2
Cenários de Execução de Consultas

Neste capítulo, apresentam-se os conceitos básicos relacionados com o processamento de consultas em SGBDs e, em particular, os conceitos relacionados com a execução de consultas. De acordo com estes conceitos, são descritos os cenários de execução de consultas introduzidos no Capítulo 1, bem como os principais trabalhos relacionados a estes cenários. Usaremos estes cenários como exemplos de extensibilidade da solução proposta neste trabalho de tese apresentada no próximo capítulo.

2.1.
Processamento de Consultas em SGBDs

O processamento de consulta em um sistema de gerência de banco de dados é realizado quando o usuário submete uma consulta a um SGBD, interativamente ou através de uma aplicação, utilizando uma linguagem de consulta de alto nível.

De uma maneira geral, o processamento de consultas envolve os seguintes passos:

a) *Tradução:* analisa sintaticamente e traduz a consulta para uma expressão interna equivalente;

b) *Validação:* valida a consulta com relação ao esquema do banco de dados (metadados) garantindo que a consulta contém somente referências válidas para objetos do banco de dados;

c) *Resolução de visão*: expande o grafo de consulta considerando as definições de visão;

d) *Otimização*: mapeia o grafo da consulta para um plano de execução da consulta otimizado de forma a minimizar as principais medidas de desempenho que são:

- Tempo de CPU, e quantidade de operações de entrada/saída
- Custos de memória (alocação máxima e a relação tempo x espaço necessário para alocação)

e) *Execução*: executa o plano de execução da consulta, gerado pelo otimizador, através de uma Máquina de Execução de Consultas (MEC).

Estes passos podem ser sub-divididos ou agrupados de acordo com a implementação do SGBD.

A adoção do processamento distribuído e/ou paralelo na maioria dos SGBDs faz-se necessária pelos principais motivos:

- Desempenho (*speed-up*): os SGBDs devem lidar cada vez mais com o aumento da complexidade das consultas e do tamanho do banco de dados.
- Escalabilidade (*scale-up*): devido ao crescimento do banco de dados, as empresas necessitam de uma maneira fácil de escalar seus sistemas (*hardware* e *software*) para acompanhar este crescimento.
- Disponibilidade (*availability*): significa manter o banco de dados o maior tempo possível em funcionamento para suportar sua utilização 24 horas x 7 dias em aplicações críticas tais como: bancos, cias. aéreas, comércio eletrônico, etc. Como há vários componentes semelhantes, pode-se explorar a replicação dos dados aumentando a disponibilidade e reduzindo o risco de falhas.

A seguir, apresenta-se uma visão geral do processamento de consultas nos seguintes SGBDs: SGBDs distribuídos, SGBDs paralelos, SGBDs XML e Sistemas de integração de dados na *web*

Os conceitos relacionados à execução de consultas serão apresentados no final deste capítulo.

2.1.1. Processamento de Consultas em SGBDs Distribuídos

Um banco de dados distribuídos é uma coleção de vários bancos de dados logicamente inter-relacionados e distribuídos por uma rede de computadores. A Figura 2 mostra um banco de dados distribuídos por vários nós (ou *sites*) de uma rede. Um Sistema de Gerência de Banco de Dados Distribuídos (SGBDD) é um sistema de *software* que permite a gerência do banco de dados distribuídos e que

torna a distribuição (de controle e de dados) transparente ao usuário (Ozsu&Valduriez, 1999). Sob o ponto de vista do usuário, um SGBDD deve se parecer exatamente como um SGBD não distribuído, no sentido de que o banco de dados distribuídos pode ser acessado por qualquer um dos nós. Segundo Date (Date, 2000), é desejável que um SGBDD possua certas características tais como:

- Grande autonomia dos SGBDs locais nos *sites*, os quais devem ser tratados como iguais e devem depender, o mínimo possível, de um *site* central;
- Processamento distribuído de consultas, ou seja, as consultas submetidas a um *site* devem ser otimizadas e podem ser executadas em outros *sites*, se necessário;
- Transparência no acesso a dados distribuídos (fragmentos e réplicas)
- Independência de plataforma de *hardware*, de sistema operacional, de rede e de SGBD local;

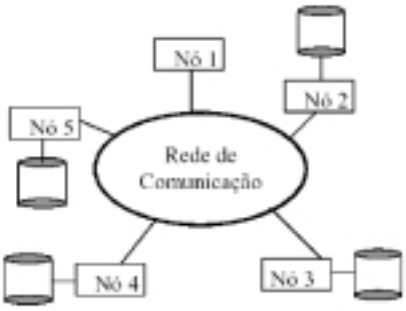

Figura 2 - SGBD Distribuído

Existem muitas arquiteturas propostas na literatura que implementam gerência de banco de dados distribuídos. Em Ozsu&Valduriez (Ozsu&Valduriez, 1999) utiliza-se uma classificação mais geral baseada em três dimensões: autonomia, distribuição e heterogeneidade. Atualmente, muitos sistemas se baseiam na arquitetura cliente-servidor, na qual existe um banco de dados armazenado num computador servidor e as consultas são iniciadas a partir de computadores clientes. Contudo, existem variações nesta arquitetura dependendo da cardinalidade do relacionamento entre clientes e servidores (Kossmann, 2000; Date, 2000). Na arquitetura cliente-servidor com transparência de distribuição, o cliente pode obter acesso a muitos servidores simultaneamente, isto é, cada

solicitação individual de banco de dados pode combinar dados de vários servidores, sendo possível dentro de uma única solicitação, combinar dados de dois ou mais servidores. Neste caso o usuário não precisa saber onde os fragmentos estão armazenados. Além disso, a gerência do processamento distribuído é de responsabilidade do servidor para o qual o cliente submeteu a sua consulta. Este servidor por sua vez, processará a consulta de duas formas possíveis:

- de forma centralizada – o servidor executa toda a consulta utilizando os dados dos outros servidores envolvidos na consulta, ou
- de forma distribuída – o servidor divide a consulta e submete as sub-consultas aos respectivos servidores de acordo com os dados envolvidos.

Atualmente, existem muitas pesquisas relacionadas às arquiteturas cliente-servidor com processamento distribuído, em particular, a arquitetura Ponto-a-Ponto (*peer-to-peer*) onde cada *site* pode agir como um servidor, processando uma consulta de forma distribuída, ou como um cliente, iniciando uma consulta.

