

Esta fase do processamento de consulta da arquitetura SODDA realiza o trabalho das duas primeiras fases da arquitetura de um processador de consultas de Kossman (2000), que são respectivamente o *parsing*, onde a consulta é traduzida para uma representação interna, e a *reescrita da consulta*, onde acontecem transformações como a eliminação de predicados redundantes e a simplificação de expressões.

Distribution Optimizer (DO). O DO deve levantar as tarefas que podem ser realizadas em paralelo, considerando que existem diversos processadores envolvidos (nos nodos) e preparando a junção dos resultados das subconsultas (que aqui serão denominadas *query parts*, QP). Essas junções são baseadas nas junções da consulta original, mas terminam o trabalho de otimização reorganizando os filtros, para evitar que grandes massas de dados trafeguem pela rede. O resultado do DO é o planejamento da execução da consulta através da montagem de uma árvore, com base nas dependências entre as QP. As QP sem dependências serão, portanto, os nós-folha da árvore.

Em um BDD, a estratégia de otimização é geralmente baseada em custos e, portanto, um importante fator a se considerar é o tráfego gerado pelos componentes da consulta. O plano de execução é fortemente influenciado pela necessidade de reduzir o tráfego de dados na rede. Devido a isso, o DO tem acesso ao serviço de estatísticas, que detém informações sobre custos de comunicação, tais como o tempo de resposta e a largura do canal de dados de cada nodo, além de custos de manipulação de dados, tais como a carga das tabelas e as faixas de valores contidos nos fragmentos de dados.

Esta fase do processamento de consulta da arquitetura SODDA realiza o trabalho referente a duas outras fases da arquitetura de um processador de consultas de Kossman (2000), que são respectivamente a *otimização da consulta*, onde são decididos os índices que serão utilizados para execução da consulta e o *refinamento do plano de consulta*, que especifica como a consulta será processada construindo uma árvore, em que cada nodo representa uma operação.

Distributed Execution Engine (DEE). Através de um caminhamento *bottom-up* na árvore gerada pelo DO, o DEE comanda a execução das QP em cada nodo. O DEE poderá identificar nós em um mesmo nível da árvore como passíveis de execução paralela e ordenar sua execução simultânea. Quando os resultados de uma QP são recebidos, o DEE cria dinamicamente uma representação do esquema global do ambiente distribuído e copia os resultados parciais. Para finalizar a consulta, o DEE realiza as junções sobre os resultados parciais e um banco de dados temporário, para gerar então o resultado final. Os resultados temporários podem ser manuseados no cliente (internamente no SODDA Hub), ou em

qualquer outro nodo, caso o cliente não tenha capacidade para armazená-los, como pode acontecer, por exemplo, em dispositivos móveis (Alves e Davis Jr. 2006).

Esta fase da arquitetura SODDA é equivalente à última fase da arquitetura de um processador de consultas de Kossman (2000), denominada naquele trabalho como *motor de execução de consultas*, responsável por disparar a execução das consultas de acordo com o planejamento traçado na fase de refinamento do plano de consulta.

O funcionamento do processamento distribuído de consultas da arquitetura SODDA pode ser visto na Figura 6. Na figura, uma consulta SQL padrão é entregue ao QDE, que a processa, descobre os nodos envolvidos na consulta, a particiona em QP e finalmente envia ao DO. O DO, por sua vez, monta o plano de execução da consulta através da construção de uma árvore onde as QP são os nodos. Por fim, o DEE executa o plano montado pelo DO, através de um caminhamento *top-down* na árvore do plano de execução. O resultado final é obtido somente após o termino do caminhamento do DEE, que, a cada nível processado, realiza uma junção com o resultado do processamento do nível anterior. Ao final do processo, o cliente recebe a resposta da sua consulta de forma transparente, sem precisar saber onde seus dados estão localizados fisicamente.

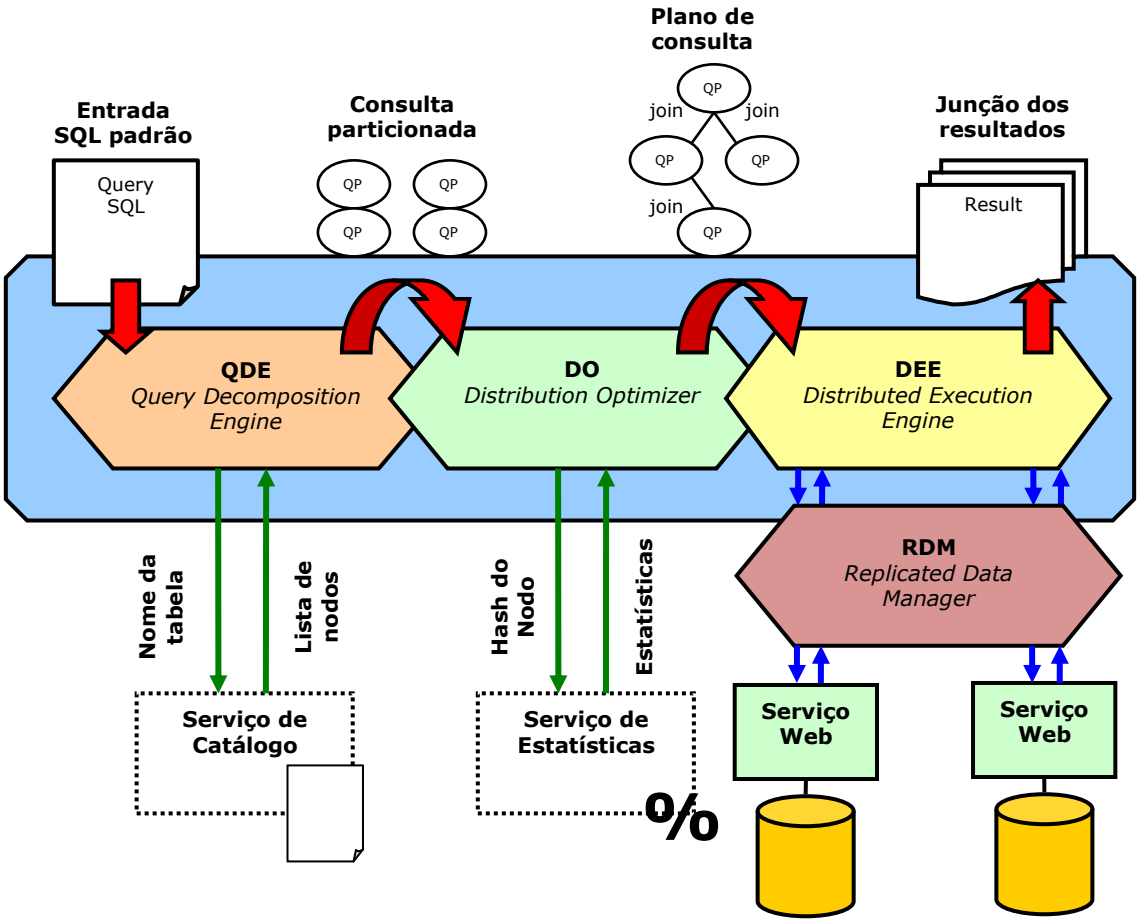

FIGURA 6. ***Distributed Query Processor.***

Nesse ponto, podemos dizer que a arquitetura SODDA, além de apresentar transparência de localização, também apresenta transparência de acesso, pois o acesso ao BDD é realizado virtualmente através de um ponto único, o SODDA Hub, que concentra os serviços necessários ao BDD. Na próxima seção serão apresentadas as demais funções do SODDA Hub, que dão suporte a manipulação de dados e controle transacional na arquitetura SODDA.

3.4 Manipulação de dados e controle transacional na arquitetura SODDA

A manipulação de dados e o controle transacional também são funções essenciais em um SGBDD. A manipulação de dados (inserção, alteração e remoção) é um elemento básico em qualquer tipo de SGBD. Em um SGBDD essa funcionalidade está fortemente acoplada ao controle transacional.

Em um SGBD centralizado, o controle transacional é geralmente utilizado em situações onde operações em diversas tabelas devem ser executadas ao mesmo tempo para manter o banco de dados em um estado consistente.

Em um ambiente distribuído, uma operação, mesmo que com apenas uma tabela, pode envolver diversos nodos. Isso acontece porque uma tabela pode ser fragmentada verticalmente e dessa maneira um simples comando precisará ser dividido em partes para que cada uma dessas partes seja executada em um local distinto.

Assim como no processador de consultas da arquitetura SODDA, considera-se que os nodos do BDD são SGBD convencionais com autonomia de funcionamento local. Se esses nodos possuem mecanismos internos que possibilitam manipulação de dados e transações, o trabalho da arquitetura SODDA consiste em particionar os comandos, descobrir para que nodos eles serão enviados e posteriormente controlar a execução global e as transações locais iniciadas em cada nodo, com intuito de garantir atomicidade da operação distribuída.