Um dos principais objetivos do processamento de consultas distribuído é reduzir, ao máximo possível, o custo de comunicação (transporte de dados, de mensagens, etc) entre os *sites* envolvidos numa consulta. Este custo pode variar de acordo com a topologia da rede, mas passa a ser mais importante que os outros dois existentes apenas no processamento centralizado que são: custo de entrada/saída e de processamento. Para que o otimizador global decida em qual *site* cada operação será executada, ele deverá considerar algumas questões, tais como:

- Executar as operações em *sites* com maior poder de processamento mas sem sobrecarregá-los;
- Os dados devem ser processados em *sites* mais próximos possíveis dos seus;
- Deve-se balancear o volume de dados a ser transportado com o custo total de processamento (custo de transmissão mais o custo de processamento em cada site)

Outra questão no processamento de consultas distribuído é onde executar a consulta: no cliente (onde a consulta foi iniciada) ou no servidor (onde os dados, necessários para a consulta, estão armazenados). Isto se traduz na escolha de uma das seguintes abordagens:

a) transporte da consulta do cliente para o servidor, para ser executada no servidor (*query shipping*)
b) transporte dos dados do servidor para o cliente, para executar a consulta no cliente (*data shipping*)
c) combinar as duas abordagens (*hybrid shipping*)

A primeira abordagem é vantajosa quando o servidor possui grande poder de processamento e os clientes possuem pouco poder de processamento. Caso isso não ocorra, haverá um gargalo no servidor tentando atender às consultas provenientes dos seus clientes. A segunda abordagem pode ser vantajosa quando há disponibilidade de processamento nos clientes, porém, há um alto custo de comunicação na rede se não houver uma gerência eficiente de *caching* no cliente. A terceira abordagem sugere utilizar o que há de melhor das duas primeiras abordagens. A maioria dos SGBDs comerciais utiliza a primeira abordagem. Uma vez que o transporte de dados seja inevitável para a execução de um plano de execução (PEC) distribuído, caberá à MEC utilizar técnicas de execução apropriadas que diminuam, cada vez mais, do custo da execução distribuída. Essas técnicas, voltadas para a sincronização e o transporte de dados entre os *sites*, serão vistas no próximo capítulo.

2.1.2. Processamento de Consultas em SGBDs Paralelos

Um SGBD paralelo surge como alternativa para solucionar o problema conhecido como “gargalo de entrada e saída” induzido pelo alto tempo de acesso a disco se comparado com o tempo de acesso à memória.

Considerando-se que na literatura existe uma diversidade de definições entre SGBD paralelos (SGBDP) e SGBD distribuídos (SGBDD) relacionamos, a seguir, as principais diferenças:

- SGBDPs são fortemente acoplados (existe somente um ponto de controle, ou seja, existe somente um SGBD que divide uma consulta em fragmentos e os executam em paralelo) e SGBDDs são fracamente acoplados
- Em SGBDDs os dados encontram-se naturalmente distribuídos pelas suas instâncias. Já nos SGBDPs a distribuição dos dados depende da arquitetura do sistema.
- SGBDDs distribuídos são mais adequados para aumentar a robustez e a disponibilidade dos dados e os SGBDPs mais adequados a escalabilidade do sistema.
- Os SGBDDs endereçam as questões de autonomia, distribuição e heterogeneidade entre os componentes.

Um SGBD paralelo pode ser visto como um SGBD implementado sobre uma arquitetura de multiprocessador. As arquiteturas de multiprocessadores contêm dois ou mais processadores, os quais compartilham algum tipo de memória com o objetivo de diminuir a troca de mensagens entre os processadores. As principais arquiteturas de mutiprocessadores são (Ozsu&Valduriez, 1999; Garcia-Molina et al., 2000):

- arquitetura de memória compartilhada (*shared-memory*),
- arquitetura de disco compartilhado (*shared-disk*),
- arquitetura sem compartilhamento (*shared-nothing*) e
- arquitetura hierárquica (*cluster*).

Muitas pesquisas têm investigado o uso dessas arquiteturas para o suporte a SGBDs paralelos. Atualmente, com o baixo custo dos *clusters* de PCs, permitiu-se uma maior utilização desta arquitetura, em contraste com o custo das arquiteturas baseadas em *hardware* específico.

De uma maneira geral, o processamento de consultas paralelo envolve a atividade de dividir uma grande tarefa em sub-tarefas menores que podem ser processadas simultaneamente. O objetivo desta abordagem dividir-e-conquistar é completar todas as sub-tarefas em menos tempo do que seria necessário para completar a tarefa como um todo (Mahapatra&Mishra, 2000). Para se obter um

ganho real de paralelismo, as sub-tarefas são processadas em processadores distintos. Estes processadores podem pertencer a um mesmo computador ou a diferentes computadores.

A forma mais simples de paralelismo é o inter-consultas que ocorre quando utiliza-se múltiplos processadores para executar várias consultas de forma independente. Embora este tipo de paralelismo não aumente o desempenho de cada consulta individualmente – dado que cada consulta é executada por apenas um processador – ele aumenta o desempenho global de processamento para um grande número de consultas processadas. Por exemplo, em aplicações tradicionais (OLTP) cada consulta é executada de forma independente obtendo-se um tempo de resposta relativamente pequeno. Porém, na medida em que o número de usuários dessas aplicações aumenta, aumenta correspondentemente a quantidade de consultas a serem executadas simultaneamente. Sem a utilização deste tipo de paralelismo, essas consultas seriam executadas por apenas um processador através do processamento de tempo-compartilhado, diminuindo o tempo de resposta.

O tipo de paralelismo que nos interessa investigar neste trabalho é paralelismo intra-consulta, o qual possibilita dividir a consulta em partes que são executadas em paralelo usando diferentes processadores, um para cada parte. Como resultado, obtém-se um decréscimo do tempo total necessário para executar toda a consulta. Este tipo de paralelismo é bastante útil em consultas complexas e de longa duração. Pode ser implementado de duas formas:

- Paralelismo inter-operador – consiste na execução assíncrona de diferentes operadores de um mesmo PEC. Pode ser dividido em dois tipos:
  - Paralelismo Horizontal (Independente): as operações de uma consulta não dependem necessariamente uma da outra e são executadas em paralelo, ou seja, o resultado de uma operação *i* não é consumido pela operação *j*, permitindo a execução de *i* e *j* de maneira independente
  - Paralelismo Vertical (*Pipeline*): os dados produzidos por uma operação são consumidos por outras operações que deles precisam.
- Paralelismo intra-operador – ocorre quando o processamento de um mesmo operador é executado em múltiplos processos semelhantes, tipicamente usando conjuntos disjuntos de dados, ou seja, um

paralelismo baseado em partição de dados. Este tipo de paralelismo requer a reescrita dos algoritmos básicos (seleção, junção, projeção, etc) para cada arquitetura de *hardware*.

Os benefícios da execução paralela nem sempre surgem sem uma contrapartida. O sobre-custo produzido pela execução paralela de um processo concentra-se em três aspectos:

- Custo inicial – tempo necessário para criar os processos e associá-los a cada parte da consulta. Pode ser bastante significativo para pequenas consultas.
- Interferência – refere-se à desaceleração (*slowdown)* que um processo impõe sobre os demais quando acessam recursos compartilhados. Pode ser grande quando há muitos processadores.
- Desequilíbrio *(Skew)* – acontece quando o processo mais lento faz com que os demais o aguardem, aumentando o tempo total de processamento da consulta.

Cabe ao otimizador de consultas determinar quais operações serão executadas em paralelo e quando haverá ganho de processamento. Isto dependerá do tipo de arquitetura usada e das técnicas de execução empregadas pela MEC que implementa o modelo de execução paralelo.

2.1.3. Processamento de Consultas em Sistemas de Integração de Dados na Web

O objetivo de um sistema de integração de dados é prover uma interface uniforme de consulta para várias fontes de dados liberando o usuário da tarefa de localizar as fontes relevantes para suas consultas, interagir com cada fonte independentemente e combinar manualmente os seus resultados. Desta forma o usuário tem a ilusão de estar utilizando um único banco de dados centralizado.

A facilidade de acesso aos dados publicados na *web* e o grande volume de informações disponíveis nos *sites*, motivaram o crescimento de aplicações que necessitam integrar dados heterogêneos e distribuídos. Uma das aplicações é o

comércio eletrônico, onde organizações precisam integrar dados da *web* (no modelo XML) com dados de suas fontes (no modelo Relacional).

Historicamente o problema de integração de dados tem sido pesquisado no contexto de banco de dados distribuídos e resultou no surgimento dos SGBDs Federados e SGBDs Múltiplos que integram vários tipos de SGBDs e que, em geral, implementam todos os serviços de SGBDs tais como: gerência de esquemas, de memória e de transações.

Com o surgimento da *web*, o problema de integração tomou nova dimensão e novas questões surgiram (Abiteboul et al., 1999; Florescu et al., 1998) tais como:

- As fontes *web* possuem um alto grau de autonomia e, por isso, suas estruturas mudam com freqüência, dificultando a manutenção do acesso às fontes. O aparecimento do padrão XML tem reduzido este problema;
- As fontes *web* são essencialmente semi-estruturadas, fornecendo poucas informações (metadados) para a integração de esquemas;
- As fontes *web* são muito dinâmicas e são criados e destruídos sem o conhecimento de seus usuários, podendo acarretar problemas na visão global da integração;
- A quantidade de fontes *web* pode ser muito grande, trazendo problema para a integração de esquemas e para a resolução de conflitos;
- As fontes *web* podem ter capacidades limitadas de processamento, dificultando o processo de otimização e execução de consultas.

Na literatura são encontradas diversas soluções para o problema de integração de dados heterogêneos que são baseadas numa arquitetura genérica de três camadas definidas a seguir:

- Camada da aplicação: é representada pelas aplicações do usuário que submetem pedidos de consulta (ou atualização) sobre os dados das fontes de dados;
- Camada de integração *(middleware)*: oferece uma visão integrada das fontes de dados e disponibiliza um esquema para esta visão (esquema global) de forma que as consultas (ou atualizações) possam ser feitas através desse esquema.

- Camada das fontes de dados: é responsável pelo acesso às fontes de dados que podem variar entre SGBDs e arquivos de dados simples, até adaptadores para fontes *web*, dependendo da arquitetura de integração.

Existem várias questões que influenciam, de um modo geral, a arquitetura dos sistemas de integração de dados. As questões mais relevantes são:

- Modelo de dados global - deve ser rico o suficiente para capturar a semântica expressa nos modelos de dados das fontes. Na literatura são encontrados modelos de dados global baseados tanto em modelos estruturados quanto em modelos semi-estruturados.
- Esquema global - representa o esquema da integração segundo um modelo global.
- Materialização da visão integrada - existem duas abordagens para se obter a integração dos dados:
  - Abordagem materializada: os dados de várias fontes são carregados para um repositório (*data warehouse*) e todas as consultas são aplicadas sobre este repositório.
  - Abordagem virtual: os dados das fontes não são replicados e estão sempre atualizados no momento da consulta.

Não existe, na literatura, uma taxonomia única para classificar sistemas de integração de dados. Algumas classificações foram propostas em: Ozsu&Valduriez, 1999; Florescu et al., 1999; Abiteboul et al, 1999. Após o surgimento da *web*, muitos sistemas, tais como: TSIMMIS (Garcia-Molina, 1997), Le Select (LeSelect, 2003) e Mix (Baru et al., 1999), têm se baseado na arquitetura mediador-adaptador (*mediator-wrapper*,) proposta por (Wiederhold, 1992) e mostrada na Figura 3, onde :

- para cada fonte de dados existe um adaptador (*wrapper*) que exporta informações sobre seu esquema e suas capacidades de processamento.

- um mediador (*mediator*) centraliza as informações exportadas, pelos adaptadores, num dicionário de dados global e é responsável pelo processamento das consultas de integração.

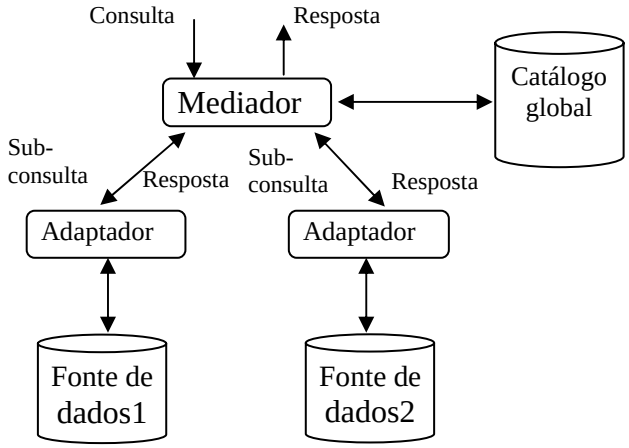

Figura 3 - Arquitetura Mediador-Adaptador

Devido à heterogeneidade dos dados, faz-se necessário criar diferentes adaptadores, uma para cada modelo de dados. A funcionalidade dos adaptadores pode variar desde simples roteador de pedidos de consultas aos SGBDs, até poderosas MECs sobre fontes com capacidades limitadas de execução.

Os mediadores podem ser desenvolvidos de forma independente e podem ser combinados permitindo uma maior escalabilidade das fontes de dados. A sua extensibilidade permite adicionar novos adaptadores sem a necessidade de se ajustar o mediador ou outros adaptadores existentes. Além disso, o mediador mantém um catálogo global usado no processamento de consultas, e que armazena:

- o esquema global de todo o banco de dados e que será usado no processamento das consultas dos usuários;
- o esquema externo exportado pelas fontes de dados, ou seja, qual parte do esquema global é armazenada por cada fonte;
- estatísticas para a otimização, caso existam.

Esta arquitetura tem como principal característica a simplicidade do *middleware* de integração pois, em geral, não implementam todos os serviços de

um SGBD devido ao fato de que a maioria das aplicações *web* somente realiza consulta sem atualização de dados.

A Figura 4 ilustra o processamento de consultas de integração baseado na arquitetura mediador-adaptador, que segue as seguintes etapas:

1. O mediador decompõe a consulta realizando as seguintes etapas:
   - Análise: baseando-se no catálogo global, seleciona as fontes de dados que podem contribuir para responder as consultas e verifica a disponibilidade das fontes selecionadas e sua capacidade de processamento;
   - Reescrita: converte a consulta (baseada no esquema global) para sub-consultas (baseadas nos esquemas locais das fontes de dados) as quais são enviadas para os respectivos adaptadores;
   - Otimização: visa selecionar um conjunto minimal de fontes para acessar e determinar a consulta mínima que deve ser enviada para cada um dos adaptadores. O otimizador deve selecionar PECs que explorem, da melhor forma possível, as capacidades de cada fonte de dados e evitar PECs contendo operações inválidas nas fontes de dados. Um dos problemas é a inexistência de estatística sobre os dados disponíveis na *web* e portanto, pouca informação para avaliar o custo de um PEC (Ives et al., 1999). Isso se deve ao fato das fontes *web* possuírem, em geral, uma autonomia e não terem sido projetadas para interagir com outros sistemas.
2. Os adaptadores traduzem as sub-consultas recebidas do mediador, do modelo de dados global para os seus modelos de dados locais, permitindo que sejam entendidas pela interface de suas fonte de dados. Após a execução das sub-consultas feita pelas fontes de dados (ou em parte, pelos adaptadores), os adaptadores retornam ao mediador os resultados gerados, já convertidos para o modelo global do mediador.