O gerenciador de transações distribuídas da arquitetura SODDA (*Distributed Transaction Manager* – DTM) é apresentado na Figura 7. Na figura, um comando é enviado ao componente decompositor de comandos (*Command Decomposition Engine*, CDE) que o particiona e envia ao coordenador de transações distribuídas (*Distributed Transaction Coordinator*, DTC). O DTC cria uma entrada na tabela de estados da transação (*Transaction Status Table*, TST) para a transação corrente e inicializa a execução. Um temporizador é inicializado e um *timeout* global é definido para a transação. Posteriormente, as operações são

enviadas paralelamente aos nodos envolvidos, juntamente com uma cópia da entrada na TST global.

Em cada nodo, uma transação local é inicializada na fonte de dados, o comando é executado e um temporizador é programado para cancelar a operação efetuada, caso a TST local não receba confirmação de sucesso em todos os nodos. O sucesso é registrado na TST local e propagado ao TST global do DTC.

O DTC recebe a confirmação do sucesso e a propaga aos outros nodos envolvidos. Se ele conseguir registrar sucesso em todos os nodos envolvidos, significa que os nodos ainda estão online e o sucesso global para operação é registrado.

Esse procedimento se repete até que o resultado de sucesso seja registrado em todas as TST locais, assim como na TST global. Dessa forma, ao marcar sucesso em todas as entradas da TST global, pode-se garantir que os sucessos foram registrados em todas as operações.

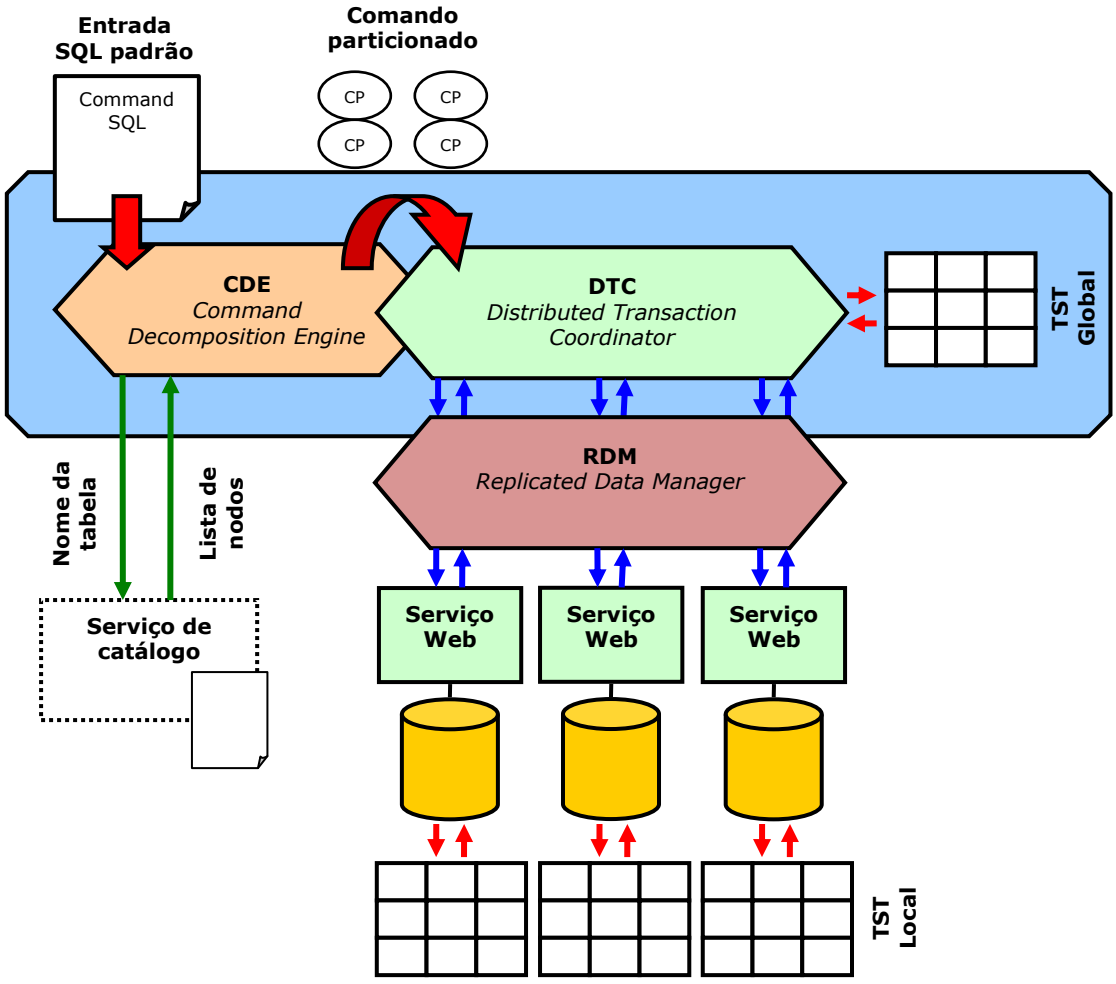

FIGURA 7. ***Distributed Transaction Manager*****.**

Em caso de falha durante a comunicação em qualquer fase do processo, o TST global não registra sucesso completo para a transação. Dessa forma, na expiração dos

temporizadores locais o TST global é consultado, e como existe uma falha registrada é realizado um *rollback* da transação local. Como os temporizadores locais foram iniciados em cascata, a falha em um nodo gera uma reação também em cascata, o que garante que os efeitos da transação distribuída sejam completamente anulados. Caso um nodo seja desconectado bruscamente, o gerenciador de transações do SGBD local se encarregará de cancelar a transação, pois o *commit* ainda não terá sido registrado.

Se não ocorrerem problemas, o TST global registra sucesso completo para a transação. Assim como no caso de falha, os temporizadores locais expiram e o TST global é consultado. Como não existem falhas registradas, ao invés de *rollback* é realizado um *commit* das transações locais. O *commit* é realizado em cascata, como no caso do *rollback*, pelo mesmo motivo: os temporizadores também são executados em cascata.

Ainda existe um caso particular, onde após registrar o sucesso global, e conseqüentemente os sucessos locais, um nodo é desconectado bruscamente. Nesse caso a transação será registrada nos nodos que se mantiverem *online* e será invalidada no nodo que foi desconectado. Para contornar esse problema, os nodos guardam as TSTs locais e os comandos pendentes de execução de forma persistente. Dessa forma, quando o nodo se reconecta ao sistema, ele entra inicialmente em modo de recuperação, buscando na TST global as operações efetuadas com sucesso que possuem correspondentes locais pendentes. Se existirem operações pendentes, o nodo as realiza na fonte de dados local, validando a transação. Esse problema não ocorre quando os nodos possuem cópias redundantes, pois no momento em que ocorre a falha de acesso, um dos nodos-filho assume o estado *master* e se responsabiliza pela sua própria atualização e de suas réplicas. No momento da recuperação, caso todos os nodos parceiros estejam marcados como *outdated*, entende-se que o estado atual do nodo em recuperação é o mais recente e que este nodo falhou antes de propagar suas modificações aos nodos de *backup*. Dessa forma, este nodo é promovido novamente a *master* e seu estado é propagado aos demais nodos de *backup* marcados como *outdated*.

Adicionalmente, a TST carrega a informação que indica que o comando foi efetivamente executado. Essa informação é enviada dos nodos para a TST global, após o *commit* efetivo. O DTC aguarda o recebimento de todas as notificações para, enfim, remover a entrada da transação corrente do TST global. Já os TSTs locais são limpos no momento em que enviam o sinal de *commit* efetivo ao DTC.

Observe-se que, considerando o mecanismo de manipulação de dados conforme descrito, não existe risco de concorrência no nível global. Qualquer conflito de dados que ocorra no nível global será transferido para o nível local, pois as transações concorrentes terão

as partes conflitantes encaminhadas para o mesmo nodo. O conflito será resolvido pelo SGBD local, usando seus próprios mecanismos de controle de concorrência.

O funcionamento básico do DTM da arquitetura SODDA é baseado no protocolo *Two-Phase Commit* (2PC) (Elmasri e Navathe 2005). Na Tabela 1 os passos de cada etapa do 2PC são comparados às ações do SODDA DTM.

Fase	**Protocolos de controle transacional**	
	2PC	**SODDA DTM**
1	Cada banco de dados participante informa ao coordenador que sua parte na transação foi concluída. O coordenador envia "preparar para efetivar" aos bancos de dados participantes. Participante recebe a mensagem e grava informações do log e de recuperação no disco. Logo após envia "ok" para o coordenador. Se houver falha na gravação no disco o participante envia "não ok" ao coordenador. Se em determinado tempo o coordenador não receber a mensagem de "ok", ele assume automaticamente "não ok".	O DTC delega *command parts* aos nodos do ambiente distribuído. Cada nodo recebe sua parte da transação, abre uma transação local, executa os comandos e envia "sucesso" ao DTC. Cada nodo inicializa um temporizador para desfazer as alterações realizadas, caso as outras operações registradas na TST global não registrem sucesso. O DTC recebe o "sucesso", propaga aos outros nodos e escreve na TST global.
2	Se todas as mensagens recebidas pelo coordenador forem "ok" e se o voto dele mesmo for "ok" a transação é bem sucedida e é enviado "efetivar" aos participantes. O banco participante efetiva a operação escrevendo "efetivado" no log. Se um "não ok" for obtido em qualquer etapa, o coordenador ordena a reversão em cada um dos participantes.	No momento que os temporizadores dos nodos expirarem, a TST global é consultada. Caso a TST possua somente sucessos registrados, a transação é considerada bem-sucedida e o comando de desfazer as alterações é cancelado. Caso contrário, o comando de desfazer é executado e a operação é considerada mal-sucedida.

TABELA 1. 2PC *versus* SODDA DTM

Enfim, com todos os mecanismos necessários a um SGBDD, resta apenas validar a proposta da arquitetura SODDA. No capítulo a seguir é apresentada a implementação de um protótipo da arquitetura, detalhes de seu funcionamento e a análise dos resultados obtidos através do trabalho de pesquisa.

4 PROTÓTIPO

Para validar a arquitetura SODDA, foi implementado um provedor de dados do *Microsoft .NET Framework*. O protótipo da arquitetura é composto por três componentes básicos: serviço de catálogo (*CatalogService*), serviço controlador de nodo (*NodeService*) e o provedor de dados (*DataProvider*). Como o objetivo do protótipo é testar e validar as idéias básicas da proposta, o serviço de estatísticas (*StatisticService*), voltado para a otimização do desempenho, não foi implementado.

Nas próximas seções são apresentados todos os componentes do protótipo implementado da arquitetura SODDA.

4.1 Provedor de dados

As classes do SODDA *DataProvider* são agrupadas em *namespaces*. Esses *namespaces* (*Hub*, *DistributedQueryProcessor*, *DistributedTransactionManager*, *Common e Wrappers*) podem ser vistos na Figura 8.

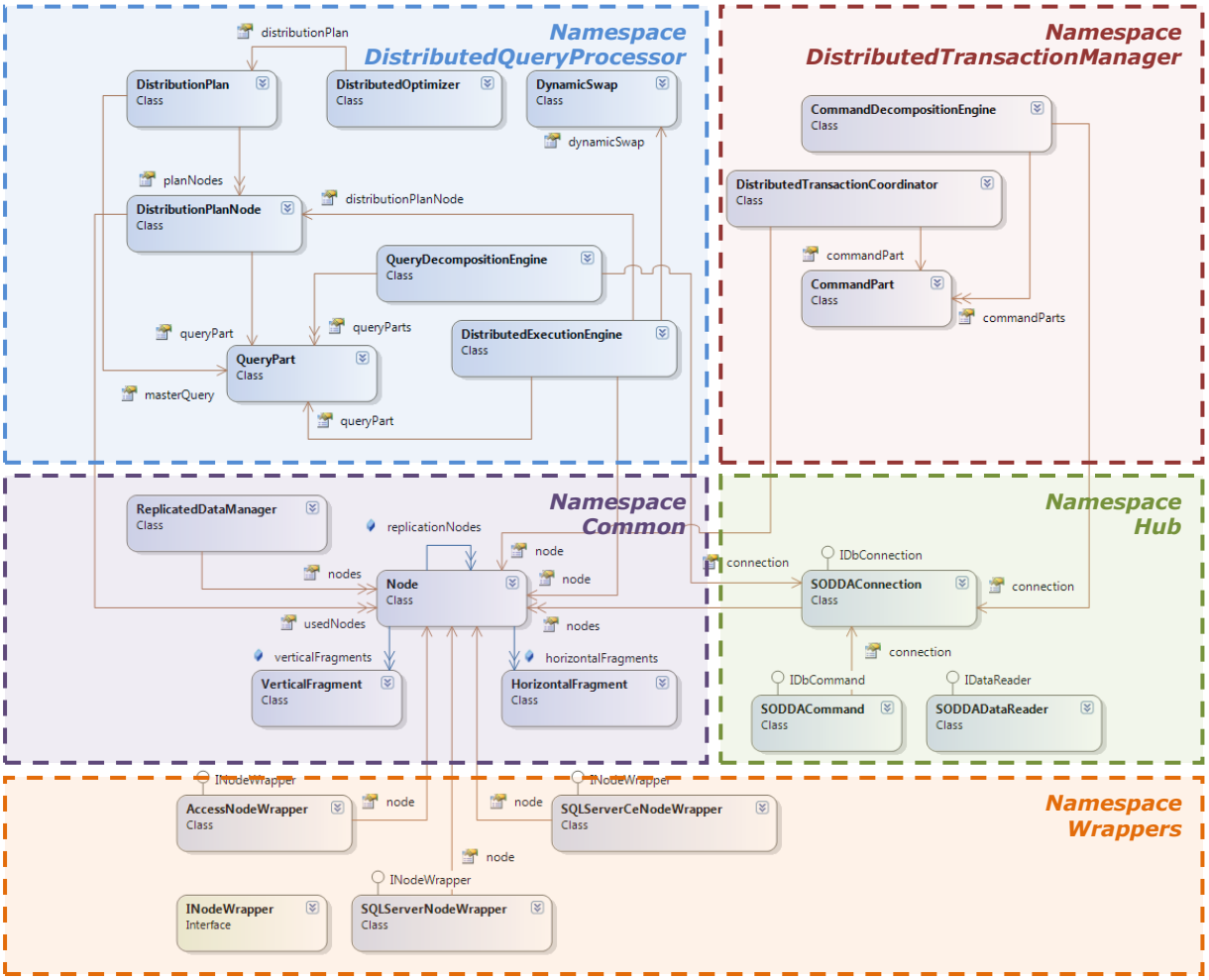

FIGURA 8. Diagrama de classes do protótipo da arquitetura SODDA.

Pertencem ao *namespace Hub* as classes de criação de conexões e execução de consultas no ambiente SODDA, tais como a *SODDAConnection*, *SODDACommand* e a *SODDADataReader*. Essas são as classes públicas utilizadas em uma aplicação baseada em SODDA. O compromisso com a transparência de utilização para o desenvolvedor do sistema é encapsulado por essas classes. A Tabela 2 demonstra a semelhança entre acessar dados através de um provedor de dados *OleDb* do *.NET Framework* e através do provedor de dados da arquitetura SODDA.

As classes do *namespace Hub* serão as únicas classes com as quais os desenvolvedores precisarão trabalhar. As demais classes são usadas internamente pela arquitetura e não foram desenvolvidas para serem manipuladas externamente, apesar de ser possível fazê-lo, como no caso da definição de um *Wrapper* para uma fonte de dados não suportada por padrão pela arquitetura SODDA.

Provider	**Script de Conexão**

.NET OleDb Provider	`Using conn As New OleDbConnection("Connection String")`
	`  conn.Open()`
	
	`  Dim cmd As New OleDbCommand("Script SQL", conn)`
	`  Dim dr As OleDbDataReader = cmd.ExecuteReader()`
	
	`  While dr.Read`
	`  'Leitura dos registros`
	`  End While`
	`End Using`

SODDA Data Provider	`Using conn As New SODDAConnection("Endereço do Serviço Catálogo")`
	`  conn.Open()`
	
	`  Dim cmd As New SODDACommand("Script SQL", conn)`
	`  Dim dr As SODDADataReader = cmd.ExecuteReader("Estratégia RDM")`
	
	`  While dr.Read`
	`  'Leitura dos registros`
	`  End While`
	`End Using`

TABELA 2. Acesso a dados no *.NET OleDb Provider* e na arquitetura SODDA

No *namespace DistributedQueryProcessor* são definidos classes e métodos internos para particionamento e planejamento de consultas, além da obtenção, manipulação e junção de resultados parciais. O particionamento da consulta é realizado pelo *QueryDecompositionEngine*, a definição e otimização do plano de consulta (*DistributionPlan*) são realizadas pelo *DistributionOptimizer* e controle de execução da consulta é realizado através do *DistributedExecutionEngine*.

No *namespace DistributedTransactionManager* são definidos classes e métodos também internos para execução de comandos SQL, que manipulam o estado dos dados armazenados nas diversas fontes de dados do ambiente distribuído. O particionamento do comando SQL é realizado pelo *CommandDecompositionEngine* e a distribuição e controle de execução dos *CommandParts* decompostos são controlados pelo *DistributedTransactionCoordinator*.

Tanto o *DistributedExecutionEngine* do *DistributedQueryProcessor*, como o *DistributedTransactionCoordinador* do *DistributedTransactionManager*, utilizam o *ReplicatedDataManager* no momento de escolher onde serão realizadas as operações sobre o banco de dados distribuídos.

No *namespace Wrappers* são definidas classes que encapsulam o acesso direto à dados na arquitetura SODDA. Esses *wrappers* seguem a *interface* apresentada na Tabela 3 e são usados no momento em que os dados são efetivamente acessados. O *CatalogService* e os *NodeServices* utilizam os métodos dos *wrappers* e encapsulam as chamadas desses métodos através de *WebMethods* dos serviços Web apresentados na seção a seguir.