3. O mediador combina os resultados parciais recebidos dos adaptadores, produzindo o resultado final da consulta ao usuário. Além disso, caso seja necessário, o mediador executará algumas operações da consulta.

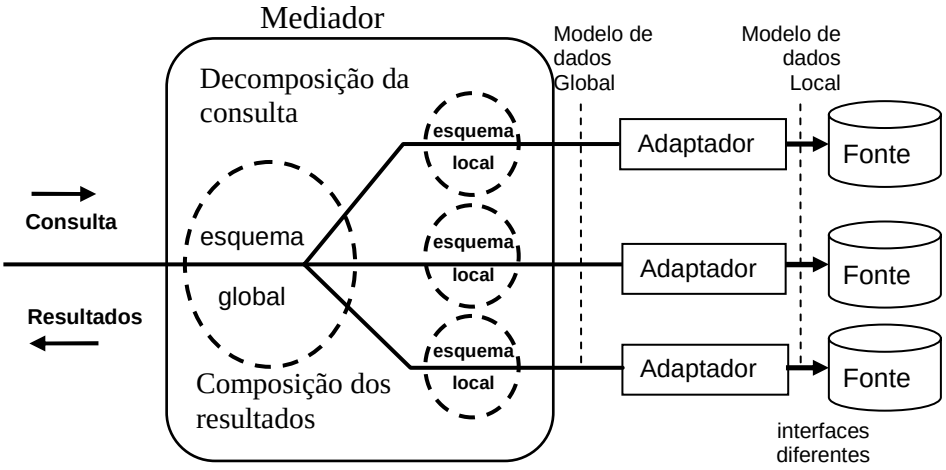

Figura 4 - Processamento de Consultas no Mediador-Adaptador

A construção de adaptadores não é uma tarefa fácil e, em alguns casos, o adaptador deve implementar algumas técnicas de execução, tais como *row blocking* e *caching*, para aumentar o desempenho. Além disso, alguns adaptadores podem ter capacidades limitadas, as quais restringem significativamente as possibilidades de execução de consultas.

Enquanto o processamento de consultas em SGBDs distribuídos tem sido bastante compreendido, isto não ocorre com os sistemas de integração de dados heterogêneos devido, principalmente, à maior dificuldade de otimização de consultas que, por sua vez, é devido ao fato de que os componentes são autônomos, possuem diferentes capacidades e geram custos que são mais difíceis de predizer.

2.1.4. Processamento de Consultas em SGBDs XML

XML *(Extensible Markup Language)* é uma linguagem de marcação padronizada pelo comitê W3C (W3C, 2003) e tem sido cada vez mais utilizada na *web* como formato para o intercâmbio de dados e de documentos. A linguagem XML difere da linguagem HTML basicamente em três principais aspectos:

- Novas marcações podem ser definidas a qualquer momento pelo usuário

- As marcações podem ser aninhadas em profundidades arbitrárias
- Um documento XML pode ser descrito através de uma gramática possibilitando a sua validação

Um documento XML é composto de elementos de marcação (tags) e texto e possui uma estrutura organizada hierarquicamente, aninhada e em forma de árvore. O Quadro 1 exemplifica um documento XML contendo dados sobre livros:

```
<?xml version="1.0" encoding="UTF-8"?>
<livros>
  <livro isbn="85-241-0590-9">
      <titulo>Proj. de Banco de Dados</titulo>
      <autor>Carlos A. Heuser</autor>
      <editora>Sagra Luzzato</editora>
      <edicao>2</edicao>
      <serie numero="4">Livros Didáticos</serie>
  </livro>
  <livro isbn="83-345-0336-2″>
      <titulo>Sistemas de Banco de Dados Cliente-Servidor</titulo>
      <autor>Rubens Nascimento Melo</autor>
      <autor>Asterio Tanaka</autor>
      <editora>Campus</editora>
  </livro>
...
</livros>
```