Interface	**Código Fonte**

.NET OleDb Provider	`Public Interface INodeWrapper`
	`    ''' <summary>`
	`    ''' A node wrapper must implement a property to retrieve own`
	`    ''' configuration`
	`    ''' </summary>`
	`    Property node() As Node`
	
	`    ''' <summary>`
	`    ''' In order to query datasource, wrapper must implement a`
	`    ''' method to query against datasource`
	`    ''' </summary>`
	`    Function ExecuteQueryPart(ByVal sql As String) As DataTable`
	
	`    ''' <summary>`
	`    ''' In order to alter datasource, wrapper must implement a`
	`    ''' method to manipulate data against datasource`
	`    ''' </summary>`
	`    Function ExecuteCommandPart(ByVal sql As String) As Boolean`
	
	`    ''' <summary>`
	`    ''' A node wrapper must implement a method to retrieve node`
	`    ''' schema`
	`    ''' </summary>`
	`    Function GetLocalSchema() As DataSet`
	
	`End Interface`

TABELA 3. *Interface INodeWrapper*.

4.2 Serviço catálogo e serviço controlador de nodo

O SODDA *DataProvider* utiliza os *WebMethods* do *CatalogService* para obtenção dos endereços e parâmetros de acessos dos *NodeServices*. Com base nessas informações, *WebMethods* dos *NodeServices* são invocados para acesso aos dados do ambiente distribuído.

O *CatalogService* guarda as informações sobre localização de nodos, fragmentação e replicação em um arquivo de configuração XML, que segue o esquema da Figura 9.

Os *NodeServices* também possuem configurações em um arquivo de configuração XML, que também segue o esquema apresentado na Figura 9. Adicionalmente, no arquivo de configuração dos *NodeServices* são omitidas informações de localização do serviço e incluídas informações sobre o tipo de fonte de dados encapsulada e parâmetros de acesso a ela.

Utilizando métodos dos *wrappers*, o *CatalogService* e os *NodeServices* são os únicos componentes da arquitetura que realizam acesso real a dados e, com isso, encapsulam todo o acesso ao banco de dados. Todos os nodos do ambiente distribuído deverão ser encapsulados por um *NodeService*.

Para testar o protótipo, foram desenvolvidos *wrappers* para três tipos de SGBD: *SQL Server*, *SQL Server Compact Edition* e *Access*. Essa opção permite que os nodos rodem em diversos sistemas, incluindo *thin clients*. Foram desenvolvidos *wrappers* apenas para bancos de dados da Microsoft com intuito de facilitar a validação do protótipo, uma vez que existe uma integração facilitada entre a linguagem de programação utilizada no protótipo e os SGBD mencionados.

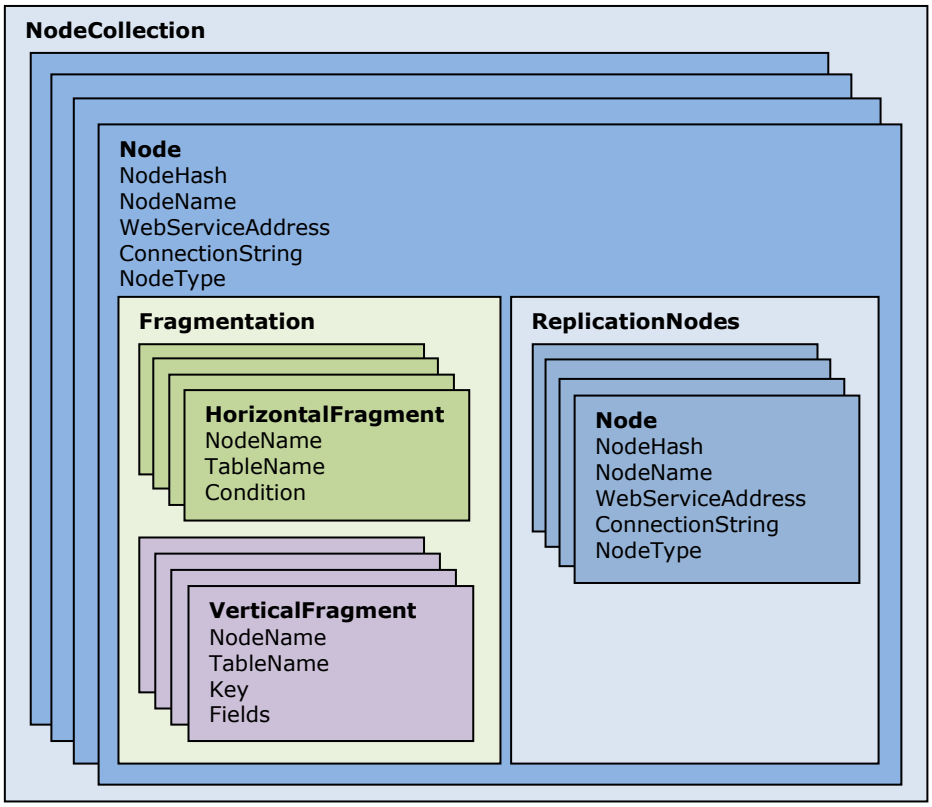

FIGURA 9. Esquema de arquivo de configuração XML para os serviços.

A escolha do *wrapper* necessário para executar uma determinada ação sobre um nodo é realizada dinamicamente através de reflexão. Através dessa técnica é possível declarar, por exemplo, um objeto do tipo *INodeWrapper* e instanciar um *SQLNodeWrapper* através de declaração implícita do tipo do objeto. Isso garante que a escolha do wrapper possa<<do que?>> seja realizada através de *wrappers* adicionais desenvolvidos independentemente, pois será necessário somente acrescentar a classe que implementa a interface *INodeWrapper* no *NodeService* e definir o parâmetro *NodeType* do arquivo de configuração do mesmo como o nome da classe acrescentada.

Na próxima seção são demonstradas as funções da arquitetura SODDA, através de diagramas UML (Booch, Rumbaugh e Jacobson 2005) específicos.

4.3 Funcionamento geral da arquitetura

O funcionamento geral da arquitetura SODDA é demonstrado no caso de uso apresentado na Figura 10. No caso de uso são apresentados os agentes envolvidos na execução da arquitetura, as funções projetadas e a interação entre elas.

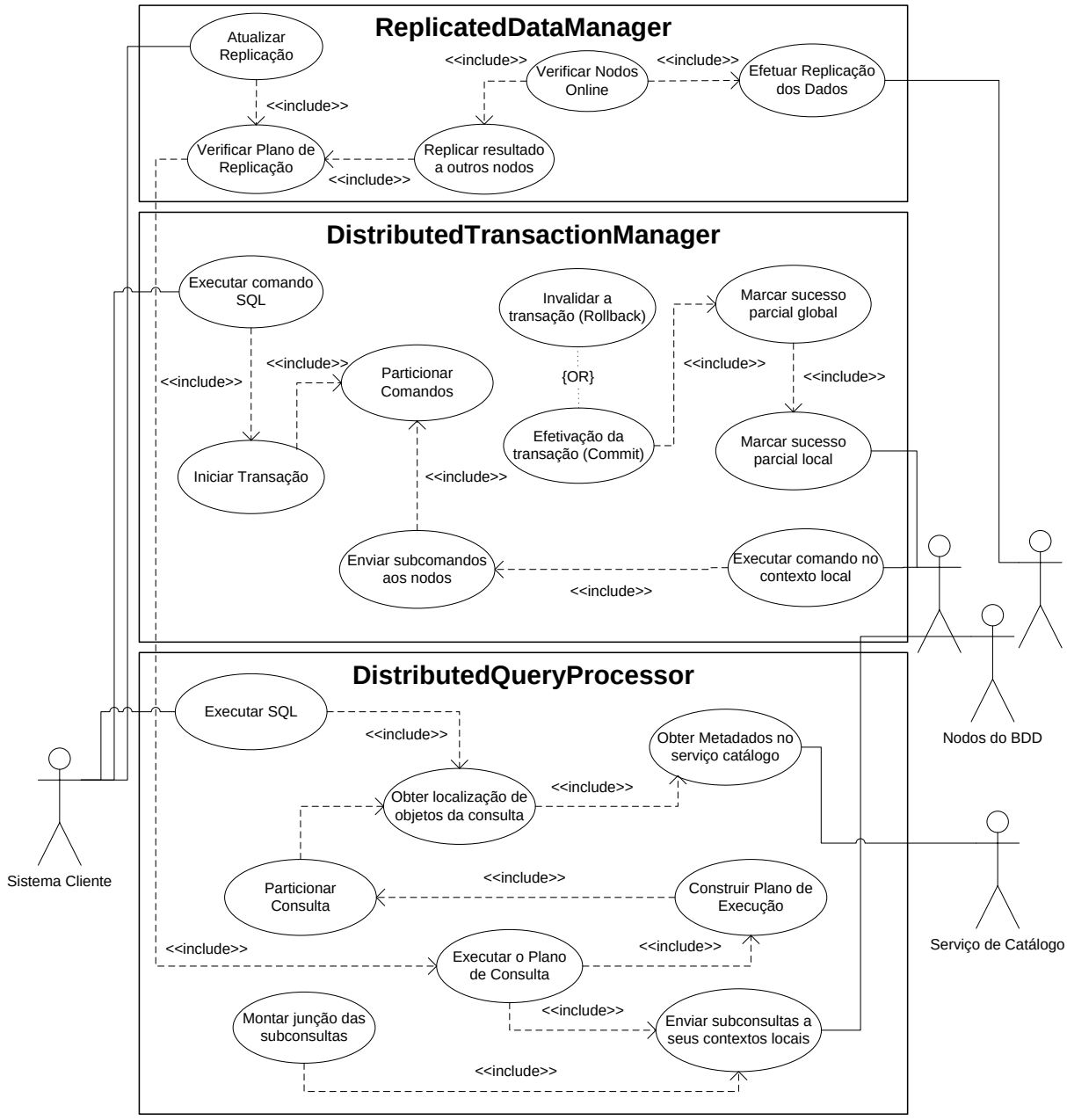

FIGURA 10. Diagrama de caso de uso da arquitetura SODDA.

O caso de uso mostra as funções de cada uma das partes do protótipo em uma visão macro. As interações entre essas funções são descritas através de relações *uses*. Os atores envolvidos em cada uma das funcionalidades também são devidamente destacados através do caso de uso.

As funções de consulta e manipulação de dados, que são provavelmente as mais importantes quando se pensa em um banco de dados, merecem um detalhamento mais aprofundado, que será apresentado nas seções a seguir através de diagramas de atividades.

4.3.1 Processamento de consultas

A seqüência de processamento de consultas no protótipo da arquitetura SODDA é apresentada na Figura 11. O diagrama de atividades mostra detalhadamente o fluxo de atividades durante o processo de consulta na arquitetura proposta, além de separar os escopos de execução dessas atividades.

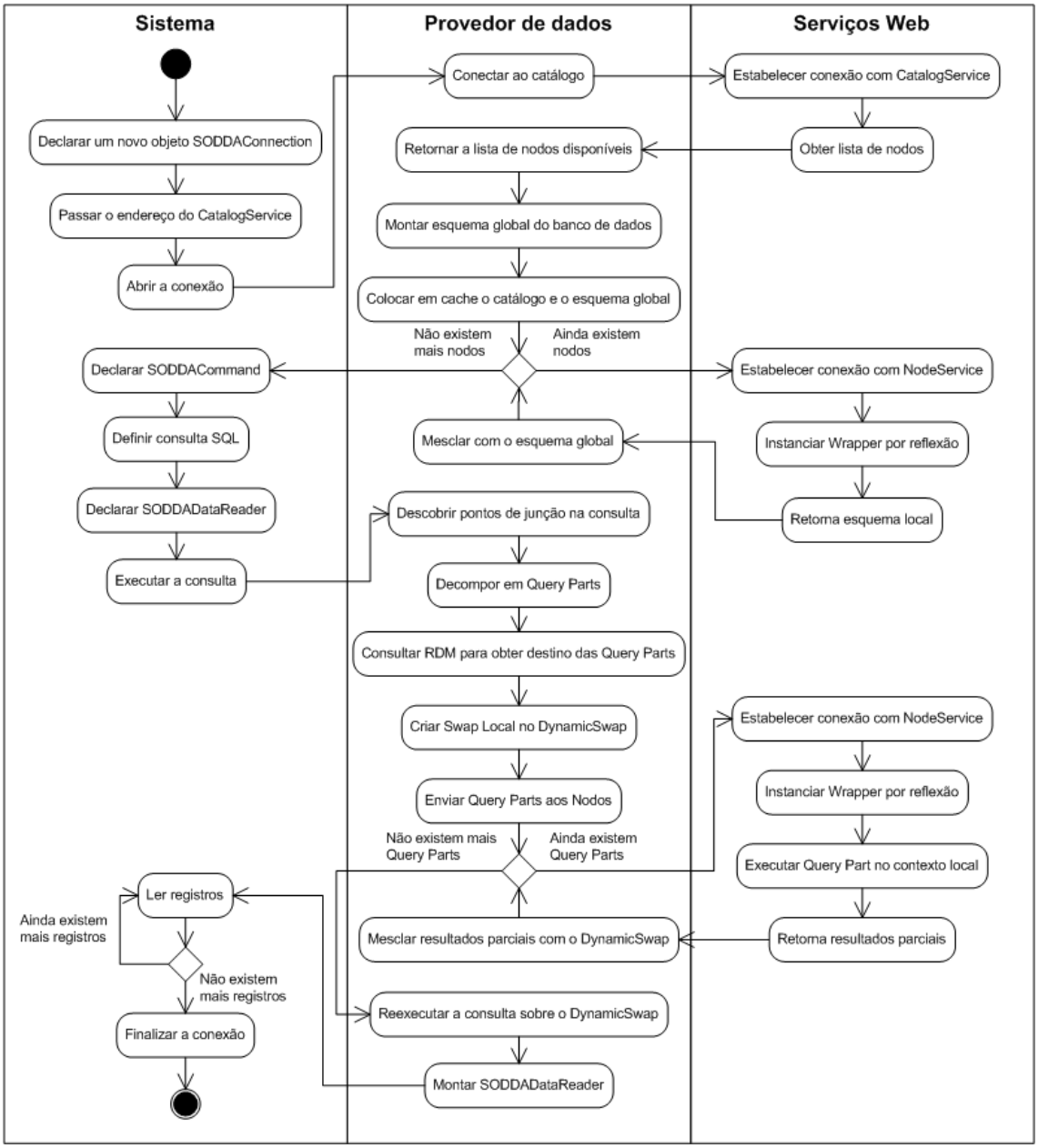

FIGURA 11. Diagrama de atividades do *DistributedQueryProcessor*.

No protótipo desenvolvido, o desenvolvedor declara um novo objeto *SODDAConnection*, passando somente o endereço do *CatalogService* e explicitamente abre a conexão. O *SODDAConnection*, por sua vez, estabelece conexão com o catálogo e retorna a lista de nodos disponíveis.

Em seguida, o *SODDAConnection* estabelece uma conexão com cada *NodeService* para retornar os esquemas locais das fontes de dados encapsuladas pelos *wrappers*. As configurações e os esquemas dos nodos são colocados em *cache* na memória da aplicação cliente.

O desenvolvedor declara um novo objeto *SODDACommand*, passando uma consulta SQL e a referência do *SODDAConnection* aberto. Logo após, ele declara um *SODDADataReader* e configura-o para receber o retorno da execução do método *ExecuteReader* do *SODDACommand.*

Usando informação do esquema, o *SODDACommand* analisa a consulta SQL e descobre os pontos de junção. Baseado nesses pontos, o *SODDACommand* constrói as *QueryParts*. Essas partes da consulta original serão enviadas e executadas nos *NodeServices* apropriados. O *SODDAConnection*, então, cria dinamicamente um espelho do esquema global no *DynamicSwap*.

Para cada *QueryPart*, o *SODDACommand* estabelece conexão com o *NodeService* apropriado e envia a consulta para obtenção de resultados parciais. No *NodeService*, o protótipo da arquitetura SODDA usa reflexão para instanciar o *wrapper* específico para a fonte de dados encapsulada. No fim da execução, o *NodeService* retorna os resultados parciais para o *SODDACommand*.

O *SODDACommand* mescla recursivamente os resultados parciais, retornados pelas execuções de *QueryParts* com os resultados já armazenados no *DynamicSwap*. No fim da obtenção e mixagem de resultados parciais, o *SODDACommand* retorna um *SODDADataReader* que permite leitura dos dados armazenados no *DynamicSwap.*

Finalmente, o desenvolvedor pode ler os registros através do *SODDADataReader* declarado, como em outro tipo de *DataReader* do *Microsoft .NET Framework*. No fim da leitura do *SODDADataReader*, ou quando o *SODDAConnection* for fechado, o *DynamicSwap* é destruído e o provedor de dados SODDA está pronto para outra iteração com a aplicação cliente.

4.3.2 Manipulação de dados e controle transacional

A seqüência de manipulação de dados no protótipo da arquitetura SODDA é apresentada na Figura 12. O diagrama de atividades mostra detalhadamente o fluxo de atividades durante o processo de consulta na arquitetura proposta, além de separar os escopos de execução dessas atividades.