Quadro 1 - Exemplo de Dados XML

Um documento XML é dito “bem formado”, quando ele obedece às convenções léxicas (ex: tags iniciais casam com tags finais). Para verificar se um documento é bem formado utilizamos um analisador (*parser*) XML. Um documento é dito “válido”, se ele está de acordo com a sua DTD (*Document Type Definition*) ou com o seu esquema XML (*XML schema)*, que define as regras de sintaxe do documento. O Quadro 2 mostra a DTD que pode ser usada para validar o documento de livros acima.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT   livros (livro+)>
<!ELEMENT   livro (titulo, autor+, editora, edicao?, serie?)>
<!ATTLIST   livro isbn CDATA #REQUIRED>
<!ELEMENT   titulo (#PCDATA)>
<!ELEMENT   autor       (#PCDATA)>
<!ELEMENT   editora(#PCDATA)>
<!ELEMENT   edicao (#PCDATA)>
<!ELEMENT   serie       (#PCDATA)>
<!ATTLIST   serie numero CDATA #REQUIRED>
```

Quadro 2 - A DTD para o documento de livros

O acesso aos dados armazenados em documentos XML, a partir de uma aplicação, pode ser feito através de uma interface de programação (API), tais como as interfaces padrão:

- SAX *(Simple API for XML)* que fornece dados para aplicação através de eventos, disparados durante a análise do documento, e que são processados pela aplicação à medida que são analisados.
- DOM *(Document Object Model)* que disponibiliza os dados para a aplicação através de uma árvore, na memória, contendo todos os dados do documento.

O Modelo de dados XML proposto pelo W3C pode ser entendido da seguinte forma:

- Uma instância do modelo de dados é uma seqüência composta de zero ou mais itens;
- Cada item pode ser um nó ou valor atômico;
- Cada nó pode ser uma das opções: Documento, Elemento, Atributo, Texto, espaço de nomes (*namespaces*), instruções de processamento (PI) ou comentário;
- Um valor atômico pode ser tipado (stri*ng, boolean, ID, IDREF, decimal, URI*, etc) ou sem tipo.

Quanto às seqüências, elas podem ser heterogêneas, combinando nós e valores atômicos, por exemplo: (<a/>, 3); podem conter duplicatas, por: exemplo: (1,1,1); não estão na mesma ordem do documento. Além disso, as seqüências aninhadas são desaninhadas automaticamente: (1, 2, (3, 4)) = (1, 2, 3, 4). Quanto aos tipos dos valores atômicos, eles podem ser um dos tipos disponíveis no padrão *XML Schema* (*xs:integer, xs:boolean, xs:date, etc*) ou um dos tipos definidos pelo usuário. Os valores atômicos trazem consigo seus tipos: por exemplo o valor (8, meuNS:Comprimento) não é o mesmo que (8, xs:integer). Quanto aos nós, eles são identificados unicamente e são ordenados de acordo com a ordem do documento.

Sob o ponto de vista de SGBDs, o modelo de dados XML é um modelo semi-estruturado, onde a estrutura física dos dados de um documento é variável e

nem sempre é conhecida à priori, diferentemente dos modelos estruturados (relacional, orientado a objetos, etc) cuja estrutura dos dados é rígida e bem conhecida. Atualmente, existem duas principais abordagens para a construção de SGBDs XML que são:

- SGBDs XML nativos
- SGBDs XML baseados em extensões

Não existe uma definição precisa do que seja SGBD XML nativo e não-nativo. Segundo Chamberlin (Chamberlin et al., 2003), a principal diferença entre eles é que ambos tornam disponível uma interface XML para acesso aos dados, mas somente o primeiro armazena e recupera dados no formato XML. Qualquer que seja a abordagem utilizada, o sistema deve implementar um modelo de dados XML recomendado pelo comitê W3C.

O processamento de consultas em SGBDs XML é similar a dos relacionais. A grande diferença é a falta de tipificação. Segundo a especificação da linguagem Xquery (XQuery, 2003), o processamento de consultas em SGBDs XML é feito em duas fases:

- Análise da Consulta - prepara o PEC através da verificação sintática, de tipos e otimização, e
- Execução da Consulta - executa o PEC e faz a verificação dinâmica de tipos

Nestas fases é importante o conhecimento da informação dos esquemas para que possa ser feita a verificação de tipos. Por isso, antes de implementar essas duas fases num processador XML, deve-se conhecer a organização física e lógica do banco de dados, tais como definição dos esquemas e dos índices dos documentos armazenados no banco de dados. Neste contexto, surge uma questão importante: “o quão flexível é o sistema para permitir a consulta sobre documentos cujos esquemas (bem como índices) não são conhecidos *à priori* (como é o caso de *streams* de dados)”?

Em relação ao acesso aos resultados de uma consulta através de uma aplicação XML, é necessário o uso de uma Interface de Programação da Aplicação (API) para que a aplicação possa criar seus objetos a partir dos dados

XML obtidos do SGBD. Como visto anteriormente, as APIs DOM e SAX são usadas para se ter acesso aos dados de um documento XML.

2.2. Máquinas de Execução de Consultas (MEC)

Uma Máquina de Execução de Consultas (MEC) é responsável pela execução de um Plano de Execução de Consulta (PEC), em geral, fornecido pelo otimizador de consultas. Considera-se que uma MEC implementa:

- um conjunto de operadores, os quais o otimizador poderá escolher para gerar um PEC;
- tuplas de dados, processadas por esses operadores e contendo uma estrutura baseada no modelo de dados;
- estruturas de dados volumosas necessárias para armazenamento de tuplas de dados durante os seus processamentos;
- uma interface comum aos operadores, a qual permite a sua extensibilidade;
- um modelo de execução, o qual o otimizador se baseia para produzir PECs.

A seguir, detalharemos os elementos que compõem uma MEC.

2.2.1. Operadores

Os operadores de uma MEC podem ser agrupados em duas categorias:

- Operadores algébricos: representam os algoritmos de implementação das operações algébricas correspondentes ao plano físico da consulta;
- Operadores de controle: são meta-operadores que estabelecem a comunicação entre os operadores algébricos.

A principal diferença entre os dois grupos é o fato de que os operadores algébricos manipulam dados e os de controle simplesmente transferem os dados entre os operadores algébricos sem processá-los. A Figura 5 exemplifica o relacionamento entre operadores algébricos (op1 e op2) e de controle (c1) dentro

de um fragmento de PEC. Considerando-se que os operadores de controle estão diretamente relacionados com a implementação do modelo de execução de consultas, eles serão ilustrados em momento oportuno. A seguir descreveremos os operadores algébricos.

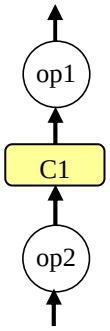

Figura 5 - Exemplo de operadores algébricos e de controle

Uma álgebra, no contexto de banco de dados, consiste num conjunto de operadores lógicos pertencentes a um modelo de dados, que definem quais consultas podem ser expressas neste modelo de dados (Garcia-Molina et al., 2000). Para que uma consulta possa ser otimizada e executada por um SGBD, os operadores lógicos devem ser implementados na respectiva MEC através de algoritmos que são denominados de operadores físicos da álgebra.

Embora os operadores lógicos de uma álgebra sejam os mesmos para vários SGBDs, os operadores físicos podem variar de um sistema para outro. Estas variações ocorrem devido às diferentes correspondências entre operadores físicos e lógicos, exemplificadas na álgebra relacional como: (a) os operadores físicos *mergejoin*, *nestedloop* correspondem ao operador lógico de junção (*join*); (b) o operador físico de ordenação *sort* não tem operador lógico correspondente, dado que a álgebra relacional é baseada em conjuntos (não ordenados);

O mapeamento entre operadores lógicos e físicos é realizado pelo otimizador e é, na maioria dos SGBDs, uma tarefa complexa (Selinger et al., 1979) pois envolve a escolha correta e eficiente dos operadores físicos que farão parte do PEC. Quanto mais operadores físicos para um mesmo operador lógico a MEC possuir, maior será o espaço de busca de possíveis planos do otimizador, devido ao fato que diferentes algoritmos geralmente possuem custos de execução diferenciados. Além disso, no mapeamento deve-se considerar também a simetria e a comutatividade entre os operadores. Alguns operadores lógicos são simétricos e comutativos mas alguns de seus operadores físicos não são. Por exemplo, o

operador de junção é simétrico, mas os algoritmos *mergejoin* e *nestedloop* são respectivamente simétrico e não simétrico.

Neste trabalho, referenciaremos os operadores físicos da álgebra como sendo operadores algébricos de uma MEC.

2.2.2. Tuplas de dados

As tuplas de dados são as unidades de dados de uma MEC que são processadas pelos seus operadores. Uma tupla[2] possui uma estrutura de dados extraída do modelo de dados subjacente a MEC. Por exemplo, uma tupla de dados relacional pode ser implementada como uma lista de atributos (nome, tipo e valor) ou pode ser implementada como um cursor sobre uma tabela (linha e coluna). Já uma tupla de dados XML pode ser implementada através de uma estrutura hierárquica. Em geral, a estrutura da tupla depende da forma de armazenamento dos dados e do seu gerenciamento de memória.

2.2.3. Estruturas de Dados

Refere-se à utilização de estruturas de dados volumosas (*bulk types*) tais como listas, árvores e sacolas, utilizadas pelos operadores da MEC para armazenar temporariamente as tuplas processadas por eles. Essas estruturas podem ser definidas *à priori*, vinculadas a implementação dos operadores, ou *à posteriori*, na instanciação dos operadores durante a execução, como é o caso da biblioteca XXL (Bercken et al., 2001) que permite combinar diferentes operadores com diferentes estruturas de dados. Por exemplo, um operador de junção *MergeJoin* pode utilizar diferentes tipos de listas para ordenar os dados.

Da mesma forma que há diferenças entre operadores lógicos e físicos da álgebra, existem estruturas de dados lógicas e físicas do sistema. Por exemplo, no nível lógico podemos citar as estruturas, tupla, relação, conjunto e pilha, que no nível físico podem ser implementadas através de *arrays*, listas encadeadas ou

[2] O termo “tupla” é utilizado neste trabalho de forma genérica, para qualquer modelo de dados.

mesmo, arquivos de bytes. O mapeamento das estruturas de dados lógicas, em estruturas físicas, faz parte do projeto físico de um SGBD.

2.2.4. Interface dos Operadores

A construção de um PEC exige que os operadores da MEC relacionem-se formando pares consumidor-produtor. Para isso, faz-se necessário implementar esses operadores usando uma interface comum. Todo o estado da arte sobre MECs está baseada na interface denominada iterador (*iterator*) (Graefe, 1993). Esta interface apresenta as três operações:

Open: prepara o operador para produzir dados;

Next: produz um dado para o operador consumidor;

Close: finaliza a execução do operador.

A interface favorece a extensibilidade dos operadores, já que a implementação de um operador não se necessita conhecer o operador para o qual produzirá ou do qual consumirá dados. Como conseqüência, um PEC pode ser escalonado em qualquer tipo de árvore e nenhum operador é afetado pela complexidade de um outro operador ou do PEC como um todo. Cada operador do PEC é escalonado pelo seu consumidor e escalona seu(s) produtor(es) através das chamadas aos métodos *open*, *next* e *close.* Através desta interface, cada operador (algébrico ou de controle) deve implementar seu algoritmo baseando-se nestas três operações. Por exemplo, o Quadro 3 mostra uma implementação do operador de junção NestedLoop baseado nesta interface.

```
/*  R e S são as entradas  */
/*  r,s,t são tuplas definidas a partir de um modelo de dados  */
Open  (R,S) {
      R.Open () ;  S.Open () ; s := S.Next () ;
}
Next (R, S) : Tupla {
      REPEAT
        r := R.Next () ;
        If (r = null)    /* R está esgotado para  o valor atual de s */
        {
                R.Close () ;
                s := S.Next () ;
                If (s=null) RETURN null;     /* R e S estão ambos esgotados */
                R.Open () ;
                r := R. Next () ;
        }
      UNTIL ( pode_juntar(r,s) ) ;
```

```
        t := juntar (r,s);
        RETURN t ;
    }
    Close (R,S) {
        R.Close () ; S.Close  () ;
    }
```

Quadro 3 - Implementação da interface iterador para o NestedLoop

Tal forma de implementar operadores tem sido utilizada por SGBDs tradicionais, sistemas de integração de dados na *web* (por exemplo: Tox (Barbosa et al., 2001)) e SGBDs XML (por exemplo: Tamino (Shoning, 2001)).

A implementação da interface iterador pode sofrer algumas variações, tais como: (i) o método *open* pode ser opcional ou usado apenas para produzir o primeiro dado do operador; (ii) o método *next* pode antecipar a produção de vários dados (bloco) e (iii) a indicação de término dos dados pode ser feita por um método específico (por exemplo: *hasnext* (Bercken et al. 2001) ou pelo próprio método *next* retornando valor nulo. Além disso, pode ser conveniente que as operações da interface possam receber (via parâmetro) o operador responsável pela sua chamada, para que seja possível executar alguma tarefa (Waas, 1999). A interface iterador pode ser adaptada para outras interfaces tais como a interface *ResultSet*, nativa da linguagem Java.

2.2.5. Planos de Execução de Consultas (PEC)

Um PEC é composto por um conjunto de operadores que se relacionam de forma a produzir o mais cedo possível, os resultados da consulta submetida ao SGBD. A maioria dos SGBDs representam um PEC através de uma árvore, onde os nós são operadores, as folhas são as fontes de dados, e os arcos são os relacionamentos entre os operadores na forma produtor-consumidor. Portanto, cada operador de um PEC exerce o papel de produtor de dados e/ou de consumidor de dados A Figura 6 ilustra um PEC contendo operadores (algébricos e de controle) relacionados na forma produtor-consumidor, onde as setas indicam o fluxo dos dados no sentido produtor-consumidor.

Na geração de um PEC, o otimizador escolhe a melhor combinação de operadores algébricos obtendo assim um PEC intermediário. Para completar o PEC final, é necessário incluir operadores de controle, permitindo isolar os

operadores algébricos de detalhes de implementação, tais como: leitura de disco, transferência pela rede, materialização dos dados, comunicação entre processos, etc.

A abordagem de utilização de operadores de controle proporciona, aos algoritmos dos operadores algébricos, a independência de implementação uma vez que eles podem ser concebidos sem a preocupação com o ambiente de implementação (Graefe, 1993). Os operadores de controle estão associados às técnicas de execução adotadas pelas MECs, como veremos mais adiante.

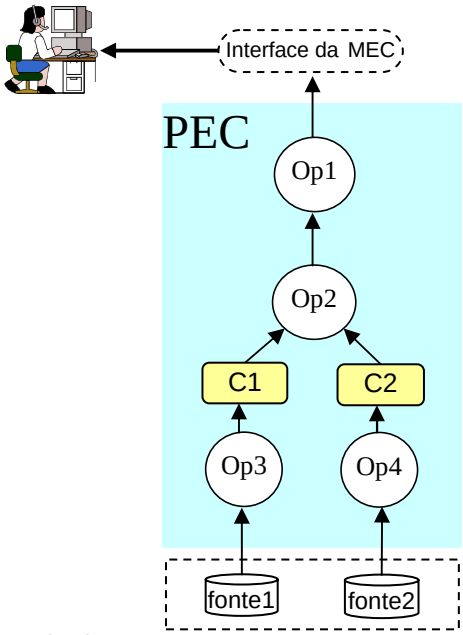

Figura 6 - Exemplo de um PEC

Os PECs podem assumir duas topologias: Linear (à esquerda ou à direita) e Ramificada (mais conhecida como *Bushy*), como ilustrado na Figura 7. Os planos lineares se diferenciam basicamente quanto ao ramo da árvore onde fluem os dados. É possível ainda existir um PEC na forma de um grafo direcionado. Neste caso, a parte comum pode ser executada separadamente e seus resultados intermediários são, em geral, materializados e lidos repetidamente.

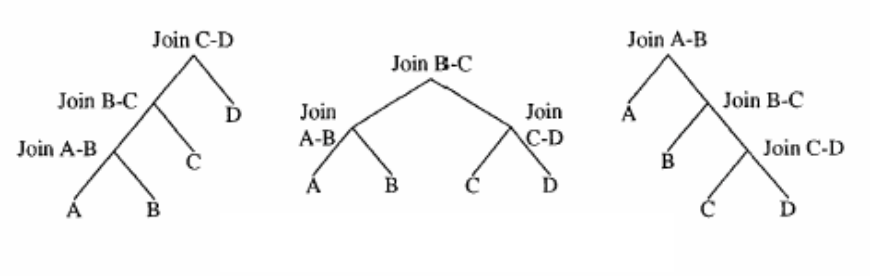

Figura 7 - Topologia de PEC: linear à esquerda, ramificada e linear à direita

A escolha da topologia a ser adotada pela MEC influencia o processo de otimização e de execução de consultas.

- Quanto ao processo de otimização, a utilização de planos lineares (i.é., planos de topologia linear), em consultas com predicados, torna-se bastante interessante na medida em que cada operação de redução (seleção ou junção) beneficia-se da operação subseqüente, reduzindo o número de dados da relação intermediária e acarretando menor tempo de execução, o que nem sempre acontece com a utilização de planos *bushy* (i.é., planos de topologia *bushy*). Além disso, o espaço de busca coberto por planos *bushy* é extremamente maior do que o produzido por planos lineares. Embora isto acarrete uma probabilidade maior de obtenção de um PEC ótimo, o tempo gasto neste processo de otimização pode torná-lo inviável, em contraste com o tempo de execução de consultas pouco complexas.
- Quanto ao processo de execução, a utilização de planos *bushy* favorece o paralelismo horizontal e a distribuição, enquanto que os planos lineares favorecem o paralelismo vertical.
  - Na estratégia de paralelismo horizontal distribuído, os PECs são expressos através de árvores *bushy* compostas de sub-consultas submetidas aos *sites* contendo os dados distribuídos. Neste caso, cada sub-consulta, a ser executada por um *site*, representa um PEC independente e paralelizável. A adoção deste paralelismo minimiza, em alguns casos, o tempo de duração da consulta (tempo de resposta), em detrimento do tempo total de execução (soma dos tempos gastos em cada operador) – que pode aumentar na medida em que um operador não se beneficia das reduções de dados de outro operador, executado em paralelo.
  - Na estratégia de paralelismo vertical, utiliza-se uma árvore linear e aloca-se cada operador em um processador diferente. Dados que atendem ao predicado de uma operação são passadas ao próximo operador que inicia seu processamento. Dessa forma, a partir do instante em que um dado passa por todas as operações, o conjunto de processadores alocados mantém-se em carga constante

processando as operações em paralelo. Há um ganho de execução na medida em que as operações, nos nós mais altos da árvore de execução, processam apenas o sub-conjunto de dados produzido pelas demais operações. Por outro lado, a dependência de execução entre os operadores pode apresentar desequilíbrio (*skew*) na distribuição de carga entre os processadores na medida em que variações, em tempo de execução (p.ex: seletividade, disponibilidade de fontes de dados, etc), impeçam a progressão uniforme dos dados pela árvore de execução.

A estratégia de execução dos operadores de um PEC é baseada num modelo de execução utilizado pela MEC que, por sua vez, está fortemente relacionado com um determinado cenário de execução de consultas.

2.3. Cenários de Execução de Consultas

Cada cenário reúne um conjunto de aplicações de consultas que são executadas segundo um certo modelo de execução. A seguir descreveremos alguns dos cenários introduzidos no Capítulo 1. Usaremos estes cenários como exemplos de extensibilidade da solução proposta neste trabalho de tese.

2.3.1. Cenário de Consultas Tradicionais

Este cenário é representado pelas aplicações sobre os SGBDs centralizados, distribuídos e paralelos, descritos anteriormente. Nestes sistemas considera-se que o otimizador da consulta produz um melhor PEC para ser executado pela MEC segundo o modelo de execução tradicional, produzindo resultados o mais cedo possível.

A seguir exemplifica-se este cenário através da execução paralela no SGBD Oracle (Mahapatra&Mishra, 2000). As consultas SQL submetidas ao sistema *Oracle* podem ser executadas em paralelo desde que haja um computador com mais de um processador (arquitetura de memória compartilhada). O servidor de paralelismo (*Oracle Paralell Server*) divide a consulta em partes que são

executadas em processos paralelos nos respectivos processadores. Por exemplo: o comando *"Select * From Produto"* numa máquina com 2 processadores, produzirá o plano de execução mostrado na Figura 8.

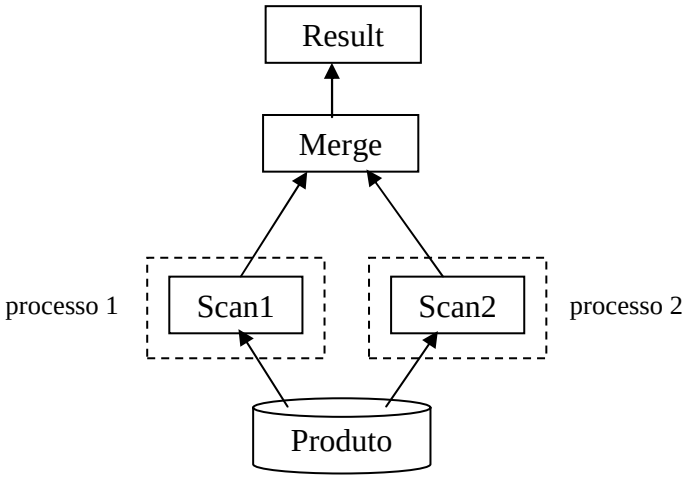

Figura 8 - PEC com paralelismo no SGBD Oracle

Neste PEC, cada operador *Scan* processa dados oriundos de diferentes partições da tabela Produto e que são combinados pela operação *Merge*, produzindo os resultados. O sistema permite que outras operações sejam paralelizadas, tais como: junção, classificação e agrupamento. O servidor permite que um banco de dados seja aberto, de maneira concorrente, para múltiplas instâncias. Cada instância porém, funciona como se fosse um servidor dedicado. Os processos 1 e 2 utilizam instâncias do servidor, uma para cada operação *Scan*. Esta arquitetura de múltiplas instâncias atende os requisitos de desempenho, escalabilidade e disponibilidade.

2.3.2. Cenário de Consultas Adaptáveis

Neste cenário, a MEC adapta a execução conforme condições em tempo de execução. As variações dinâmicas ocorrem uma vez que os operadores em um PEC:

(a) têm seu desempenho prejudicado devido ao acesso imprevisível às fontes de dados envolvidas na consulta, principalmente em se tratando

de fontes *web*, embora o registro de variações na taxa de acesso a disco possa também motivar tais flutuações, ou

(b) detectam um problema interno, por exemplo, falta de memória.

Considera-se um sistema de processamento de consultas adaptativo se (i) recebe informações do seu ambiente, (ii) usa estas informações para determinar o seu comportamento e (iii) processa de forma contínua, gerando uma interação entre o ambiente e o sistema. Desta forma, a execução adaptativa oferece uma otimização em tempo de execução (dinâmica), em contraste com o processo tradicional de otimização que é estático.

A execução adaptável sobre um conjunto de operadores de um PEC pode ser feita ao nível de tupla ou ao nível de fragmento. No primeiro caso, cada tupla é direcionada para o operador de maior desempenho, até completar todas as operações necessárias e ser enviada para a saída. No segundo caso, a execução ocorrerá de forma tradicional dentro de um fragmento do PEC até que seja detectado algum problema, e neste caso, a execução do fragmento é interrompida e o otimizador é chamado para gerar outro fragmento alternativo para ser executado. Entre as principais soluções encontradas na literatura estão: *Query Scrambling* (Amsaleg et al., 1996), *Tukwila* (Ives et al., 1999), *Eddies* (Avnur&Hellerstein, 2000) e *Hybrid* (Bouganim et al., 2001).

2.3.3. Cenário de Consultas Contínuas

A execução de consultas de forma contínua permite aos usuários obterem novos resultados de um banco de dados, sem que a mesma consulta seja submetida repetidamente. Em geral, o fluxo de controle é direcionado a dados, ou seja, com a chegada de novos dados, novos resultados são produzidos. Além disso, a duração de uma consulta pode ser muito longa. Exemplos de trabalhos relacionados são: *NiagaraCQ* (Chen et al., 2000) e *TelegraphCQ* (Chandrasekaran et al., 2003).

2.3.4. Cenário de Consultas baseadas em *Streams*

Este cenário de execução é caracterizado pelos seguintes elementos:

- Os dados da *stream* chegam *online*;
- A MEC não tem controle sobre a ordem na qual os elementos da *stream* chegam para serem processados;
- Uma *stream* pode ter tamanho quase ilimitado;
- Os dados são processados uma única vez devido às limitações de espaço na memória.

Exemplos de aplicações para este cenário incluem: aplicações financeiras, aplicações de gerenciamento do tráfego de rede; aplicações de personalização na *web* utilizando *click-streams* e aplicações que processam dados de chamadas telefônicas. Exemplos de trabalhos relacionados com este cenário são: *Continuous Eddies* (Madden et al., 2002), *TurboPath* (Josifovski et al., 2002) e o descrito em (Dobra et al., 2002).

2.3.5. Cenário de Consultas de Integração na Web

Nesta seção descreveremos o uso de uma MEC em um ambiente de integração de dados na *web*. O ambiente escolhido é o *Le Select* (LS) (LeSelect, 2003) que é um *middleware* para integração de dados que combina a arquitetura mediador-adaptador com a arquitetura de SGBDD ponto-a-ponto, como mostrado na Figura 9, cujas principais funcionalidades são listadas abaixo:

- O modelo de dados global do sistema é o modelo relacional usado para integrar fontes heterogêneas.
- Não existe um esquema global dos dados. Ao invés disto, cada servidor LS é um responsável pela publicação e acesso às fontes de dados (locais e remotas),
- As fontes de dados são identificadas por uma URL (por exemplo: *leselect://rodin.inria.fr/wrapper/Table*)

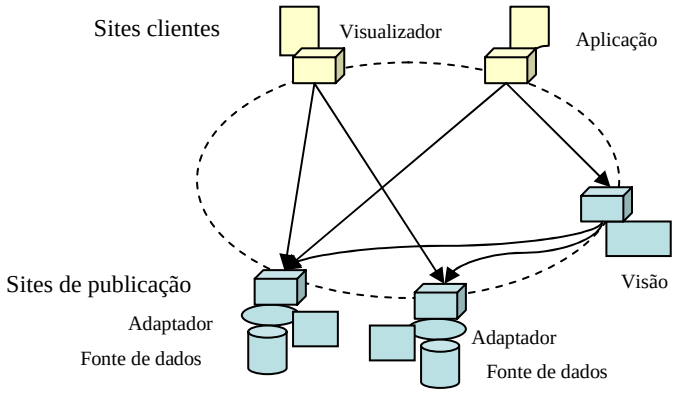

Figura 9 - Arquitetura do sistema *Le Select*

As aplicações têm acesso ao sistema através de consultas SQL. Dependendo da consulta, um servidor poderá executá-la localmente ou remotamente com a colaboração de outros servidores, ou seja, um servidor recebe pedidos de consultas, a partir de aplicações, ou sub-planos de consultas, a partir de outros servidores. Para permitir isso, cada servidor contém uma MEC completa (Figura 10) capaz de executar PECs contendo operadores algébricos e de controle (para acesso e transferência de dados), ambos implementando a interface iterador. Além disso, cada adaptador deve publicar seus metadados. O publicador deverá implementar o método *executeQuery* de cada adaptador, bem como o método *next* para acesso aos seus resultados. O acesso a adaptadores (locais e remotos) é feito através de *proxies* (Gamma et al., 1995) e o acesso a metadados é feito a partir dos adaptadores. Um dos metadados exportado por um adaptador é a descrição das capacidades de consulta da sua fonte de dados, tais como: filtros sobre atributos das relações; junção e união de tabelas; expressões matemáticas em projeções.

O otimizador produz um PEC distribuído formado pela combinação de sub-planos de topologia *bushy*, enviados aos adaptadores, com um plano linear paralelo em *pipeline*, que integra os resultados intermediários produzidos pelos sub-planos. A comunicação entre sub-planos enviados a sites distribuídos é feita por uma versão do operador de controle *Exchange*. O PEC distribuído é particionado em diversos sub-planos. Cada sub-plano é enviado ao servidor publicador dos objetos nele referenciados. A execução global segue o modelo de execução iterador realizando paralelismo *pipeline* entre os sites participantes.

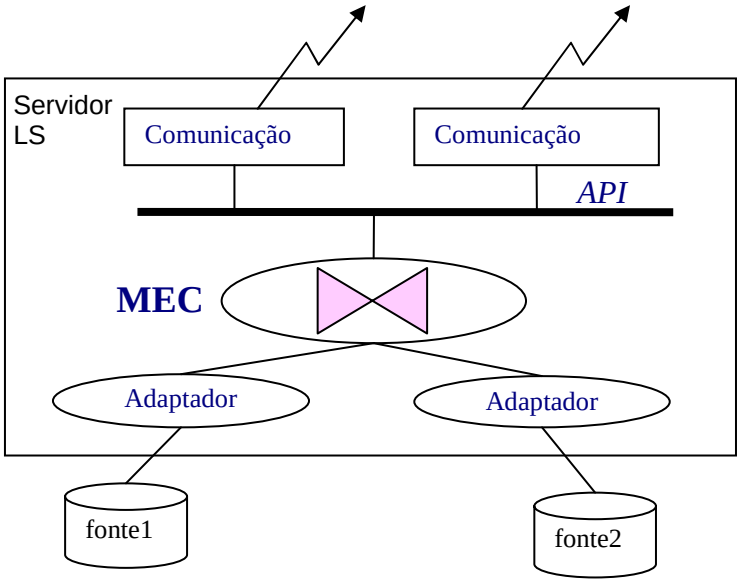

Figura 10 - O servidor do *Le Select*

O LS implementa o operador *BindJoin* (Florescu et al., 1999) que modela uma variação da operação de junção onde o operando à esquerda é uma expressão relacional e à direita, uma relação com restrição de acesso. Este operador não apresenta as propriedades de simetria e associatividade, encontradas na operação de junção. A execução do *BindJoin* processa-se segundo o modelo tupla-a-tupla, onde as tuplas da relação à esquerda (*outer*) são lidas e os valores correspondentes aos atributos obrigatórios da relação à direita (*inner*) são a ela enviados, um a um, juntamente com um identificador de tupla. A relação *inner* é acessada com o valor obrigatório, e as tuplas associadas, caso existam, são retornadas ao operador. Para cada uma das tuplas retornadas, a tupla correspondente ao valor avaliado compõe uma tupla resposta da junção.

2.4. Síntese do Capítulo

Neste capítulo, foi apresentada uma visão geral dos conceitos relacionados ao processamento de consultas em SGBDs distribuídos e paralelos, SGBDs XML e Sistemas de Integração de Dados na *web*, focando principalmente em questões de execução de consultas. Em particular, mostramos os principais conceitos relacionados à Máquinas de Execução de Consultas (MEC) bem como um exemplo de MEC num ambiente de integração de dados na *web*. Descrevemos

alguns Cenários de Execução de Consultas relevantes para esta tese. Observando-se as MECs existentes nestes cenários de execução, verifica-se que as mesmas foram projetadas para modelo de execução e de dados específicos, ou seja, cada uma suporta apenas um modelo de execução que, em geral, fica implícito e codificado na sua implementação. O estudo destes cenários de execução serve como base para a solução proposta nesta tese mostrada no próximo capítulo.

Neste ponto, é importante listar os principais conceitos que utilizaremos neste trabalho:

- execução de PECs compostos de operações de seleção, projeção e junção;
- execução de consultas apenas de leitura não tratando, portanto, de aspectos relacionados à atualização e gerência de transação;
- execução de consultas em ambiente distribuído e paralelo, com ênfase no suporte a consultas de integração.
- execução de consultas nos modelos de dados relacional e XML.

Adicionalmente, como o principal objetivo desta tese é construir uma MEC extensível, não discutiremos técnicas de otimização de consultas.