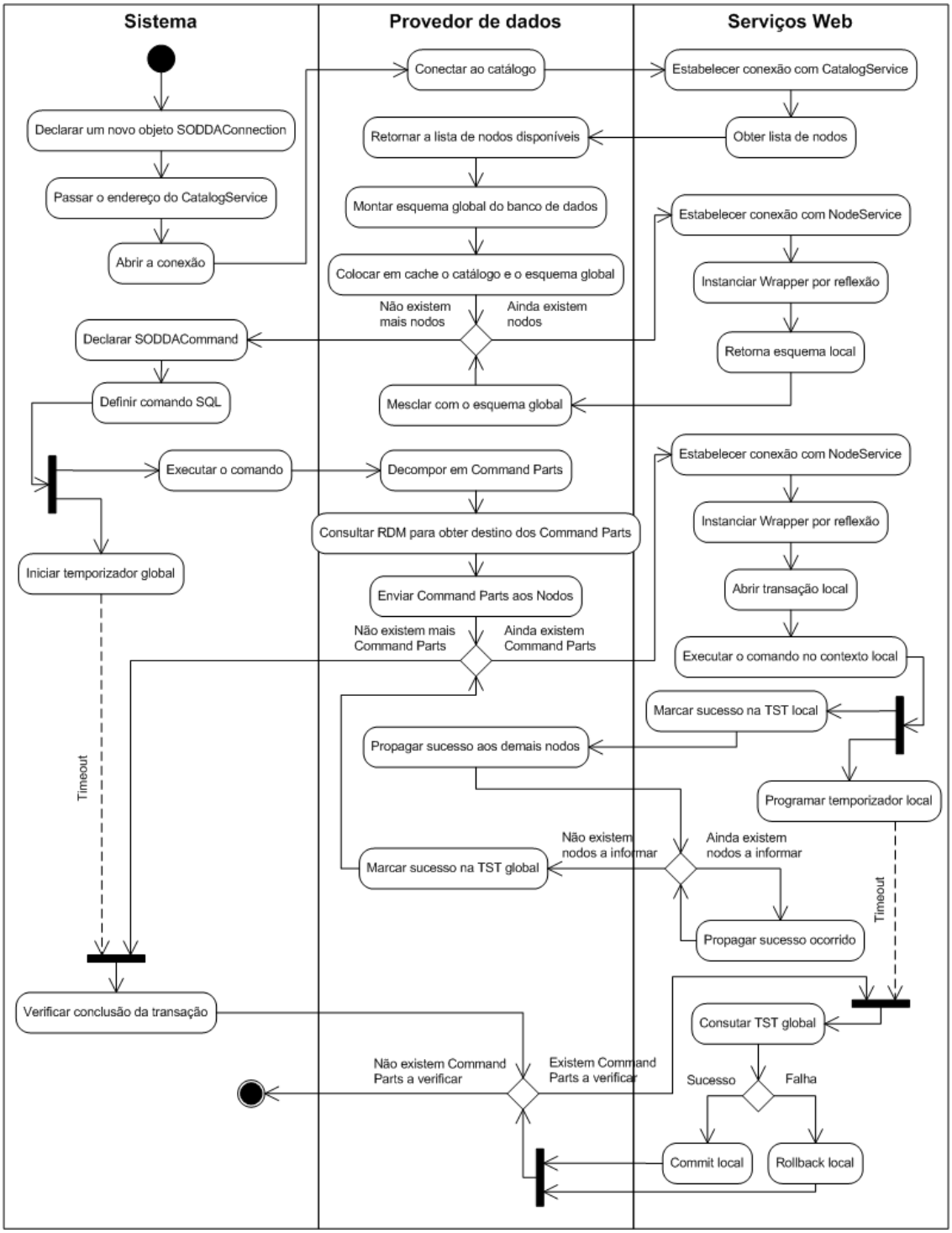

FIGURA 12. Diagrama de atividades do *DistributedTransactionManager*.

Até o momento em que a consulta é separada em *QueryParts*, o funcionamento do *DistributedTransactionManager* é idêntico ao funcionamento do *DistributedQueryProcessor*.

A partir de então, o desenvolvedor declara um novo objeto *SODDACommand*, passando não mais uma consulta SQL, mas sim um comando de manipulação de dados (*Insert, Delete ou Update*) e a referência do *SODDAConnection* aberto. Logo após ele solicita a execução do método *ExecuteNonQuery* do *SODDACommand.* Juntamente com o inicio da

transação distribuída, o *SODDACommand* inicializa um temporizador global para cancelar ou validar a transação em um determinado tempo.

Usando informação do esquema, o *SODDACommand* analisa a consulta SQL e descobre se mais de um nodo precisará participar da operação de manipulação de dados. Baseado nessas informações, o *SODDACommand* constrói os *CommandParts*.

Para cada *CommandPart*, o *SODDACommand* estabelece conexão com o *NodeService* apropriado e envia o comando para manipulação de dados em contexto local. No *NodeService*, o protótipo da arquitetura SODDA usa reflexão para instanciar o *wrapper* específico para a fonte de dados encapsulada. Uma transação local é inicializada e o comando, enfim, executado. Um temporizador local é inicializado para desfazer ou confirmar a transação, caso nenhuma informação adicional seja entregue ao nodo.

O *SODDACommand* repete essa operação até que todos os *CommandParts* sejam entregues aos respectivos nodos. No fim da execução, ou na expiração dos temporizadores, é inicializada uma verificação no estado global da execução da transação e, caso ela tenha sido realizada com sucesso em todos os nodos, um *commit* local é realizado em cada nodo. Caso isso não aconteça, um *rollback* local é então realizado em cada nodo.

Após a apresentação de todo o funcionamento da arquitetura, é necessário avaliar o funcionamento da arquitetura SODDA através de testes.

4.4 Testes de funcionamento

Para avaliar o funcionamento da arquitetura proposta, foi elaborado um projeto simples de banco de dados, que permite testar os conceitos apresentados e avaliar o funcionamento de alguns elementos básicos da arquitetura, como a comunicação via serviços Web, bem como os mecanismos de decomposição e coordenação de consultas.

O banco de dados usado no teste de funcionamento segue o diagrama apresentado na Figura 13. Esse banco de dados representa de modo simplificado a realidade de uma universidade, que possui um cadastro de alunos, matérias, avaliações e notas. O objetivo foi simular a distribuição de dados acadêmicos entre unidades fisicamente distantes entre si, como acontece na Pontifícia Universidade Católica de Minas Gerais. Para isso, o teste considera a distribuição de dados através das unidades Coração Eucarístico e São Gabriel da PUC, bem como a existência de um centro de processamento de dados, o DATAPUC, que possui dados comuns entre essas unidades.

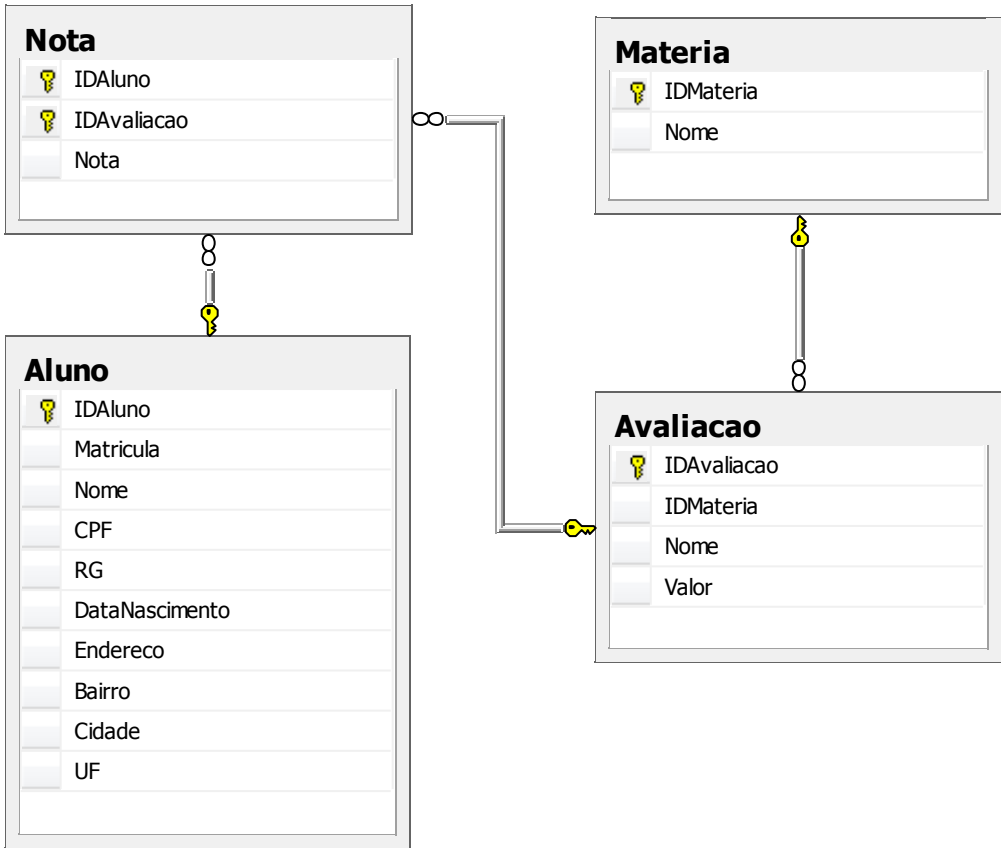

FIGURA 13. Banco de dados usado no teste de funcionamento da arquitetura SODDA.

Para realizar os testes, foi realizada, primeiramente, a carga de um banco de dados, como apresentado na Tabela 4.

Carga de tabelas				
Tabela	Materia	Avaliacao	Aluno	Nota
Linhas	5	20	21.908	438.160

TABELA 4. Carga de dados para teste de funcionamento da arquitetura SODDA.

Foram então configurados dois ambientes de distribuição. Na primeira configuração de distribuição a carga foi particionada em três nodos: nodo DATAPUC, onde foram alocados os registros de matérias e avaliações, nodo São Gabriel, onde foram alocados 10.000 registros de alunos e 200.000 registros de notas, e nodo Coração Eucarístico, onde foram alocados os 11.908 registros de alunos restantes, juntamente com os demais 238.160 registros de notas.

Na segunda configuração de distribuição, a carga foi particionada em cinco nodos: o nodo DataPuc , onde foram alocados os registros de matérias e avaliações, nodo São Gabriel, onde foram alocados 5.000 registros de alunos e 100.000 registros de notas, nodo Coração Eucarístico, onde foram alocados mais 5.000 registros de alunos e outros 100.000 registros de notas correspondentes, nodo Betim, onde foram alocados outros 5.000 registros de alunos e seus 100.000 registros de notas, e nodo Contagem, onde foram alocados os 6.908 registros de

alunos restantes, juntamente com os demais 138.160 registros de notas. Toda essa configuração é sintetizada na Tabela 5.

Tabela	**Banco de dados convencional**	**Banco de dados baseado em SODDA (3 nodos)**		**Banco de dados baseado em SODDA (5 nodos)**	
Materia	5	DATAPUC	5	DATAPUC	5
Avaliacao	20	DATAPUC	20	DATAPUC	20
Aluno	21.908	São Gabriel	10.000	São Gabriel	5.000
				C. Eucarístico	5.000
		C. Eucarístico	11.908	Betim	5.000
				Contagem	6.908
Nota	438.160	São Gabriel	200.000	São Gabriel	100.000
				C. Eucarístico	100.000
		C. Eucarístico	238.160	Betim	100.000
				Contagem	138.160

TABELA 5. Configuração do teste de funcionamento da arquitetura SODDA.

Para execução dos testes foram escritas três consultas SQL. A primeira consulta retorna todos os alunos, as matérias nas quais estão matriculados, avaliações e notas. Essa consulta retorna 438.160 registros. A segunda consulta agrupa as notas dos alunos e limita o universo de busca a uma só disciplina. Com isso, a consulta deve retornar 21.908 registros. A terceira consulta adiciona uma restrição à segunda consulta, filtrando os alunos que ficaram com menos de 60 pontos no somatório de notas, e com isso a consulta retorna apenas 2.318 registros. As consultas SQL estão expressas na Tabela 6.

As consultas foram então executadas no banco de dados convencional e nos ambientes distribuídos configurados, para análise comparativa. As consultas foram executadas com sucesso e apresentaram resultados idênticos, o que comprova que um banco de dados pode ser particionado e consultado através da arquitetura SODDA, e que o protótipo realiza as operações corretamente.

Configuração	Consulta SQL

Consulta 1: listagem simples de registros	`SELECT a.*,`
	`m.Nome,`
	`av.Nome,`
	`n.Nota`
	`FROM Aluno AS a INNER JOIN`
	`Nota AS n ON a.IDAluno = n.IDAluno INNER JOIN`
	`Avaliacao AS av ON n.IDAvaliacao = av.IDAvaliacao,`
	`Materia AS m`
	`WHERE m.IDMateria = av.IDMateria`
	`ORDER BY m.Nome, a.IDAluno`

Consulta 2: listagem com operação de agregação	`SELECT a.IDAluno,`
	`a.Nome,`
	`m.Nome,`
	`SUM(n.Nota) AS Nota`
	`FROM Aluno AS a INNER JOIN`
	`Nota AS n ON a.IDAluno = n.IDAluno INNER JOIN`
	`Avaliacao AS av ON n.IDAvaliacao = av.IDAvaliacao,`
	`Materia as m`
	`WHERE m.Nome = 'PAA'`
	`AND m.IDMateria = av.IDMateria`
	`GROUP BY a.IDAluno, a.Nome, m.Nome`
	`ORDER BY a.IDAluno`

Consulta 3: listagem com operação de agregação e restrição	`SELECT a.IDAluno,`
	`a.Nome,`
	`m.Nome,`
	`SUM(n.Nota) AS Nota`
	`FROM Aluno AS a INNER JOIN`
	`Nota AS n ON a.IDAluno = n.IDAluno INNER JOIN`
	`Avaliacao AS av ON n.IDAvaliacao = av.IDAvaliacao,`
	`Materia as m`
	`WHERE m.Nome = 'PAA'`
	`AND m.IDMateria = av.IDMateria`
	`GROUP BY a.IDAluno, a.Nome, m.Nome`
	`HAVING SUM(n.Nota) < 60`
	`ORDER BY a.IDAluno`

TABELA 6. Consultas SQL para o teste de funcionamento da arquitetura SODDA.

No entanto, problemas de desempenho são claros quando as consultas são realizadas utilizando a arquitetura SODDA. Foi constatado que o *overhead* de comunicação de serviços Web é consideravelmente alto, pois são introduzidas etapas que ocorrem de forma transparente para o desenvolvedor, mas que causam um impacto computacional considerável.

Toda informação trocada com o serviço Web é realizada através de documentos XML. Para isso, são introduzidas marcações que aumentam o tamanho dos registros. Um registro de nota, por exemplo, possui 17 bytes (4 bytes do campo idaluno, 4 bytes do campo idavaliacao e 9 bytes do campo nota). Só a marcação XML tem aproximadamente 68 bytes de tamanho.

Somando os dois valores, teríamos um total de 85 bytes, ou seja, aproximadamente cinco vezes o tamanho do registro sem a marcação XML.

Além de todo o custo já mencionado, ainda existem outros custos de mensuração mais difícil, como a montagem dos envelopes SOAP, além dos custos de requisição HTTP e de serialização dos documentos XML.

Para amenizar os impactos de desempenho causados pelos diversos componentes da arquitetura SODDA, a introdução do mecanismo de otimização é indispensável. Para este teste de funcionamento, as consultas foram executadas de forma canônica (não otimizada) e o tempo de resposta foi muito elevado, como pode ser visto na Tabela 7. Esse desempenho inviabiliza a utilização da arquitetura SODDA em aplicações reais. Somente a implementação completa da arquitetura, somada a uma análise mais profunda das questões de desempenho, poderá apontar a viabilidade ou não da solução.

	Consulta 1 (em ms)	**Consulta 2 (em ms)**	**Consulta 3 (em ms)**
Banco de dados convencional	38.000	1.000	3.000
Banco de dados baseado em SODDA (3 nodos)	166.890	69.407	62.235
Banco de dados baseado em SODDA (5 nodos)	250.991	76.892	67.651

TABELA 7. Análise simplificada do desempenho da arquitetura SODDA.

De qualquer forma, mesmo com os problemas de desempenho apontados, os resultados iniciais são promissores, pois os ganhos esperados com o mecanismo de otimização projetado podem diminuir consideravelmente o tráfego de dados, reduzindo a amplitude do principal problema encontrado na solução proposta. A prova disso é que a limitação do tamanho da massa de dados a ser retornada dos serviços componentes do ambiente distribuído reduz consideravelmente o tempo global do processamento da consulta.

Enfim, após o teste, resta-nos avaliar os resultados obtidos através do trabalho de pesquisa. O próximo capítulo encerra o trabalho, avaliando se os objetivos traçados foram atingidos, apresentando as lições que se pôde extrair da experiência, e realizando apontamentos para possíveis trabalhos futuros.

5 CONCLUSÕES E TRABALHOS FUTUROS

A arquitetura de banco de dados apoiada em serviços, proposta neste trabalho, emprega alguns dos mais interessantes recursos da orientação a serviços para implementar uma nova espécie de banco de dados distribuído, na qual os nodos podem ser SGBD independentes entre si e heterogêneos. A idéia central do trabalho envolve a substituição do aparato de comunicação requerido por um SGBDD por serviços Web, com troca de dados em XML.

Alguns dos ganhos esperados através da adoção da arquitetura SODDA incluem facilidade de implementação, devido à semelhança com métodos convencionais de acesso a dados em bancos de dados centralizados, baixo custo de comunicação e maior capilaridade de acesso, devido à utilização da Internet como infra-estrutura de comunicação.

A arquitetura SODDA pode ser considerada uma implementação completa da arquitetura referencial de SGBDD. Isso pode ser visto ao analisar os requisitos do modelo conceitual, frente aos recursos oferecidos na arquitetura. Todos os requisitos de transparência necessários são atingidos pela proposta, assim como os recursos mínimos necessários.

Devido ao uso de uma estrutura sólida de comunicação baseada em SOA e XML, os antigos problemas de comunicação dos SGBDD podem ser considerados resolvidos, o que torna a arquitetura SODDA em uma consistente alternativa para construção de bancos de dados distribuídos.

Como é esperado em soluções SOA, a adição de nodos é facilitada e as condições para inclusão de nodos redundantes são favoráveis. Até mesmo os serviços de infra-estrutura da arquitetura, como o serviço de catálogo e o serviço de estatísticas, podem em princípio ser replicados.

O serviço de estatísticas torna mais fácil a otimização e execução das consultas distribuídas, por unificar o tratamento de metadados de desempenho e permitir a implementação futura de um mecanismo de atualização automática de estatísticas através de um agente que vasculhe os nodos em períodos de baixa carga de trabalho.

Infelizmente, a aplicação real da arquitetura depende da finalização do protótipo com todas as funcionalidades previstas, principalmente o serviço de estatísticas, o que possibilitaria uma otimização baseada em custos mais eficiente e dessa maneira, um menor tráfego na rede. O modelo atual, sem otimização da árvore de consulta, se mostrou inviável no teste realizado, devido a problemas de desempenho. De qualquer forma, trata-se de um problema contornável e conhecido, que pode ser resolvido com pouco esforço. A limitação de

tempo do trabalho de pesquisa limitou os esforços dedicados à implementação do protótipo e nem todos os recursos puderam ser testados.

Com a escolha do *Microsoft .NET Framework* para implementação do protótipo foi possível otimizar o tempo gasto com o desenvolvimento de diversas partes do protótipo. Sua escolha foi importante para obtenção de resultados através dos testes do protótipo. No entanto, a maneira com que o .NET implementa suporte a SOA pode ter contribuído para perda de desempenho na troca de mensagens entre o *DataProvider* e os *NodeServices*. Como uma integração entre .NET e Java/J2EE é possível através da opção por SOA, pode ser interessante implementar e avaliar a adoção de serviços Web em Java rodando nos nodos do banco de dados distribuído. Isso inclusive excluiria a necessidade de um servidor Web robusto como o *Microsoft Internet Information Services*, hoje necessário em cada nodo do ambiente. Para utilizar SODDA em aplicações Java seria interessante reescrever o também o *DataProvider* em Java, pois assim a arquitetura proposta se tornaria automaticamente uma solução multi-plataforma.

Enfim, a arquitetura SODDA deixa uma extensa lista de possibilidades que podem ser trabalhadas, para criação de uma solução real baseada na arquitetura.

Primeiramente é necessário terminar a implementação da proposta e para isso o serviço de estatísticas é prioritário. O teste de funcionamento da arquitetura SODDA deixou claro que a execução não otimizada das consultas no ambiente distribuído é inviável, devido à transferência desnecessária de fragmentos de dados.

Após a implementação do serviço de estatísticas, é necessário um compreensivo teste de desempenho da arquitetura, com intuito de validação para aplicação da proposta em situações reais.

Estudos comparativos com arquiteturas do tipo cliente/servidor podem trazer resultados interessantes. Se a arquitetura SODDA apresentar desempenho ao menos próximo deste tipo de arquitetura, sua utilização pode-se tornar interessante, devido aos outros ganhos obtidos através da distribuição, como replicação e tolerância a falhas.

Além dos testes de desempenho, é recomendável avaliar o esforço de implementação de uma solução puramente SODDA, comparativamente às soluções Oracle e SODA. Os níveis de transparência obtidos pela arquitetura SODDA indicam uma implementação menos traumática.

Ao que tudo indica a arquitetura SODDA poderia ser utilizada para integração de bancos de dados utilizados em sistemas de informação em funcionamento dentro das

organizações. A avaliação dessa possibilidade é interessante para viabilizar a integração de sistemas de informação legados.

Pode ser interessante criar *Data Source Wrappers* para SGBD comerciais comuns como MySQL e Oracle. Essa tarefa não é complexa, pois requer a simples implementação da interface *Wrapper*. Isso fortaleceria o conceito de suporte a fontes de dados heterogêneas.

Também pudemos observar que os nodos do ambiente distribuído não precisam necessariamente ser controlados por um SGBD completo. Como eles são acessados através de um serviço Web, é possível ter outras fontes de dados agindo como nodos do SGBDD, tais como:

- Documentos de dados estruturados como arquivos XML e CSV, que são habitualmente utilizados como fontes de dados;
- Documentos semi-estruturados, tais como planilhas, documentos de texto e páginas Web.

No entanto, é importante lembrar que o *Data Source Wrapper* precisará implementar lógica própria para processamento de consultas e manipulação de dados, o que não acontece quando a fonte de dados é um SGBD completo, que possui seus mecanismos próprios de acesso a dados.

Outra possibilidade envolve a criação de um dispositivo de *hot-swapping* para nodos. Com ele seria possível garantir a retirada de um nodo para manutenção ou por problemas técnicos, sem exigir a reprogramação manual das entradas no serviço de catálogo.

É importante explorar a replicação de serviços Web, para evitar queda dos serviços de catálogo e de estatística, o que evitaria uma queda completa de todo o banco de dados distribuído, devido à falha de apenas um nodo (Ribeiro Jr., Valente e Carmo 2006).

Enfim, para melhorar o balanceamento de carga na arquitetura SODDA é importante um estudo mais aprofundado desses tipos algoritmos, o que garantiria um sistema ainda mais robusto e tolerante a falhas.

REFERÊNCIAS

Alonso, G., *et al.* (2004). **Web-Services: Concepts, Architecture and Applications**, Springer Verlag.

Alves, L. L. e Davis Jr., C. A. (2006). **Interoperability Through Web Services: Evaluating OGC Standards in Client Development for Spatial Data Infrastructures**. VIII Brazilian Symposium on GeoInformatics (GeoInfo 2006), Campos do Jordão (SP).

Booch, G., Rumbaugh, J. e Jacobson, I. (2005). **UML–Guia do Usuário**. Rio de Janeiro, Campus.

Bray, T., *et al.* (2006). **Extensible Markup Language (XML) 1.0 (Fourth Edition) - W3C Recommendation 16 August 2006, edited in place 29 September 2006**. Disponível em: http://www.w3.org/TR/xml/. Acesso em: 24/04/2007.

Buyya, R. (1999a). **High Performance Cluster Computing: Architectures and Systems, Vol. 1**, Prentice Hall Inc.

Buyya, R. (1999b). **High Performance Cluster Computing: Architectures and Systems, Vol. 2**, Prentice Hall Inc.

Campbell, D. (2005). **Service Oriented Database Architecture: App Server-Lite?** Proceedings of the ACM SIGMOD International Conference on Management of Data.

Cardellini, V., Colajanni, M. e Yu, P. S. (1999). **Dynamic Load Balancing on Web-Server Systems**. IEEE Internet Computing. **Vol. 03:** 28-39.

Curbera, F., *et al.* (2002). **Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI**. IEEE Internet Computing. **Vol 6:** 86-93.

Elmasri, R. e Navathe, S. B. (2005). **Sistemas de banco de dados**. São Paulo, Addison Wesley.

Endrei, M., *et al.* (2004). **Patterns: Service-Oriented Architecture and Web Services**, International Business Machines Corporation (IBM).

Ferris, C. e Farrel, J. (2003). **What Are Web Services?** Communications of the ACM. **Vol 46:** 31.

Foster, I., *et al.* (2002). **The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration**. Disponível em: http://www.globus.org/research/papers/ogsa.pdf. Acesso em: 20/05/2008.

Huhns, M. e Singh, M. P. (2005). **Service-Oriented Computing: Key Concepts and Principles**. IEEE Internet Computing. **Vol 9:** 75-81.

Kacsuk, P., Fahringer, T. e Németh, Z. (2007). **Distributed and Parallel Systems: From Cluster to Grid Computing**, Springer.

Kiely, D. (2006). **How SQL Server 2005 Enables Service-Oriented Database Architectures**. Disponível em: http://www.microsoft.com/technet/prodtechnol/sql/2005/sqlsoda.mspx. Acesso em: 20/06/2008.

Kossman, D. (2000). **The State of the Art in Distributed Query Processing**. ACM Computing Surveys. **Vol. 32:** 442-469.

Microsoft. (2007). **SQL Server Books Online (September 2007)**. Disponível em: http://msdn.microsoft.com/en-us/library/ms130214.aspx. Acesso em: 21/06/2008.

MySQL AB. (2008). **MySQL 5.1 Reference Manual**. Disponível em: http://dev.mysql.com/doc/refman/5.1/en/index.html. Acesso em: 21/06/2008.

Oracle. (2008). **Oracle Database Documentation Library 11g Release 1 (11.1)**. Disponível em: http://www.oracle.com/pls/db111/homepage. Acesso em: 18/06/2008.

Ozsu, M. T. e Valduriez, P. (1999). **Principles of distributed database systems**, Prentice Hall Inc.

Paton, N. W., *et al.* (2002). **Database Access and Integration Services on the Grid.** UK e-Science Technical Report Series.

PostgreSQL, G. D. G. (2008). **PostgreSQL 8.3.3 Documentation**. Disponível em: http://www.postgresql.org/docs/8.3/static/index.html. Acesso em: 21/06/2008.

Ribeiro Jr., J. G., Valente, M. T. O. e Carmo, G. T. S. (2006). **Smart Proxies para Invocação de Serviços Web Replicados**. Proceedings of the 12th Brazilian symposium on Multimedia and the web, Natal, Rio Grande do Norte, Brazil.

Tok, W. H. e Bressan, S. (2006). **DBNet: A Service-Oriented Database Architecture**. Proceedings of the 17th International Conference on Database and Expert Systems Applications (DEXA'06).

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo