Fundação Edson Queiroz
Universidade de Fortaleza - UNIFOR
Centro de Ciências Tecnológicas – CCT
Mestrado em Informática Aplicada - MIA

UMA ESTRATÉGIA PARA OTIMIZAÇÃO DO PROCESSAMENTO DE CONSULTAS NA ARQUITETURA AMDB USANDO REGRAS (EVENTO-CONDIÇÃO-AÇÃO)

Ernani Andrade Leite

Dezembro – 2004

Ernani Andrade Leite

UMA ESTRATÉGIA PARA OTIMIZAÇÃO DO PROCESSAMENTO DE CONSULTAS NA ARQUITETURA AMDB USANDO REGRAS (EVENTO-CONDIÇÃO-AÇÃO)

Dissertação apresentada ao Curso de Mestrado em Informática Aplicada – MIA da Universidade de Fortaleza sob a orientação do Prof. Dr. Ing. Angelo Roncalli Alencar Brayner.

Fortaleza – 2004

BANCA EXAMINADORA

_____
Prof. Ângelo Roncalli Alencar Brayner, Dr.-Ing.

_____
Profª Maria Lígia Barbosa Perkusich, Drª.Sc.

_____
Prof. Pedro Porfírio Muniz Farias, Dr. Sc.

Aprovada em _____/_____/_____

“Não é o desafio com que nos deparamos que determina quem somos e o que estamos nos tornando, mas a maneira como respondemos ao desafio. Somos combatentes, idealistas, mas plenamente conscientes, Porque o ter consciência não nos obriga a ter teoria sobre as coisas: só nos obriga a sermos conscientes. Problemas para vencer, liberdade para provar. E, enquanto acreditamos no nosso sonho, nada é por acaso”.

HENFIL

AGRADECIMENTOS

Em primeiro lugar agradeço a DEUS pelo dom da sabedoria e saúde que me concedeu para participação no curso de pós graduação e conclusão deste trabalho;

A minha esposa, Ana Lúcia, e meus filhos: Ernani Filho, Arthur e Cecília, pelos constantes momentos de ausência que os fiz passar e os momentos de lazer que nos foram suprimidos;

Aos meus pais, Celeste e Deusdedith, pelo incentivo e carinho com que sempre me encorajam durante minhas difíceis caminhadas;

Aos meus irmãos e irmãs, pelo incentivo e apoio a obtenção deste título para minha formação profissional, em especial a “Tia Edna”, pela paciência e serenidade demonstrada no convívio diário.

Ao Chanceler Emerson Azevêdo por me proporcionar a oportunidade de cursar o mestrado.

Ao Prof. Angelo Brayner, meu orientador e amigo, que teve paciência e capacidade de me conduzir nesse processo de investigação científica. Sua dedicação e apoio foram decisivos.

A todas as pessoas, amigas e amigos, em especial aos Profs. José Aguiar e Edmar Candeia, que souberam com humildade, colaborar para a conclusão deste trabalho.

Aos Professores e Mestres pela participação na banca examinadora.

RESUMO

O rápido desenvolvimento da tecnologia de comunicações móveis está expandindo o acesso e a demanda por informação, independente da localização do usuário ou da informação. Como essa informação deve estar armazenada em diferentes lugares e/ou em bases de dados heterogêneas, o seu gerenciamento é um grande desafio. O acesso à informação distribuída implica no compartilhamento de bases de dados independente de onde elas estejam armazenadas e deve ser feita garantindo a integridade dos dados, como também a velocidade na resposta da consulta. A tecnologia de comunicação móvel tem permitido associar a mobilidade à base de dados, e isso aumenta o desafio no trabalho com informação distribuída. Para permitir o uso de base de dados móveis muitos problemas de hardware e software devem ser resolvidos. Alguns desses problemas são similares aos encontrados em bases de dados distribuídas. Apesar das diferenças, pois cada um tem suas próprias características, essas tecnologias enfrentam dificuldades em comum, dentre elas tem-se: a replicação de dados (fragmentos de dados são armazenados em mais de um lugar) o que implica em mecanismos de junção desses dados, os modelos de transações e processamentos de consulta, entre outros. Pesquisas têm sido realizadas para criar modelos e arquiteturas para solucionar os problemas com essas tecnologias. Um exemplo é a arquitetura AMDB (Arquitetura de Banco de Dados Móveis), que proporciona uma iteração entre membros de uma coleção de bancos de dados autônomos e moveis denominada CBDM (Comunidade de Bancos de Dados Móveis). Nessa comunidade bancos de dados móveis podem entrar ou sair a qualquer tempo, o que implica na necessidade de regras para os processos de conexão e desconexão, mesmo que essa última seja involuntária. Neste contexto, cada participante da CBDM pode acessar um outro banco de dados participante através da infra-estrutura de comunicação. Este trabalho apresenta um estudo sobre os operadores de junção adaptativos para uso na arquitetura AMDB, além de sugerir ações a eventos ocorridos durante o processamento de consulta, como a substituição de um operador, quando este apresentar ineficiência no processamento. Os critérios para disparar ações decorrem da restrição de memória e demora na entrega de tuplas, que impactam ou inviabilizam o processamento de consulta na comunidade, caso não recebam tratamento adequado. Além disso, foram sugeridas ações de controle na arquitetura AMDB durante eventuais desconexões dos membros da comunidade, quer sejam voluntárias ou involuntárias.

ABSTRACT

The rapid development of the mobile communications technology it is expanding the access and the demand for information, independent where the user or information is located. Since this information may be stored in different places and/or heterogeneous databases, its management it is a great challenge. The access of distributed information implicates in the sharing of the databases instead of where they are stored and it must be done guaranteeing the integrity of the data involved, as well as the speed in the answer the these consultations. The mobile technology has permitted to associate mobility to the databases, and it increases the challenge in the work with distributed information. To permit the use of the technology of mobile databases many hardware and software problems have to be solved. Some of those problems are similar to those in distributed databases. Although they are different technologies, because each one has specific characteristics, they have difficulties in common, we mentioned: the data replication (fragments of data stored in more than a place) what implicates in the need of operators for junction of those data, the models of transactions and processing of consultations, among others. Research has been done to create models and architectures to mobile databases and to circumvent the problems with this technology. An example is AMDB (Mobile Accessing Databases) architecture, it provides an iteration among members of a collection of movable and autonomous databases, named Mobile Database Community (MDBC). Mobile databases can enter or to leave of a community any time, and it that must happen in agreement with some rule that treats connections and disconnections, same being those last ones involuntary. In such context, each participant member of CBDM, it can access the other databases through a communication infrastructure without thread. This work presents a study on the adaptative junction operators for use in the architecture AMDB. Action actions to events happened during the consultation processing are presented, as the substitution of an operator, when this to present inefficiency in the processing. The criteria to discharge actions elapse of the restriction of memory and it is long in the t-uplas delivery, that impact or they make unfeasible the community's processing in case they don't receive appropriate treatment. Besides, it is suggested action of control in the architecture AMDB during eventual disconnections, voluntary or involuntary, of the community's members.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

CAPÍTULO 1
INTRODUÇÃO

1.1 Motivação

A velocidade com que a tecnologia da comunicação móvel vem se expandindo traz a demanda de acesso a informações, independente da localidade do usuário ou da informação requerida. Um grande desafio é o gerenciamento dessas informações, visando garantir a integridade dos dados envolvidos nos processamentos, bem como a rapidez na resposta a estas consultas .

Com o desenvolvimento de sistemas de comunicação sem fio e de computadores portáteis, a computação móvel tornou-se realidade no ambiente computacional moderno. A integração destas duas tecnologias possibilita que computadores móveis (*laptops*, *notebooks*, *palms*, etc...) conectem-se, como componentes, a um ambiente distribuído e disponibilizem seus recursos computacionais, mesmo que mudem constantemente sua localização, ou seja, apesar de não apresentarem localização fixa em uma rede. Atualmente, o compartilhamento de informações entre múltiplas fontes de dados heterogêneas, autônomas, distribuídas e móveis tem emergido como um requerimento estratégico que deve ser suportado pela tecnologia de bancos de dados. Usuários que carregam equipamentos portáteis estão habilitados para acessar serviços de bancos de dados independentemente da sua localização física ou padrão de movimentação. De acordo com este novo cenário, comunidades de bancos de dados móveis podem ser formadas. Definimos uma Comunidade de Bancos de Dados Móveis (CBDM) como uma coleção dinâmica de banco de dados móveis, distribuídos e autônomos, na qual cada usuário do banco de dados pode acessar

cada um dos outros bancos de dados através de uma infra-estrutura de comunicação sem fio.[1]

Os ambientes de computação móvel utilizam uma combinação de *hosts* fixos e móveis, onde os *hosts* fixos trocam mensagens através de uma infra-estrutura padrão Internet. *Hosts* fixos atuam como estações de base e controlam o tráfego de mensagens entre os *hosts* móveis. Entretanto, uma rede móvel pode funcionar independente de um mecanismo de comunicação física. A estas estruturas dinâmicas denominamos "*mobile ad hoc network*" [2].

Para que usuários de uma CBDM possam acessar base de dados compartilhadas, muitos problemas de hardware e software têm de ser resolvidos, possibilitando o uso efetivo da tecnologia de bancos de dados móveis. Alguns desses problemas já são conhecidos do uso de bancos de dados distribuídos. Embora sejam tecnologias distintas, pois cada uma possui suas especificações, entre as dificuldades em comum da implantação de bancos de dados distribuídos e móveis, citamos: cópias idênticas (fragmentos de dados armazenados em mais de uma localidade), os modelos de transações e processamento de consultas, a modelagem da base de dados e a linguagem de consulta a essas bases devem ter diferenças das comuns, bem como as falhas e recuperação[3].

Quanto aos problemas do processamento de consulta, destacam-se três técnicas tradicionais que têm demonstrado ineficiência, numa CBDM: *ausências de estatísticas* – devido a variação do número de bancos de dados móveis autônomos, participantes da comunidade; *tempo de resposta de uma consulta* – independente da escolha do melhor plano de consulta, problemas nas desconexões devido ao ambiente móvel em si, podem tornar o plano ineficiente e; *operadores com mais de uma fase que não apresentam suporte a técnica de pipelining* (operadores de consulta que suportam a técnica de *pipelining* geram

---

[1] MENDONÇA, N., et al. **Sharing Móbile in Dynamically Configurable Environments.** [S.n.t.].

[2] MACKER, J. P., M. S. Corson. **Mobile Ad Hoc Networks and the IETF.** Internet Engineering Task Force, MANET Working Group. Available online at http://www.ietf.org/html.charter/manet-charter.html.

[3] SILBERCHATZ, A.; et al. **Sistema de Banco de Dados.** [S.n.] Makron Books, 1999.

tuplas tão logo tuplas de entrada do operador são recebidas) – por exemplo, o operador *has join* que possui as fases de construção e comparação e não usa *pipelining*; isto é, a segunda fase só inicia-se quando a primeira fase é totalmente concluída. Logo, a fase de comparação só pode ser executada após a fase da construção ser totalmente concluída, o que pode afetar seu desempenho no processamento, caso ocorra atraso na entrega dos dados na primeira fase.

Apesar das dificuldades encontradas, o desenvolvimento dos sistemas de comunicação sem fio vem conseguindo vencer parte desses obstáculos. Códigos e agentes móveis que migram entre espaços lógicos dos *hosts* da rede são explorados como um novo paradigma de desenvolvimento para aplicações distribuídas. Referenciamos novos modelos, arquiteturas e tecnologia para computação móvel.

O presente trabalho, enfoca o problema do processamento eficiente de consulta numa CBDM. A arquitetura AMDB[4] (Acesso a Banco de Dados Móveis) [30] será utilizada como ambiente computacional para validar a abordagem proposta, visto que a arquitetura AMDB[5] apresenta processamento de consulta que necessita de mais estratégias (regras) de otimização.

1.2 Objetivos

Quando tratamos de consultas em sistemas de banco de dados móveis estas diferem das consultas em sistemas fixos, pois nos sistemas móveis a mobilidade adiciona poder ao usuário móvel, ao mesmo tempo aumenta a complexidade de processamento das consultas; pois, as redes de comunicação necessitam ser ubíquas, garantindo a conectividade independente da localização do usuário. Acrescente-se o fato de uma configuração dinâmica da rede, pois a participação de usuários é ocasional e a migração desses usuários pode ocorrer

[4] MORAES FILHO, José Aguiar. **AMDB – An approach for Sharing Mobile Databases. Master Dissertation.** Universidade de Fortaleza (UNIFOR), 2003.

[5] Id.Ibid.,2003.

com freqüência; isto é, a rede deve ser dinamicamente configurável. Redes com estas características são denominadas redes *ad hoc*[6]. Fica evidente que os mecanismos aplicados a ambientes de banco de dados tradicionais não podem suprir as necessidades dessa nova tecnologia. Deve-se considerar que o processamento de consultas no modelo de rede da computação móvel e a autonomia dos equipamentos portáteis, assim como o controle de transações e recuperação devem estar preparados para as desconexões dos clientes móveis, e reconexões quando tratamos de equipamentos móveis. A arquitetura proposta em[7] representa tais características.

Diante da motivação exposta este trabalho têm como objetivos:

i. Estabelecer regras (evento-condição-ação), visando otimizar o processamento de consulta na arquitetura AMDB, ativadas a partir da identificação de eventos que diminuam a eficácia do tempo de resposta as consultas;

ii. Identificar entre os operadores adaptativos Ajax, Mobijoin e Xjoin; quais ações devem ser ativadas durante o processamento de consulta na arquitetura AMDB visando melhorar o desempenho durante o processamento;

iii. Estabelecer ações durante as conexões ou desconexões (voluntárias ou involuntárias) na arquitetura AMDB, garantindo um melhor desempenho da CBDM, durante eventuais desconexões dos seus participantes;

1.3 Estrutura da Dissertação

Este trabalho está organizado da seguinte forma: além desta introdução, contém mais seis capítulos, descritos, resumidamente, a seguir:

---

[6] MACKER, J. P., M. S. op. cit. [S.d.].
[7] MORAES FILHO, José Aguiar. op. Cit. (UNIFOR), 2003.

O segundo capítulo, apresenta os principais conceitos que norteiam a tecnologia de Banco de Dados, buscando fornecer noções fundamentais para a compreensão dos modelos clássicos existentes.

No terceiro capítulo, são apresentados conceitos sobre computação móvel, meios de comunicação, bem como a idéia de mobilidade lógica e física, com o objetivo de contextualizar o funcionamento da computação móvel, servindo como referencial para o capítulo seguinte.

No quarto capítulo, apresentam-se os principais estudos da literatura sobre ambientes de computação móvel.

No quinto capítulo, apresenta-se e detalha-se a arquitetura AMDB proposta em[8], visando dar o suporte para a proposta de solução do problema caracterizado na introdução.

No último capítulo, apresenta-se uma proposta para otimizar o processamento de consulta da arquitetura AMDB, que é o objetivo deste trabalho.

Finalmente, conclui-se o trabalho, evidenciando-se suas contribuições e futuras atividades de pesquisa que poderão ser desenvolvidas.

[8] Id. Ibid., 2003.

CAPÍTULO 2

BANCOS DE DADOS: CONCEITOS E ARQUITETURAS

2.1 Introdução

Este capítulo apresenta uma breve introdução sobre SGBD - Sistemas Gerenciadores de Banco de Dados.

A seção 2.2 descreve os SGBD's e seus principais componentes: termos básicos como Sistemas Gerenciadores de Banco de Dados, Processador de Consulta e Sistema de Armazenamento, entre outros. A seção 2.3 apresenta as principais arquiteturas de Banco de Dados encontrados na literatura.

2.2 Modelo Clássico de SGBD

Os sistemas de banco de dados estão disponíveis em máquinas que abrangem desde pequenos micros até computadores de grande porte. A Figura 1 mostra uma arquitetura de um Sistema de Banco de Dados.

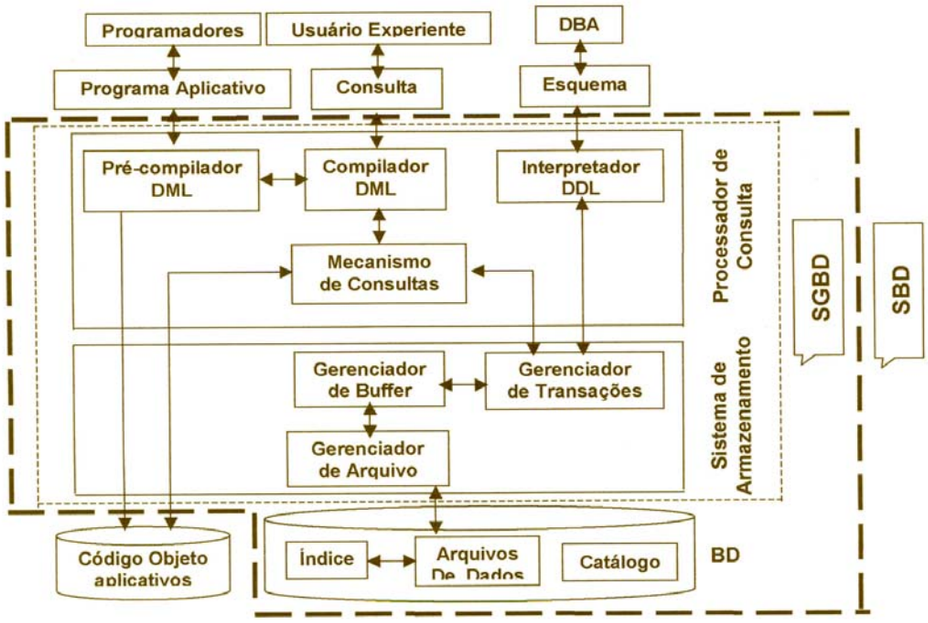


FIGURA 1 - ARQUITETURA DE UM SISTEMA DE BANCO DE DADOS

Um SGBD é um software que manipula todos os acessos ao banco de dados e possui vários componentes e funções de controle e proteção dos dados. Tais funções visam proteger o banco de dados de possíveis danos voluntários ou involuntários. O SGBD é constituído por um conjunto de dados associados a um conjunto de programas que possibilitam o armazenamento, recuperação e a manipulação de grandes volumes de informações de forma eficiente, garantindo a segurança e a integridade desses dados[8].

A Figura 1 apresenta o SGBD em duas partes: Processador de Consultas e Sistema de Armazenamento. A seguir descrevemos os principais componentes:

2.2.1 Processador de Consultas

O Processador de consultas, conforme verificado na Figura 1, possui quatro componentes: *Compilador DML* – analisa sintática e semanticamente comandos expressos em uma linguagem de consulta, além de traduzi-los para

[8] DATE, C. J. **Introdução a Sistemas de Banco de Dados.** [S.n.], Campus, 1998.

uma forma de representação interna de consulta; *Pré-Compilador DML* – inseridos em programas de aplicação, traduz comandos *DML* em chamadas a procedimentos na linguagem hospedeira; *Interpretador DDL* – interpreta os comandos *DDL* e armazena-os no catálogo e; *Mecanismo de Consultas* – Executam instruções de baixo nível geradas pelo compilador *DML*, visando otimizar e gerar os planos de execução de consultas.

2.2.2 Sistema de Armazenamento

O Sistema de Armazenamento possui três componentes: *Gerenciador de Transações* - tem como função controlar execução concorrente entre as transações bem como a recuperação do BD no caso de eventuais falhas; *Gerenciador de Buffer* – responsável pela recuperação (ou intermediação) de dados do disco e carrega-los na Memória Principal em forma de páginas e; *Gerenciador de Arquivos* – responsável pelo armazenamento físico dos dados no disco , ou seja, gerencia a alocação de espaço em disco e as estruturas dos dados usados para representar tais dados.

Detalharemos os processadores de consultas, visto que é o foco de nossa proposta para a arquitetura AMDB. A Figura 2 mostra as fases do Processamento de Consulta tradicional. Definimos processamentos de consultas como as atividades envolvidas em extrair dados de um Banco de Dados[9]. Tais atividades incluem: a tradução de consultas expressas em linguagens de alto nível do Banco de Dados em expressões que podem ser implementadas no nível físico do sistema de arquivos, otimizações, traduções e na avaliação das consultas. Para ilustrar, dividimos em duas fases o processamento em uma consulta expressa em uma linguagem de consulta de alto nível, como a SQL :

---

[9] SILBERCHATZ, A.; et al. **op. cit.**, 1999.

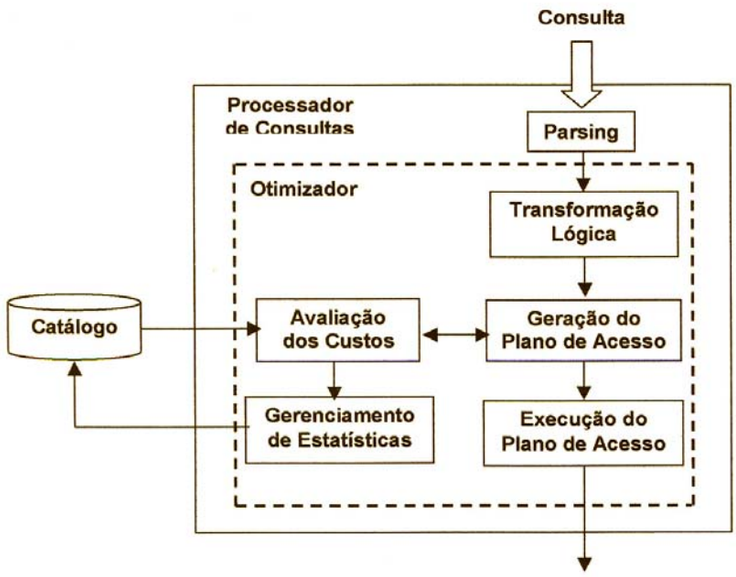


FIGURA 2 – FASES DO PROCESSAMENTO DE CONSULTA

Fase (I): análise sintática e semântica

Na análise sintática e semântica, a linguagem de consulta primeiramente passa a ser examinada, onde são identificados os símbolos da linguagem, tais como: palavras-chave da SQL, nomes de atributos e nomes de relações (no texto da consulta); seguido por uma analise (parser), onde são verificados a sintaxe da consulta para determinar se ela está formulada de acordo com as regras de sintaxe (regras de gramática) da linguagem de consulta. A consulta deve ainda ser validada, isto é, verificar se todos os nomes de atributos e relações são válidos e semanticamente significativos no esquema do banco de dados específico que está sendo consultado.

Fase (II): Otimização

Uma consulta geralmente tem muitas estratégias de execução, e o processo utilizado para escolher a estratégia mais adequada para processar uma consulta é denominado de otimização de consulta. Cabe ao otimizador a tarefa

de produzir um plano de execução. Esta fase é subdividida em: transformação lógica, avaliação dos custos, e geração do Plano de Acesso.

- **Transformação Lógica**

A transformação é responsável por três ações: Padronização – forma padronizada visando otimização; Simplificação – eliminação de redundância; Melhoria – uso de expressões mais eficientes para melhor desempenho na execução.

- **Avaliação dos Custos**

O custo de avaliação de uma consulta pode ser medido por meio de vários recursos diferentes, tais como: acesso a discos, tempo de CPU para executar uma consulta, custo da comunicação, entre outros, e possui como objetivo minimizar o tempo de resposta de uma consulta bem como a utilização de recursos para uma determinada saída. Os principais itens para otimizar uma consulta, são: Custo de comunicação – transmissão de dados; Custo de acesso à memória secundária – carregamento de páginas na memória principal; Custo de armazenamento – ocupação de *buffers*; Custo de processamento – utilização de CPU;

- **Geração do Plano de Acesso**

O plano de acesso descreve a seqüência de execução das operações de uma consulta, a partir das tabelas existentes até seu resultado final. Suas principais ações: 1) gerar possíveis planos de acessos para a execução da consulta; 2) estender os planos de acesso com detalhes físicos dos dados, tais como existência de índices e 3) escolher o plano mais barato com base no modelo de custo e custos de processamento.

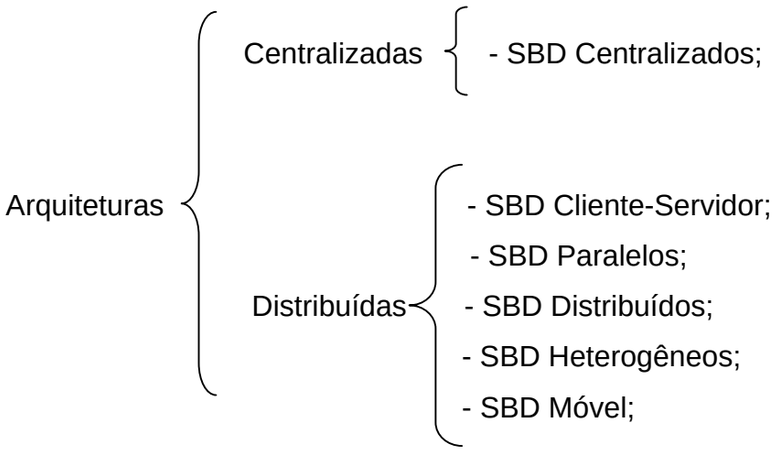
Ressalte-se que, em alguns casos, o plano de execução escolhido não é a estratégia ótima (melhor), e sim uma estratégia razoavelmente eficiente para executar a consulta. Encontrar a estratégia ótima geralmente consome muito

tempo, exceto para consultas mais simples, e pode requerer informações sobre a maneira como os arquivos são implementados e até mesmo o conteúdo dos arquivos.

Se um SGBD fornece somente uma linguagem de navegação, existem poucas oportunidades para a otimização extensiva de consulta por parte do SGBD. É possível também o programador escolher a estratégia “ótima” de execução, quando estiver trabalhando com linguagens de banco de dados de baixo nível.

2.3 Arquiteturas de Sistemas de Banco de Dados

A arquitetura de um Sistema de Banco de Dados – SDB, está diretamente relacionada ao modelo computacional sobre o qual o Sistema de Banco de Dados é executado. Pode-se então, classificar os sistemas de BDS como descrito a seguir:



2.3.1 Centralizada

Nesta arquitetura, os SBD são centralizados, e seus componentes residem no mesmo *host*.

2.3.2 Distribuída

Neste tipo de arquitetura, pode haver uma distribuição de componentes do SBD, de controle em execução de tarefas ou de ambas. Portanto, dependendo dos critérios de distribuição, podemos ter arquiteturas distintas de SBDD, a saber: Sistemas de Banco de Dados Cliente-Servidor – é o sistema que tem como característica a distribuição de funções do SGBD entre Clientes e Servidor; Sistemas de Banco de Dados Paralelos – é o sistema que tem como característica a distribuição de funções do SGBD entre diversos sistemas computadorizados; Sistemas de Banco de Dados Distribuídos – é o sistema que tem como característica a distribuição de dados através de diversos BDSs homogêneos; Sistemas de Banco de Dados Heterogêneos – é o sistema que tem como característica a distribuição de dados através de BDSs heterogêneos e autônomos; Sistema de Banco de Dados Móvel – é o sistema que tem como característica a distribuição de funções e dados do SGBD entre clientes e servidor em ambientes de computação móvel.

2.4 Resumo

Neste capítulo, apresentamos uma breve explanação sobre as arquiteturas de Bancos de Dados e descrevemos as etapas do processamento de consulta para um ambiente estático. No próximo capítulo descreveremos o ambiente de computação móvel, bem como os problemas da mobilidade.

CAPÍTULO 3

COMPUTAÇÃO MÓVEL

3.1 Introdução

Os avanços tecnológicos, especificamente no que diz respeito às redes sem fio, e a popularidade dos dispositivos móveis portáteis deram origem a um novo paradigma de computação chamado de computação móvel, onde as máquinas não possuem mais localização fixa. Esse novo paradigma de computação possibilita às pessoas acessarem bancos de dados remotos a partir de qualquer lugar, em qualquer hora e enquanto as mesmas estão em movimento. Assim, os usuários móveis podem locomover-se junto com seus *hosts* através de posições físicas ou de regiões geográficas diferentes e, ainda permanecer conectado à rede através de ligações sem fio.

Neste capítulo, a seção 3.2 apresenta a mobilidade e os ambientes móveis de suporte. Na seção 3.3 descreve-se um breve resumo sobre redes sem fio. No seção 3.4 caracterizam-se os tipos de mobilidade, considerados neste trabalho.

3.2 Modelo de Computação Móvel

Apresentado na Figura 3 um modelo abstrato para um ambiente de computação móvel pode ser definido como será descrito a seguir. Os computadores móveis, denominados genericamente de unidades móveis – MU, são agrupados em componentes chamados células. Cada célula representa uma área geográfica coberta por uma infra-estrutura de comunicação sem fio. Tais células podem representar uma rede de área local sem fio (WLAN), uma rede *ad*

hoc, uma área geográfica coberta por uma rede de telefonia celular (chamada também de célula), ou uma combinação de tais tecnologias de comunicação (por exemplo, uma rede *ad hoc* dentro de uma célula de uma rede de telefonia celular).

As estações de suporte à mobilidade (chamadas estações base) são os componentes de uma plataforma de computação móvel, que têm uma interface sem fio permitindo uma comunicação entre as unidades móveis situadas em células diferentes. Assim, cada célula deve ser associada a uma estação de suporte móvel. A comunicação entre as duas estações base é realizada através de uma rede fixa. De fato, as estações móveis de suporte representam hospedeiros (*hosts*) fixos interconectados através de uma rede de alta velocidade ligada por cabos. Do ponto de vista da tecnologia de banco de dados, podem existir duas classes de sistemas de banco de dados em um ambiente de computação móvel:

(I) Uma classe consistindo de sistemas de banco de dados que são hospedados e *hosts* fixos. Nesta classe, os sistemas de banco de dados não podem mudar sua posição física através do tempo. Em outras as palavras, seu endereço de rede são conhecidos anteriormente e não mudam; e

(ii) Uma classe de sistemas de banco de dados formada por computadores móveis (ou em *hosts* móveis). Nesta classe, a posição física de sistemas banco de dados pode variar através do tempo. Para ilustrar, considere que uma consulta seja submetida a um banco de dados móvel na posição A. Durante a execução da consulta, o *host* abrangendo o sistema de banco de dados move-se para a posição B e recebe uma outra consulta. Após isso, o *host* move-se para a posição C e fornece os resultados para a primeira consulta, que são enviados à posição A. Os bancos de dados, nesta classe, são denominados de bancos de dados móveis.

Vale a pena observar que, quando uma célula em um ambiente de computação móvel representa uma WLAN ou uma rede *ad hoc*, os computadores móveis dentro da célula podem se comunicar entre si diretamente. Por outro lado,

quando uma célula representa uma área coberta por uma rede de telefonia celular, as unidades móveis necessitam usar a estação de base da célula para se comunicar.

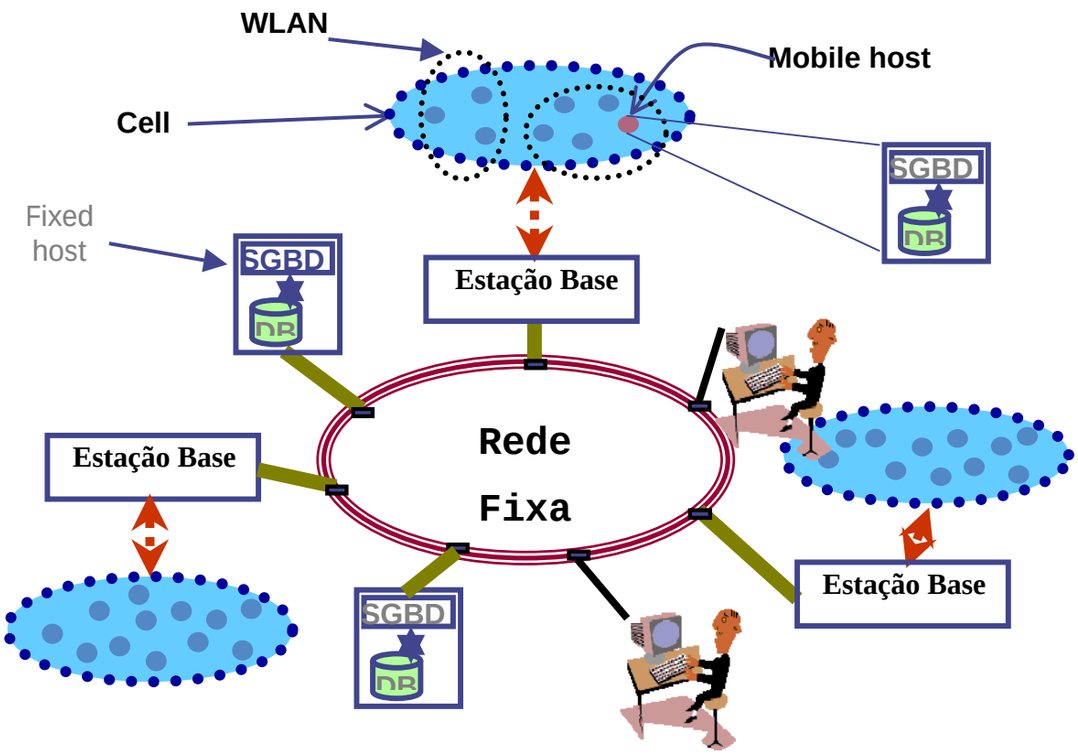


FIGURA – 3 – MODELO DE COMPUTAÇÃO MÓVEL

Um computador móvel pode usar diferentes tecnologias de comunicação. O *handoff* vertical significa a migração de um computador móvel de uma determinada tecnologia de comunicação para outra. Por exemplo, um computador móvel pode usar uma *WLAN* para comunicar-se com um outro computador móvel dentro da mesma célula, e ele pode também usar uma rede de telefonia celular para comunicar-se com os computadores móveis situados em uma outra célula.

3.3 Redes sem Fio

Recentemente, diversos pesquisadores na área da rede de comunicação voltaram seu foco para um tipo de rede de comunicação que não usa ligações através de fios. Tal rede de comunicação emprega o rádio, ondas infravermelhas ou de satélite para conduzir sinais de dados de computadores. É chamada rede sem fio ou infra-estrutura de comunicação sem fio.

As redes sem fio geralmente possuem uma topologia estática no sentido de que o número e o tipo de seus componentes devam ser conhecidos anteriormente. Nos últimos anos, o IETF (Força Tarefa de Engenharia de Internet) está desenvolvendo, através de seu grupo de trabalho chamado de MANET (Redes Móveis *Ad hoc*), um tipo de rede sem fio denominada de rede *ad hoc*. Uma rede *ad hoc* é uma infra-estrutura de comunicação sem fio que é formada dinamicamente configurável. Uma outra característica importante de uma rede *ad hoc* é que ela possui uma topologia dinâmica, que muda rapidamente em relação ao movimento de unidades móveis e de conexões transientes. Qualquer componente de uma rede *ad hoc* pode se juntar ou abandonar a rede a qualquer hora.

As infra-estruturas de comunicação sem fio, e, consequentemente, as redes *ad hoc* sofrem as limitações a seguir. As redes sem fio têm uma capacidade baixa de transmissão (*bandwidth* baixa) em comparação às redes fixas. Além disso, as redes sem fio possuem uma taxa de erro elevada de transmissão. Estas características limitam o número de aplicações baseadas na transmissão sem fio e oferecem um desafio aos pesquisadores que fazem possível o desenvolvimento de aplicações gerais baseadas nesta infra-estrutura da rede.

3.4 Mobilidade Física e Lógica

Uma comunicação sem fio permite a mobilidade física entre seus componentes (unidades móveis). Hoje em dia, um exemplo da mobilidade física é o sistema de telefonia celular. Os dispositivos do telefone celular podem mover-se

dentro de uma área coberta de comunicação sem fio sem rompimentos de comunicação. Além disso, enquanto se movem dentro da área, podem se conectar um ao outro.

O conceito lógico da mobilidade veio da pesquisa da tecnologia de software sobre o código de migração e evoluiu do conceito do agente móvel de software (ou agente móvel). Os agentes móveis são partes de código que têm a capacidade de migrar de uma determinada unidade (dispositivo ou máquina) para outra a fim executar uma tarefa em nome do usuário da unidade. As plataformas dos agentes móveis, chamadas de sistema de agente móvel , foram construídas em academias tais como AGLETS[10] e Lime[11]. É importante notar que a mobilidade lógica não está restrita a uma infra-estrutura de comunicação subjacente. De fato, a mobilidade lógica pode estar presente em uma rede com fios.

De uma perspectiva da tecnologia de banco de dados, a mobilidade pode ser categorizada em dois tipos diferentes:

(I) Mobilidade Física: este tipo de mobilidade trata com a conexão entre a mobilidade espacial de clientes e de usuários de banco de dados através das diferentes regiões do espaço;

(ii) Mobilidade Lógica: este tipo de mobilidade está relacionada à migração de código entre diversos clientes móveis e usuários de banco de dados. A fim de fornecer a mobilidade lógica, os usuários dos computadores móveis devem poder gerar códigos de acesso a banco de dados (comandos SQL, procedimentos armazenados ou métodos), que poderiam migrar de forma autônoma para diversos usuários de bancos de dados. Quando tal código chega a

---

[10] ARIDOR, Y. LANGE, Danny B. **Agent Design Patterns: Elements of Agent Application Design.** Proceedings of the second International Conference on Autonomous Agents. May, 10/13. Minneapolis, Minnesota, United States, 1998, p.108-115.

[11] MURPHY, A. L., et al. **A Middleware for Physical and Logical Mobility.** Proceedings of the 21st International Conference on Distributed Computing Systems. April 16/19. Mesa, AZ. 2001, p. 524-533.

um usuário de banco de dados, deve ser executado localmente. Em seguida, ele deve retornar para seu local de origem (carregando o resultado do acesso para o *host* que gerou o código de acesso). É importante observar que a arquitetura AMDB fornece o suporte para ambos os tipos da mobilidade.

3.5 Resumo

Neste capítulo, descrevemos o ambiente de computação móvel, suas restrições e os tipos de mobilidade física e lógica, entre outros conceitos. Tais informações fundamentam o próximo capítulo, onde será discutida a tecnologia de Bancos de Dados em ambientes móveis.

CAPÍTULO 4

TECNOLOGIA DE BANCOS DE DADOS EM AMBIENTES MÓVEIS

4.1 Introdução

Este capítulo apresenta as abordagens do processamento de consultas sobre o acesso a banco de dados móveis.

Levando-se em conta que nas CBDM as desconexões são voluntárias ou involuntárias, o que ocasiona uma baixa taxa de entrega dos dados; comprometendo o resultado eficaz de uma consulta, faz-se necessário uma investigação dos principais problemas identificados neste ambiente.

Na seção 4.2 são analisados os principais problemas identificados no processamento de consulta para ambientes móveis.

Na seção 4.3 são analisados operadores adaptativos para o processamento de consultas em ambientes com suporte a mobilidade.

4.2 Processamento de Consulta em Ambientes Móveis

Nesta seção analisaremos o problema de processamento de consulta, em um contexto com suporte à computação móvel; bem como trabalhos publicados sobre o problema de processamento da consulta em ambientes com fontes de dados heterogêneas, como a Internet[12] [13] [14] [15] [16] [17], e processamentos

[12] CHEN, J. et al. **op. cit.,** 2000, p.379-390.

envolvendo grande volume de dados[18] [19]. Estes trabalhos são interessantes porque algumas de suas técnicas podem ser usadas em um ambiente com suporte a mobilidade. As técnicas têm as seguintes características em comum:

(i) Atingir um menor tempo de resposta: no processamento de consulta tradicional verifica-se que a estratégia é minimizar o throughput. As novas abordagens privilegiam a disponibilização das tuplas resultantes tão logo isto seja possível ainda que a técnica ou algoritmo empregado tenha que continuar com o processamento de outras tuplas.

(ii) Adaptar-se às capacidades de processamento das fontes de dados: esta característica garante que uma consulta não deve ser bloqueada por uma fonte de dados com baixa taxa na entrega de dados. Os novos operadores de consulta e os novos planos de execução são propostos para o não bloqueio. Novos algoritmos são propostos para implementar os operadores de consulta. Por exemplo, a operação de junção pode ser executada pelos algoritmos Ripple Join[20] ou Xjoin[21]. Esses algoritmos usam técnicas de simetria que processam cada entrada de dados de uma forma totalmente separada. Os novos planos de

---

[13] DAS, S. et al. op. cit., 2002, p.631-638.

[14] HAAS, et al. **Ripple Joins for Online Aggregation.** Proceedings of the 1999 ACM SIGMOD International conference on Management of Data. May, 31 – Jun, 03. Philadelphia, Pennsylvania, United States,1999, p.287-298.

[15] HAMEURLAIN, A. et al. Mobile query optimization based on agent-technology for distributed datawarehouse and OLAP applications. Proceedings of the 13th International Workshop on Database and Expert Systems Applications. September, 02/06. Aix-en-Provence, France, 2002, p.795-799.

[16] LEE, C-H., CHEN, M-S. **Processing Distributed Mobile Queries with Interleaved Remote Mobile Joins.** IEEE Transactions on Computers, V. 51, Nº 10. October, 2002, p.1182-1195.

[17] Id. **Using Remote Joins for the Processing of distributed Mobile Queries.** Proceeding of the 7th International Conference on Database Systems for Advanced Applications, April 18/21, Hong Kong, China, 2001, p.226-233.

[18] AVNU, R., HELLERSTEIN, Joseph M. **Eddies: Continously Adaptative Query Processing.** Proceedings of the 2000 ACM SIGMOD International conference on Management of Data. May, 15/18. Dallas, Texas, United States, 2000, p.261-272.

[19] LIMTHANMAPHON, B., et al. **An Agent-Based Negotiation Model supporting transactions in Electronic Commerce.** Proceedings of the 11th International Workshop on Database and Expert Systems Applications. September 06/08. Greenwich, London, UK, 2000, p.440-444.

[20] HAAS, et al. **op. cit.**1999, p.287-298.

[21] URHAN, T., FRANKLIN, Michael J. **XJoin: A Reactively-Scheduled Pipelined Join Operator.** IEEE Data Engineering Bulletin. V. 23 Nº 2. June, 2000, p.27-33.

execução favorecem o *pipeline*. Operadores de consulta que suportam a técnica *pipeline* geram tuplas de resultado que são enviadas imediatamente ao operador superior (no plano de execução de consulta), sem nenhuma necessidade de materialização de resultados intermediários. Ou seja, os operadores estão processando continuamente (emitindo e recebendo) tuplas.

(iii) Otimização de consultas durante o processamento: tradicionalmente, a otimização de consulta é estática. O melhor plano deve ser encontrado antes da execução da consulta. O plano não é modificado durante a execução, mesmo se o volume de fonte de dados tenha repentinamente crescido. Assim, pode acontecer da otimização tornar-se ineficaz. As novas abordagens propõem uma otimização passo a passo, de acordo com o contexto corrente das fontes de dados acessadas. Neste caso, o plano inicial de execução pode ser modificado para um outro mais eficiente no contexto corrente. Ele é chamado de processamento adaptativo de consulta.

(iv) Fornecimento de resultados parciais da consulta: esta é uma outra característica muito comum nos trabalhos de pesquisa estudados, conhecidos como consulta *on line* ou consulta contínua. Enquanto os resultados são gerados durante a execução da consulta, eles devem imediatamente retornar ao usuário desde que se garanta sua validade e não duplicação. O usuário pode cancelar a consulta a qualquer hora, sempre que ele suponha ter obtido dados suficientes. O cancelamento da execução da consulta pode também ser feito através de um mecanismo de evento-condição-ação.

Um sistema de processamento de consultas adaptativo é um sistema que muda seu comportamento dependendo da ocorrência de uma mudança no ambiente, com a finalidade de atingir uma determinada meta. Quanto mais imprevisível for o ambiente, mais se necessitará utilizar técnicas de processamento de consulta adaptativo. Observa-se que tais características discutidas anteriormente, sugerem que o uso de mecanismos adaptativos de processamento de consulta em ambientes móveis pode render benefícios, pois, em geral, ambientes de computação móvel apresentam restrições de recursos computacionais além de alta variabilidade do ambiente.

Um mecanismo de processamento de consultas é dito adaptativo[22], quando interage com o ambiente no qual a consulta está sendo executada, obtendo informações do ambiente e reagindo de acordo. Em outras palavras, modificando seu comportamento. Portanto, a eficácia de um sistema adaptativo de processamento de consultas depende de quão bem ele obtém e usa a informação do ambiente para ajustar seu comportamento.

Sistemas de processamento de consulta adaptativos possuem ainda, três aspectos relevantes[23]: a freqüência, os efeitos e a extensão da adaptabilidade. A freqüência de adaptabilidade relaciona-se à freqüência com que o sistema pode receber informação do ambiente (e modificar seu comportamento). Os efeitos da adaptabilidade estão relacionados às mudanças provocadas pelo novo comportamento e o seu próprio custo. A extensão da adaptabilidade diz respeito ao quanto à modificação de comportamento contribui para atingir a meta determinada.

A tabela a seguir sumariza as principais diferenças entre os dois tipos de processamentos:

TABELA 1 -DIFERENÇAS PROCESSAMENTOS ESTÁTICOS X ADAPTATIVO

Processamento Estático	Processamento Adaptativo
Encontrar um plano de consulta ótimo e executá-lo	O plano de consulta pode ser modificado em tempo de execução; dependendo de mudanças no ambiente
Não existe mecanismo de *feedback*	Existe mecanismo de *feedback* e é usado durante a execução da consulta.

[22] HELLERSTEIN, J. M., FRANKLIN, M.J., CHANDRASEKARAN, S., DESHPANDE, A., HILDRUM, K., MADDEN, S., RAMAN, V., E SHAH, M. A. **Adaptative query processing: Technology in evolution**. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 23,*2 (Junho 2000), 7-18.

[23] Id Ibid. (Junho 2000), 7-18.

A adaptabilidade tem sido incorporada em alguns operadores, porém, ainda é considerada um desafio para a comunidade de banco de dados. A seguir descreveremos 7 (sete) operadores que procuram solucionar o problema de processamento de consulta em ambientes de computação móvel.

4.3 Operadores Adaptativos para Processamento de Consulta

4.3.1. Junção Móvel

Lee e Chen[24] [25] propõem o uso do operador de semi-junção unido ao modelo de custo (chamado de junção móvel remota) ao de multi-junção. O modelo proposto de custo leva em conta as características de ambientes móveis. De acordo com Lee e Chen, “a junção móvel remota pode ser eficientemente intercalada ao processamento de consulta para reduzir o custo da transmissão de dados do processamento de multi-junção”. A abordagem apresenta um algoritmo que explora características de assimetria para o processo de consulta móvel. Embora o trabalho de Lee e Chen seja baseado no ambiente móvel no qual a estação de suporte móvel tem um papel chave no processo de consulta, ele não permite operações de junção diretamente entre unidades móveis.

4.3.2 Double Pipelined Hash Join (implementação do Tukwila)

O Tukwila[25] é um processador de consultas projetado para lidar com o atraso de entregar da tupla durante o processamento de consultas sobre diversas fontes de dados, tentando mitigar este retardo.

A idéia chave é intercalar etapas de planejamento e etapas de execução de consulta. Ao executar uma consulta o otimizador não gera planos completos

---

[24] LEE, C-H., CHEN, M-S. **Processing Distributed Mobile Queries with Interleaved Remote Mobile Joins.** IEEE Transactions on Computers, V. 51, Nº 10. October, 2002, p.1182-1195.

[25] Id. Ibid. , 2001, p.226-233.

[25] IVES, Zachary G., et al. op. cit.,1999, p.299-310.

de execução. Uma re-otimização incremental pode então ser feita em qualquer parte do plano dinamicamente. Os planos de execução são divididos em fragmentos, e a re-otimização ocorre no nível do fragmento. No Tukwila, um fragmento pode ser visto como um sub-plano (parte do plano) e pode mesmo conter somente um operador de consulta. Os fragmentos são necessariamente *pipeline*. O otimizador de Tukwila gera uma árvore de consulta anotada. As anotações correspondem às regras de evento-condição-ação em cada nó da árvore, que fornecerá a base para a re-otimização. Os operadores de consulta recolhem a informação estatística sobre cardinalidade das fontes de dados no momento da execução, e quando um evento é ativado, o gerador de consulta usa essa informação para provocar uma ação. A árvore de operadores de consulta é executada em uma forma top-down. O gerador de consulta Tukwila suporta os seguintes operadores de junção: os operadores convencionais tais como a junção nested-loop join, e a junção Hash Join, e o Double Pipelined Hash (DPHJ).

O DPHJ é um operador de junção hash[26] no qual ambas as fontes de dados de entrada sofrem hash independentemente. O operador mantém uma tabela hash para cada origem de dados em memória. Cada uma das fontes envia tuplas para o operador . O operador ao receber a tupla, adiciona-a à tabela hash correspondente e compara-a com a tabela hash da fonte oposta. Este procedimento acontece para tuplas recebidas de ambas as fontes. Sendo simétrico em sua execução, o DPHJ não bloqueia o processamento quando experimenta retardo de liberação de tuplas das fontes.

Todavia, manter tabelas hash (potencialmente grandes) em memória é um ponto fraco do DPHJ. Logo, quando ocorre um estouro de memória, a implementação do DPHJ no Tukwila fornece duas possíveis soluções, dependendo da latência de obtenção dos resultados definida pelo usuário: Maior Latência e Latência Reduzida. No caso da escolha de Maior Latência, o DPHJ se comportará como um algoritmo Hybrid Hash Join, incorrendo em menor I/O e com possibilidade de parada na execução da consulta. Em Latência Reduzida, a implementação do DPHJ no Tukwila usa o Incremental Symetric Flush, no qual as

[26] SILBERCHATZ, A.; et al. **op. Cit.** [S.n.] Makron Books, 1999.

duas tabelas hash são gravadas em disco sem bloquear o processamento de novas tuplas. Neste caso, ambas as fontes de dados continuam a serem processadas em um evento de estouro de memória.

4.3.3 Eddy e suas derivações

Um dos trabalhos de pesquisa os mais influentes é o Avnu[27]. Ele propõe um mecanismo de roteamento de tuplas, chamado Eddy que é interposto entre operadores de consulta e as fontes de dados, de modo que uma consulta continue a execução mesmo se uma ou os mais fontes de dados pararem a entrega de tuplas. A natureza de Eddy é eminentemente simétrica e independe do algoritmo do operador. Logo, Eddy está continuamente pronto para receber tuplas das fontes de dados e repassá-las para um ou mais operadores e receber tuplas de operadores e repassá-las para outros operadores. Avnu destaca que o potencial do Eddy é atingido completamente quando todos os operadores de consulta tiverem algoritmos simétricos. A proposta de Eddy nasceu no projeto River da Universidade da Califórnia, em Berkeley. River é uma máquina de consulta de fluxos de dados que acomoda as flutuações de taxa da entrega dos dados de fontes de dados. Um ponto negativo de Eddy é que seu desempenho depende da política de roteamento, que, por sua vez, é dependente de implementação, de acordo com a abordagem. Quanto melhor for a política do roteamento, melhor será o desempenho de Eddy.

Sendo a política de roteamento o fator crítico de desempenho do Eddy, foram propostas extensões. Uma delas foi a do Juggle Eddy de Raman[28], na qual a política de roteamento é baseada em uma função de custo e é aplicada continuamente tentando responder corretamente a duas questões: “De todas as tuplas recebidas, qual a próxima a ser roteada?” e “Qual o operador a receber a tupla escolhida, dentre todos os operadores possíveis?”. Outra derivação se deu

---

[27] AVNU, R., HELLERSTEIN, Joseph M. **Eddies: Continously Adaptative Query Processing.** Proceedings of the 2000 ACM SIGMOD International conference on Management of Data. May, 15/18. Dallas, Texas, United States, 2000, p.261-272.

[28] RAMAN, V., HELLERSTEIN, Joseph M. **Partial Results for Online Query Processing.** Proceedings of the 2002 ACM SIGMOD International conference on Management of Data. Jun 03/06. Madison, Wisconsin, United States, 2002, p.275-286.

com o STeM Eddy de Maden[29]. Maden propõe uma política baseada em “tickets” na qual quanto mais seletivo for o operador mais tickets ele recebe. Um operador com muitos “tickets” significa que o operador processa rapidamente as tuplas. Assim quanto mais “tickets” um operador recebe mais tuplas podem ser roteadas para ele.

4.3.4 Família de algoritmos Ripple Join

Ripple Join é uma família de algoritmos simétricos de junção e podem ser vistos como uma generalização do algoritmo Nested-Loop Join na qual os papéis de operando interno e externo são continuamente trocados.

Inicialmente proposto para sistemas online de suporte à decisão, Ripple Joins usam técnicas estatísticas de amostragem para decidir sobre as trocas de papeis dos operandos. Note que as tuplas são processadas em ordem aleatória. O usuário controla a freqüência das amostragens e em uma dada amostragem (chamado de passo de amostragem) um operando participa como interno, se este operando fez o papel de operando externo na amostragem anterior.

A família dos Ripple Joins é composta dos seguintes algoritmos: block ripple join, index ripple join e hash ripple join. Um dos pontos fracos dos Ripple Join é a dificuldade de paralelizar sua execução. Outro ponto, especificamente no caso do hash ripple join, é o estouro de memória. Na implementação do hash ripple join original é assumido que o usuário dará por completa a consulta antes que todos os dados tenham sido processados. Logo, quando ocorre um estouro de memória nas tabelas hash, o hash ripple join comporta-se como o block ripple join, perdendo, portanto, em desempenho.

---

[29] MADEN, S., et al. **Continuously Adaptive Continuous Queries over Streams.** Proceedings of the 2002 ACM SIGMOD International conference on Management of Data. Jun, 03/06. Madison, Wisconsin, United States, 2002, p.49-60.

4.3.5 Ajax

O operador adaptativo Ajax[30], propõe minimizar o impacto do processamento de consulta em uma CBDM, por contemplar as seguintes características:

(i) Simetria: AJAX trata cada fonte de dados de forma independente;

(ii) *Pipelining* e *Intra-Operator Pipelining*: o tempo de resposta da consulta é diminuído por meio da geração de tuplas resultado tão logo tuplas de entrada do operador são recebidas, garantindo, dessa forma, um fluxo contínuo de tuplas entre operadores no plano de execução da consulta. O algoritmo fornece também um nível de pipelining intra-operador. Neste caso, há um fluxo contínuo de tuplas entre estágios de execução do algoritmo (por exemplo, geração da tabela *hash* e comparação de tuplas);

(iii) Comparação progressiva (*progressive probing*): a comparação e a distribuição das tuplas em memória são feitas usando estruturas de *hash*. Nestas estruturas, as tuplas são comparadas progressivamente, implicando na diminuição do *overhead* causado pela comparação e pelo *hashing*;

(iv) *Hashing* com buckets de tamanhos variáveis (*dynamic-bucket hashing*): os *buckets* são de tamanho variável implicando em maior flexibilidade no seu gerenciamento;

(v) Prevenção de *overflow* de memória (*memory overflow prevention*): o algoritmo garante um monitoramento constante da memória utilizada, antecipando-se, dessa forma, a eventos de *overflow* de memória.

O operador é dividido em duas fases:

* Fase I

[30] WERBET, E., at al. **AJAX – Um Algoritmo de Junção Adaptativo para Bancos de Dados Móveis.**,2004

No início da primeira fase, uma *thread* é criada para cada fonte de dados envolvida na Junção, alocando uma tabela *hash* correspondente a cada fonte de dados. A tabela *hash* do Ajax é uma lista de *buckets*, de tamanho variável, onde cada *bucket* é uma lista de tuplas.

Com a finalidade de evitar comparações indevidas e duplicações de tuplas o operador Ajax usa a técnica de comparação progressiva. Ressalte-se que o processo de comparação é realizado para um determinado par de *buckets*, enquanto outros pares continuam sendo alimentados por tuplas que estão chegando das relações envolvidas na operação de junção (inclusive o par corrente, caso novas tuplas cheguem), logo, novos *buckets* vão sendo criados e incluídos nas tabelas *hash*.

As tuplas recém-chegadas, e que não foram comparadas na iteração corrente, sê-lo-ão na próxima iteração e assim sucessivamente, para todas as tuplas que chegarem durante a execução do operador.

* Fase 2

Esta fase inicia-se quando as fontes terminam de enviar suas tuplas ou quando se identifica um atraso no envio de tuplas de ambas fontes de dados. O Ajax possui um processo responsável pela monitoração contínua do uso de memória. Uma vez que seu uso ultrapasse uma determinada taxa de ocupação da memória, o Ajax identifica o par de *buckets* com o mesmo endereço que ocupar maior área de memória. A seguir, o par de *buckets* é descarregado em memória secundária, liberando memória para o recebimento de novas tuplas. Assim, após detectado uma das condições anteriores, os *buckets* que foram gravados em disco são lidos e o estágio de comparação é realizado para cada par de *buckets* com o mesmo endereço.

A adaptabilidade ou a não interrupção da execução do operador utiliza estratégia de comparação em 4 passos. Para tanto, considere duas relações R1 e R2 para melhor entendimento dos passos a seguir:

Passo 1: Comparar todas as tuplas dos *buckets* de R1, com tuplas dos *buckets* de R2, ambos armazenados em disco; **Passo 2:** Comparar todas as tuplas dos *buckets* de R1, armazenados em disco, com tuplas dos *buckets* de R2, armazenados em memória; **Passo 3:** Comparar todas as tuplas dos *buckets* de R1, armazenados em memória, com tuplas dos *buckets* de R2, armazenados em disco; **Passo 4:** retornar a execução da Fase 1.

A seguir mostramos a tabela resumo dos tratamentos efetuados pelo operador Ajax, visando minimizar os pontos críticos do ambiente de computação móvel:

TABELA 2 – RESUMO DAS AÇÕES DO OPERADOR AJAX

CARACTERÍSTICA	TRATAMENTO
Overflow de Memória	Dispositivo de monitoramento contínuo de memória, que ao verificar 80% da memória total do sistema computacional utilizada, descarrega em memória secundária o par de *bucket* que ocupar maior área de memória , liberando a área para recebimento de novas tuplas.
Acesso a disco	O acesso a disco ocorre em duas situações: para evitar overf*l*ow de memória ou quando cessarem o envio de tuplas de ambas fontes de dados. É a alternativa para evitar que o atraso no envio de tuplas, interrompa o processamento.
Atraso no fornecimento de tuplas (bloqueio)	O operador possui suporte a simetria, isto é, as fontes de dados envolvidas na junção são processadas independentes uma da outra, assim, caso uma fonte interrompa o fornecimento de tuplas, o operador

	continua recebendo e processando as tuplas da outra fonte de dados; Caso ambas fontes de dados interrompam o fornecimento de tuplas, devido ao término ou atraso do envio de ambas fontes de dados, inicia-se a Fase 2 do operador.
Bucket	Tamanho variável
Fases	Duas fases

4.3.6 MobiJoin

É um operador baseado nos operadores Symmetric Hash Join (SHJ) e Xjoin[30]. Possui como diferencial um mecanismo de controle para evitar a geração de resultados incompletos e/ou com duplicação de tuplas necessitando para tanto que sejam acrescidos um atributo às tuplas; além de um efetivo controle de alocação de *buckets* em memória, reduzindo significativamente a taxa de transferências de *buckets* em memória para o disco o que proporciona uma melhor utilização da memória disponível. Esta é outra grande vantagem do *MobiJoin*[31], pois a memória é um fator crítico em ambientes com suporte à computação móvel.

O operador adaptativo MobiJoin, propõe minimizar o impacto do processamento de consulta em uma CBDM, fundamentado em 03 princípios:

- produção incremental de resultados à medida que os dados são disponibilizados;

- continuidade no processamento da consulta mesmo que a entrega dos dados esteja bloqueada e;

[30] HAAS, Peter J., et al. **op. Cit.**, 2002

[31] VASCONCELOS, E., Brayner, A. **op. cit.,** VI Workshop de Comunicação Sem Fio e Computação Móvel (WCSF), 2004.

- reação a situações de limitação de memória durante a execução do operador.

O operador é dividido em três fases:

* Fase I

Esta fase tem como objetivo operar a maior quantidade de tuplas possível nos espaço de memória disponível no momento da execução da junção. A fase I inicia-se quando as tuplas das fontes de dados envolvidas na junção começam a chegar. É executada enquanto tuplas de pelo menos uma fonte de dados estiverem chegando e termina quando todas as tuplas de ambas fontes de dados tiverem sido recebidas. Caso haja uma interrupção no recebimento das tuplas de ambas as relações, a execução desta fase é suspensa, uma vez que não haverá tuplas disponíveis para processamento da junção e reiniciada quando novas tuplas voltarem a chegar.

Inicialmente, são criadas tabelas *hash* para cada fonte de dados envolvida na operação de junção; tendo uma função *hash* h sobre os atributos da junção. A tabela *hash* é formada por pares de *buckets* correspondentes em memória. Cada *bucket* é formado por um conjunto de tuplas, onde cada tupla é identificada através do identificador único de tupla do bucklet (BTID), que pode ser um número sequencial incrementado à medida que as tuplas vão chegando ao *bucket*. Este identificador é adicionado a todas as tuplas que são alocadas em um *bucket* e também representa a quantidade de tuplas armazenadas no *bucket* até o momento. Os *buckets* armazenados em memória possuem mesmo tamanho. Quando o espaço do bucket for completamente preenchido e existir necessidade de alocar novas tuplas, este é transferido para um *bucket* de memória em disco, liberando memória para recebimento de novas tuplas.

* Fase II

Esta fase inicia-se quando o recebimento de tuplas das duas relações envolvidas na junção for interrompido. Isto posto, um *bucket* em disco é

selecionado e suas tuplas são comparadas com as tuplas do bucket correspondente da fonte de dados oposta armazenada em memória .

Para evitar a produção de tuplas duplicadas, o MobiJoin utiliza uma tabela de controle (matriz de bits), visto que tuplas que estão em disco já podem ter sido comparadas com algumas tuplas do *bucket* da fonte de dados oposta que estão armazenadas em memória. Esta tabela possui os BTIDs das tuplas. As tuplas que já foram processadas possuem BTIDs com valor 1 e as tuplas não processadas possuem valor 0.

Após o processamento de todas as tuplas do *bucket* em disco, o MobiJoin checa se o bloqueio do recebimento dos dados continua. Caso o bloqueio continue, outro *bucket* de disco é processado, caso contrário, a execução desta fase é suspensa e a primeira fase é reiniciada.

* Fase III

Esta fase inicia-se após o recebimento de todas as tuplas das relações envolvidas na operação de junção. Esta fase garante o processamento completo, pois a 1ª e a 2ª fases podem ter computado apenas resultados parciais.

O objetivo desta fase é realizar a junção de todas as tuplas dos pares de *buckets* correspondentes que já foram descarregados em disco e que ainda não foram comparadas. Para melhor entendimento, considere duas fontes de dados R1 e R2. Para cada tabela de controle, o MobiJoin executa os seguintes passos:

Passo 1: lë os valores das células, identificando as tuplas que ainda não foram processadas; **Passo 2:** selecionar as tuplas de um *bucket* em disco de uma relação R1, que não forma processadas, e comparar com as tuplas do *bucket* em memória da relação oposta R2. Para cada tupla lida da relação R1 uma função *hash* h’ é aplicada sobre o atributo de junção para alocar a tupla em uma tabela *hash* em memória; **Passo 3:** após o processamento das tuplas da relação R1, as tuplas não processadas do *bucket* em disco de R2 são lidas e

comparadas com as tuplas não processadas do *bucket* de memória correspondente da relação R1;

Verifica-se que as tuplas de R1, que estavam em disco, foram alocadas em *buckets* de memória com base na aplicação da função *hash* h'. Logo, é necessário comparar as tuplas da relação R1 com as tuplas da relação R2. Para tanto, cada tupla de R2 é aplicada a função *hash* h' sobre o atributo de junção, retornando desta forma o endereço do *bucket* que contém tuplas de R1 e que devem ser comparadas com as tuplas de R2.

A seguir mostramos a tabela resumo dos tratamentos efetuados pelo operador MobiJoin, visando minimizar os pontos críticos do ambiente de computação móvel:

TABELA 3 – RESUMO DAS AÇÕES DO OPERADOR MOBIJOIN

CARACTERÍSTICA	TRATAMENTO
Overflow de Memória	Quando o espaço de memória de um determinado *bucket* X for completamente preenchido e houver necessidade de alocar novas tuplas em X, o conteúdo de X é transferido para um *bucket* de disco, liberando memória para recebimento de novas tuplas.
Acesso a disco	O acesso a disco ocorre em trës situações: 1) quando ocorrer *overflow* de memória; 2) quando o recebimento de tuplas de ambas relações for interrompido e; 3) na Fase III, quando o operador verifica se existem tuplas de um *bucket* de disco ainda não processadas;
Atraso no fornecimento de tuplas (bloqueio)	O operador possui suporte a simetria, isto é, as fontes de dados envolvidas na junção são processadas independentes uma da outra, assim, caso uma fonte interrompa o fornecimento de tuplas, o operador continua recebendo e processando as tuplas da outra

	fonte de dados; Caso ambas fontes de dados interrompam o fornecimento de tuplas, devido ao término ou atraso do envio de ambas fontes de dados, inicia-se a Fase II do operador.
Bucket	Tamanho fixo
Fases	3

4.3.7 XJoin

É um operador adaptativo e que também usa a função *hash*, o *pipelining* e a simetria para evitar o atraso no processamento de tuplas. O Xjoin[32] é fundamentado em dois princípios fundamentais:

- otimização dos resultados. Com base na técnica de *pipelining*, garantindo uma produção incremental de resultados à medida que as tuplas das tabelas envolvidas na junção são disponibilizadas;

- evitar o bloqueio. Garantir a continuidade do processamento quando houver atraso na entrega de tuplas a partir de uma ou mais fontes de dados;

O operador Xjoin possui três estágios:

* Estágio I

Este estágio é similar à fase I do AJAX quando são criadas tabelas *hash* e usado a função *hash* para alocação das tuplas nas tabelas. Em seguida, são efetuadas a junção das tuplas residentes em memória. Este estágio termina quando o operador deixa de receber as tuplas de uma das fontes de dados fontes de dados ou sofre atraso na recepção de tuplas para uma delas.

* Estágio II

Neste estágio, são processadas as tuplas que foram descarregadas em disco. É iniciado quando no estágio I as fontes de dados atrasam a entrega das tuplas e termina quando as tuplas das fontes de dados começam a entregar novamente suas tuplas.

* Estágio III

Este estágio inicia-se quando o primeiro e o segundo estágios terminarem completamente. Este estágio, tem como objetivo garantir que a duplicação de tuplas criadas durante o primeiro e segundo estágios não façam parte do resultado final. A seguir mostramos a tabela resumo dos tratamentos efetuados pelo operador Xjoin, visando minimizar os pontos críticos do ambiente de computação móvel:

TABELA 4 – RESUMO DAS AÇÕES OPERADOR XJOIN

CARACTERÍSTICA	TRATAMENTO
Overflow de Memória	Quando ocorrer um *overflow* um *timestamp* é adicionado a tupla e o *bucket* a qual ela pertence é transferido para o disco, liberando memória.
Acesso a disco	O acesso a disco ocorre em duas situações: 1) Quando ocorrer *overflow* de memória; 2) quando o recebimento de tuplas de ambas relações for interrompido.
Atraso no fornecimento de tuplas (bloqueio)	O operador possui suporte a simetria, isto é, as fontes de dados envolvidas na junção são processadas independentes uma da outra, assim, caso uma fonte interrompa o fornecimento de tuplas, o operador continua recebendo e processando as tuplas da outra fonte de dados;

[32] HAAS, Peter J., et al. **op. Cit.**, 2002

	Caso uma das fontes de dados interrompam o fornecimento de tuplas, devido ao término ou atraso do envio de uma das fontes de dados, inicia-se a Fase II do operador.
Bucket	Tamanho fixo
Fases	3

4.3 Resumo

A adaptabilidade deve ser destacada quando a propriedade da mobilidade é adicionada à tecnologia de banco de dados. Este é o caso dos ambientes cujas configurações sejam muito flexíveis e dinâmicas, tais como as comunidades móveis de banco de dados (CBDM). A arquitetura AMDB realiza um processo de consulta adaptável e dinâmico a fim de alcançar bancos de dados móveis participantes de CBDM, conforme verificaremos no capítulo a seguir.

CAPÍTULO 5

ARQUITETURA DE BANCOS DE DADOS MÓVEIS – AMDB

5.1 Introdução

A arquitetura AMDB[31], é fundamentada no uso de agentes de softwares e prover serviços e funcionalidades para a criação dinâmica de comunidades de banco de dados móveis (CBDM).

Como a arquitetura AMDB utiliza-se de agentes móveis de software, esta independe dos tipos de códigos existentes nos SGBDs, garantindo o suporte a mobilidade, pois os agentes encarregam-se das transformações necessárias visando a compatibilidade, durante a execução das transações no ambiente, desde que os agentes que compõem a arquitetura AMDB sejam instalados na unidade móvel que hospedea um SGBD qualquer.

5.2 Arquitetura AMDB

Conforme descrito em[32], na arquitetura AMDB, existem duas classes de agentes: estáticos e móveis.

---

[31] MORAES FILHO, José Aguiar. op. Cit. (UNIFOR), 2003
[32] Id. Ibid., [S.n.], 2003.

A classe de agente estático é composta pelos agentes Administradores e agentes Mantenedores:

Agentes Administradores – São responsáveis pelo gerenciamento de recursos locais de um computador móvel a serem utilizados durante a execução de agentes móveis visitantes.

Agentes Mantenedores – Fornecem uma interface entre usuários de uma unidade móvel e a plataforma AMDB, além de criarem o contexto de execução para os agentes móveis; logo estes são mediadores da interação entre Agentes Móveis e sistemas de banco de dados dos computadores móveis participantes de uma CBDM.

A classe de agentes móveis é composta por agentes Executores, Carregadores e Promotores:

Agentes Executores – São os responsáveis diretos pela execução de tarefas solicitadas por usuários, como consultas a dados, atualizações de dados entre outras da CBDM.

Agentes Carregadores – São responsáveis pelo transporte de resultados parciais de volta para a unidade móvel que submeteu a consulta e/ou para a unidade móvel eleita como armazenadora temporária de resultados parciais de uma consulta.

Agentes Promotores – Possuem dupla função:

(i) Obter os esquemas dos diversos Bancos de Dados e entregar aos Mantenedores de cada computador móvel e ii) Eleger uma unidade móvel como armazenador temporário dos resultados parciais no caso de computadores móveis com pouca capacidade de processamento.

A seguir, a Figura 4 apresenta o modelo abstrato da arquitetura AMDB, onde estão representados todos os agentes descritos anteriormente, bem como o fluxo de iteração entre eles.

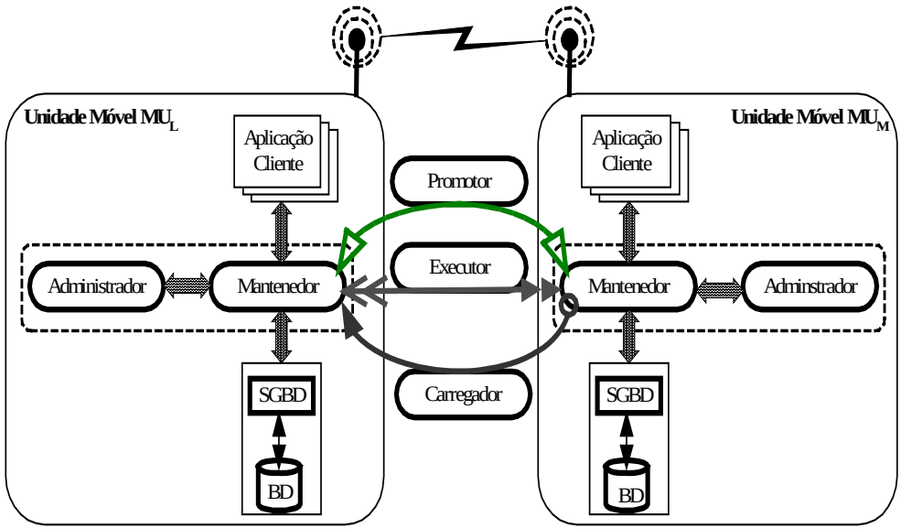


FIGURA 4 – MODELO ABSTRATO DA ARQUITETURA AMDB

Para ilustrar o funcionamento da arquitetura AMDB, suponha duas unidades móveis MU_L e MU_M , e que a unidade móvel MU_M deseja iniciar a composição de uma CBDM. O passo inicial é MU_M declare-se como coordenadora ao seu agente Mantenedor. A seguir, o agente Mantenedor de MU_M envia esta informação sobre a nova comunidade a todos os computadores móveis que estão na área de cobertura do serviço de comunicação sem fio utilizado por MU_M. A partir de ingresso de novos participantes na CBDM, o papel de coordenador exercido por MU_M não é mais necessário, pois a coordenação pode ser executada de forma colaborativa e local por cada agente mantenedor das unidades móveis participantes.

Suponha que a unidade MU_L da Figura 4, deseje participar da CBDM, criada por MU_M . A principal finalidade de uma CBDM é o compartilhamento de

dados, e para tanto, toda nova estação participante deverá efetuar uma declaração explícita para que possa conhecer os esquemas de bancos de dados disponíveis na comunidade. Esta tarefa é executada pelo agente Mantenedor local. Este agente, por sua vez cria uma agente móvel Promotor e delega-o a tarefa de consulta às unidades participantes quanto aos esquemas de BD de cada unidade participante da comunidade. Os esquemas devem ser disponibilizados em XML Schema[33], uma linguagem de definição de esquemas para dados XML. Ao retornar a unidade origem, o agente Promotor entrega o resultado ao agente Mantenedor.

Após os esquemas das unidades serem disponibilizados aos participantes da CBDM, que no nosso exemplo é a unidade MU_L, o agente Mantenedor de MU_L , disponibiliza uma interface para que usuários de programas aplicativos possam solicitar acesso aos bancos de dados da CBDM. As solicitações de acesso são geradas pelo agente Mantenedor em Xquery[34], linguagem de consulta de dados XML. Isto posto, o agente Mantenedor cria um agente Executor na unidade MU_L . Como parâmetros de entrada, o agente Executor recebe a solicitação de acesso em expressão Xquery e esquemas XML dos bancos de dados a serem acessados. Para cada solicitação de acesso, deve ser criado um agente Executor.

Em[35], descreve-se o fluxo e funcionamento dos agentes, conforme exemplificado na Figura 4:

O agente Mantenedor de MU_L cria um agente Executor que deve migrar a $MU_{M,}$ considerando que nesta unidade reside o SGBD que deve processar a consulta especificada em MU_L. Na realidade, o agente Executor deve ter a propriedade de migrar para várias unidades da comunidade. Ao chegar em MU_M, o agente Executor originário de MU_L envia para o agente Mantenedor local uma expressão Xquery. Este agente mantenedor, por sua vez, mapeia a consulta para

---

[33] IVES, Zachary G., et al. **op. cit.**,1999, p.299-310.

[34] Id. Ibid., 2001, p.20-26.

[35] MENDONÇA, N., et al. **op. cit.**, [S.n.], 2003.

a linguagem de consulta nativa do banco local e a submete para o SGBD local como um usuário comum. Ao receber o resultado da consulta local, o agente Mantenedor de MU_M o envia para o agente Executor de MU_L. Como os resultados podem consistir de grandes volumes de dados e como o Executor pode ainda visitar outros participantes, ele cria um agente Carregador e transfere os resultados para este. Ao receber o resultado, o Carregador inicia uma viagem de retorno à unidade de origem do Executor que o criou. Uma vez feito isto, o Executor pode migrar para outras unidades da comunidade. De forma assíncrona, o Carregador leva os dados de volta à unidade MU_L. O Executor, ao retornar à sua unidade de origem, avisa ao agente Mantenedor do término de sua tarefa e se destrói. O agente Mantenedor local da unidade MU_L exibe, então, as informações retornadas ao usuário. Note que a exibição não se dá apenas quando o Executor retorna, mas quando os agentes Carregadores chegam com os dados.

5.3 Processamento de Consulta na Arquitetura AMBD

O agente Mantenedor é o responsável por disponibilizar o esquema local aos demais integrantes da Comunidade. Este esquema conterá além dos metadados, dados do catálogo do BD local necessários ao processamento de consultas, tais como: cardinalidades das tabelas, existência de índices, entre outros. Assim, um computador que deseje participar de uma CBDM, enviará a cada integrante da comunidade, através do seu agente Promotor, o seu esquema local, que retornará os esquemas locais das unidades visitadas. A partir de então a unidade terá todas as informações possíveis para a execução de uma consulta aos vários bancos de dados pertencentes a comunidade.

Numa CBDM, uma consulta poderá ser local ou distribuída. A consulta local envolve apenas objetos do BD local da unidade. Neste caso, o agente Mantenedor solicitará ao SGBD local que execute a consulta e retornará o resultado ao usuário. O processamento e a otimização da consulta ficará a cargo do SGBD local.

Uma consulta distribuída poderá envolver vários bancos de dados residentes em mais de uma unidade móvel, inclusive a unidade originadora da consulta. A consulta inicia-se quando um usuário ativa o agente mantenedor com uma consulta. O Agente Mantenedor ao detectar que a consulta é distribuída, criará um agente móvel executor e o encarregará da execução da consulta. O agente Mantenedor fornecerá ao agente executor a consulta em formato Xquery[36]. O agente Executor, então efetuará a otimização da consulta baseada nos esquemas de bancos de dados a serem acessados, já conhecidos pela unidade, na quantidade de unidades móveis envolvidas e localização dos dados, no volume dos dados em cada objeto de banco afetado, bem como nas restrições de integridades conhecidas. A otimização é global e o processamento da consulta deve ser dinâmico e adaptativo ao contexto de execução da consulta.

O processamento de consultas na arquitetura AMDB é composto de seis fases, conforme visto na figura 5, a saber:

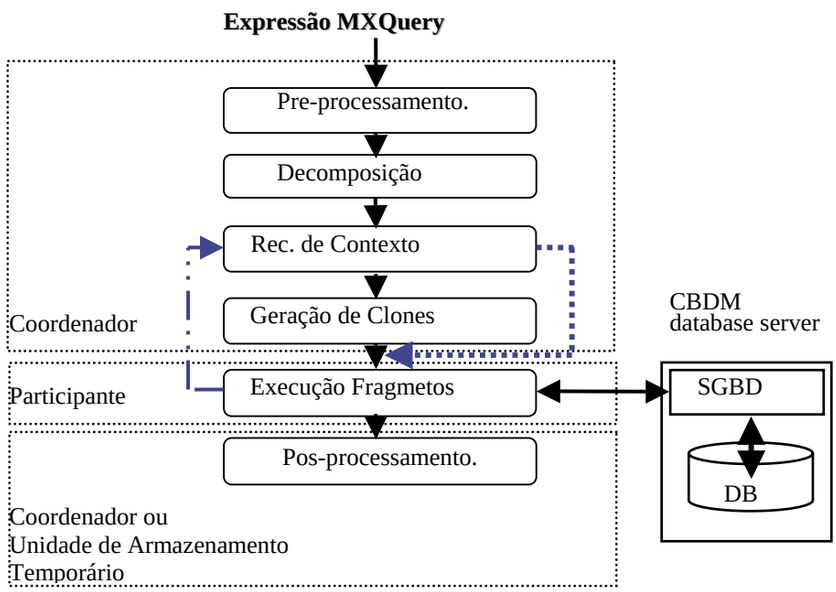


FIGURA 5 – FASES DO PROCESSAMENTO DE CONSULTA DA ARQUITETURA AMDB

[36] Id. Ibid., 2001, p.20-26

Pré-processamento – O agente Executor, responsável pela execução de uma consulta, recebe uma consulta Xquery e mapeará a expressão Xquery em um Plano de Consulta (PEC). Um PEC é representado através de uma árvore de operadores[37];

Decomposição – O agente Executor inicia esta fase, particionando em fragmentos[38]. Cada fragmento pode representar recursivamente uma sub-árvore cujo atributo de localidade refere-se a uma unidade móvel na rede, na qual o fragmento da consulta deve ser executado;

Reconhecimento de Contexto – Nesta fase, o agente Executor identificará se o SGBD responsável pela execução do fragmento possui recursos necessários para processar o fragmento. Caso um SGBD não apresente suporte para realizar uma determinada operação, esta deverá ser executada pelo agente Executor. A execução será realizada na unidade de origem ou em uma unidade de armazenamento temporariamente eleita. Para tanto, faz-se necessário uma reorganização do PEC, para que as operações a serem executadas pelo agente Executor localizem-se em nós de níveis mais altos. O nó que representa a última operação a ser executada pela consulta é considerado o nó de nível mais alto;

Fase de Geração de Clones – Nesta fase, o agente Executor criará clones que serão responsáveis pela execução dos fragmentos gerados a partir da fase de decomposição. Isto posto, cada um deles será enviado a uma unidade móvel para a execução do fragmento. Durante a migração, cada clone carregará o fragmento de sua consulta;

Fase de Execução de Fragmentos - Esta fase inicia-se no instante que um clone do agente Executor migrar para a unidade móvel onde deverá ser executado o fragmento de consulta. Ao chegar na unidade móvel, o clone fornecerá seu fragmento de consulta ao agente Mantenedor local em formato Xquery. O agente Mantenedor, funcionando como um wrapper, converterá o

---

[37] BARISH, G., et al. **An Efficient Plan Execution System for Information Management Agents.** Proceedings of the second International Workshop on Web Information and Data Management. November, 02/06. Kansas City, Missouri, United States, 1999, p.1-5.

[38] SILBERCHATZ, A.; et al. **op. cit.**, 1999.

fragmento para linguagem de consulta nativa do SGBD da unidade, e pedirá ao SGBD para executa-la e retornará os dados para o agente móvel (o clone do agente Executor responsável pela execução da consulta global). A otimização local da consulta será realizada pelo SGBD local;

Pós-processamento – Esta fase é iniciada quando os clones começam a retornar a unidade de origem. Durante esta fase, os clones entregam os resultados parciais ao agente Executor que poderá iniciar a execução das operações que não puderam ser executadas pelos SGBDs locais nas unidades móveis. O agente Executor não aguardará que todos os resultados parciais dos fragmentos retornem para iniciar suas operações; ele iniciará a execução das operações de mais alto nível da árvore de operadores o mais rápido. Terminada a execução, o agente Executor projetará os resultados finais para o usuário na unidade de origem.

5.5 Resumo

Neste capítulo apresentamos uma breve descrição do funcionamento da arquitetura AMDB, enfatizando seu suporte a mobilidade e as particularidades do processamento de consulta para o ambiente com suporte a mobilidade. No capítulo seguinte, descreveremos regras (evento-condição-ação) para otimização durante o processamento de consulta nesta arquitetura.

CAPÍTULO 6

OTIMIZAÇÃO DO PROCESSAMENTO DE CONSULTA DA ARQUITETURA AMDB USANDO REGRAS (EVENTO-CONDIÇÃO-AÇÃO)

6.1 Introdução

Neste capítulo analisaremos os operadores adaptativos de junção: Ajax,[39], MobiJoin[40],e Xjoin,[41], mediante critérios que impactam ou inviabilizam o processamento de consultas na arquitetura AMDB. Serão propostas um conjunto de regras para garantir um melhor desempenho do processamento de consulta na arquitetura AMDB, que é o objetivo deste trabalho. Este capítulo está assim estruturado. Na seção 6.2 analisaremos os operadores adaptativos de junção Ajax,[42], MobiJoin[43],e Xjoin,[44], fazendo uma comparação entre as principais características de cada um, visando propor aquele que mais se adapta um ambiente com suporte a mobilidade como a arquitetura AMDB. Na seção 6.3 apresentamos uma proposta para otimizar situações de desconexões voluntárias

---

[39] WERBET, E., at al. **AJAX – Um Algoritmo de Junção Adaptativo para Bancos de Dados Móveis.**,2004.

[40] VASCONCELOS, E., Brayner, A. **op. cit.,** VI Workshop de Comunicação Sem Fio e Computação Móvel (WCSF), Outubro2004..

[41] URHAN, T., FRANKLIN, Michael J. **op. cit.**, 2000, p.27-33.

[42] WERBET, E., at al. **op**.cit.,2004

[43] VASCONCELOS, E., Brayner, A. **op. cit.,** VI Workshop de Comunicação Sem Fio e Computação Móvel (WCSF), Outubro2004

[44] URHAN, T., FRANKLIN, Michael J. **op. cit.**, 2000, p.27-33.

ou involuntárias de unidades móveis pertencentes uma CMDB, usando a arquitetura AMDB.

6.2 Análise comparativa dos Operadores Adaptativos

A arquitetura AMDB, por ser um ambiente de computação móvel, está sujeito a eventuais desconexões de seus participantes, durante um processamento de consulta. A arquitetura propõe o uso de operadores adaptativos no processamento de consulta tais como os estudados no capítulo 4. Sob este aspecto, a tabela a seguir resume as principais características dos operadores Ajax,[45], MobiJoin[46],e Xjoin,[47], quanto aos critérios de vazão no recebimento de tuplas e memória disponível, durante um processamento de consulta em ambientes com suporte a mobilidade como a arquitetura AMDB:

TABELA 5 – COMPARATIVO ENTRE OPERADORES ADAPTATIVOS

OPERADOR	**CARACTERÍSTICAS**					
	Simetria	Fases	*Bucket*	Acesso Disco	*Overflow*	*Pipelining*
Ájax	Sim	2	Variável	2 situações	Prevenção	Sim
MobiJoin	Sim	3	Fixo	3 situações	BTID	Sim
Xjoin	Sim	3	Fixo	2 situações	*Timestamp*	Sim

Na arquitetura AMDB, o agente Executor é o responsável pelo monitoramento dos eventos relacionados a processamento de consulta. Caso haja problemas durante o processamento de consulta, como uma demora na entrega das tuplas, o Executor reavalia o PEC e pode escolher outro operador, para continuar o processamento de consulta, solucionando o problema no atraso de tuplas.

---

[45] WERBET, E., at al. **AJAX – Um Algoritmo de Junção Adaptativo para Bancos de Dados Móveis.** 2004.

[46] VASCONCELOS, E., Brayner, A. **op. cit.,**VI Workshop de Comunicação Sem Fio e Computação Móvel (WCSF), Outubro2004

[47] URHAN, T., FRANKLIN, Michael J. **op. cit.**, 2000, p.27-33.

6.3 Regras para o Processamento de Consulta da Arquitetura AMDB

Quando se submete uma consulta, a meta do agente Executor é encontrar o melhor plano global possível que apresente melhor desempenho no processamento da consulta. Este desempenho é avaliado levando-se em consideração a vazão de tuplas e utilização de memória.

A tabela a seguir, estabelece um conjunto de regras (evento-condição-ação) para garantir um melhor desempenho durante o processamento de consulta na arquitetura AMDB:

TABELA 6 – REGRAS PARA PROCESSAMENTO CONSULTA

EVENTO	CONDIÇÃO	AÇÃO
Estatísticas Imprecisas	Frequência irregular na entrega de tupals Se N*B>= 0,8M	Reavaliar o PEC e substituir o operador em uso.
Fonte de Dados Indisponível	Indisponibilidade total	Usar operador *outer join*
	Indisponibilidade parcial	Processar *buckets* do disco. Os operadores adaptativos efetuam essa ação com eficácia.
Overflow de memória	Se N*B>= 0,8M	Antecipar-se ao overflow, descarregando buckets no disco.
MU não dispõe recursos para processar consulta	Memória insuficiente	Reformular o PEC – Fase Reconhecimento de contexto.

Legendas: N- número de *buckets* em memória;
B- tamanho do *bucket* ;
M- memória disponível.

6.3.1 Regra para Estatísticas imprecisas

Conforme demonstrado na tabela, durante o processamento de consulta na arquitetura AMDB, existem duas condições relacionadas a este evento. Para

solucionar o problema de *overflow* de memória, quando um agente Administrador detectar um percentual, de aproximadamente, 80% (Se N*B>= 0,8M) de uso do total da memória de uma MU envolvida no processamento de consulta, tenha como ação avisar ao agente Executor para que este possa executar uma ação de reavaliação do PEC, substituindo o operador em uso (por exemplo, o Mobijoin) pelo operador Ajax. Entre os três operadores comparados na seção anterior, o Ajax possui uma prevenção de *overflow* quando o uso de memória chega ao valor de 80%. Verifica-se que este é o que melhor se adapta a esta situação; visto que tanto o Mobijoin quanto o Xjoin só executam a ação de descarregar em disco os *buckets*, após ocorrer o *overflow.*

Quanto a outra condição, frequência irregular na entrega de tuplas, para solucionar o problema, sugerimos que quando o agente Executor detectar que está havendo uma entrega de tuplas com freqüência irregular, isto é, não está ocorrendo um intervalo constante na entrega de tuplas, o Executor reavalia o PEC e substitui o operador em uso pelo operador Mobijoin. Entre os operadores em estudo, este é o que melhor se adapta a esta situação, pois; na 2ª fase o Mobijoin ao término do processamento de cada *bucket* armazenado em disco, checa se o bloqueio das fontes de dados continua, e só após essa checagem, retorna a primeira fase, caso o bloqueio tenha finalizado, ou processa o *bucket* seguinte do disco.

6.3.2 Regra para Fonte de Dados Indisponível

Conforme demonstrado na tabela, a este evento também existem duas condições relacionadas: fonte de dados indisponíveis parcialmente ou totalmente.

- **Fonte parcialmente indisponível**

Quando uma consulta é submetida, o plano de consulta é fragmentado em função principalmente da localização dos dados. Durante a execução da consulta, uma ou mais fonte de dados podem está temporariamente indisponíveis. Quanto este evento ocorrer, deverá ser disparada uma ação visando evitar o

bloqueio do processamento de consulta. Os três operadores estudados, usam *pipelining* para evitar o bloqueio do processamento de consulta.

- **Fonte totalmente indisponível**

Quando ocorrer a indisponibilidade total de uma das fontes de dados, sugerimos usar a operação *outer join*, visando dar continuidade ao processamento.

6.3.3 Regra para *Overflow* de Memória

Conforme demonstrado na tabela, a este evento existe uma condição relacionada: se o comprometimento da memória chegar a 80% (Se N*B >= 0.8M), verifica-se que os três operadores possuem tratamento ao *overflow* de memória, mas apenas o operador Ajax possui um processo responsável pela monitoração contínua de memória que antecipa-se ao *overflow*. A ação prevista para este evento é o descarregamento de *buckets* em disco, liberando memória para recebimento de novas tuplas a serem processadas.

Estudos que tratam do particionamento e armazenamento de arquivos entre memória e disco, nos apresentam regras[48] que adaptamos para o monitoramento do espaço em memória visando evitar o *overflow* durante o processamento de consulta na arquitetura AMDB.

O cálculo do número máximo de partições do arquivo a ser armazenado em disco, simbolizados pela letra **F**[50], é dado pela equação:

F = $\lfloor M / C - 1 \rfloor$**;** onde **C** é o tamanho mínimo de um grupamento para partição.

O número mínimo de partições de arquivos requerido para armazenamento em disco é denominado pela letra **K**, que deverá satisfazer a condição: $0 \leq K \leq F$.

---

[48] GOETZ GRAFE. Query Evaluation Techiniques for Large/Databases. ACM Computing Surveys, Vol. 25, Nº 2, june 1993.
[50] Id Ibid. (June 1993).

Adaptando essa idéia para o monitoramento de memória visando evitar o *overflow* de memória durante o processamento de consulta na arquitetura AMDB, estabelecemos um algoritmo para executar este controle:

Legendas:

R – relação a ser processada;

M – memória disponível;

F – nº máximo de partições do arquivo a ser armazenado em disco;

K - nº minimo de partições do arquivo a ser armazenado em disco;

ALGORITMO MONITORAMENTO

```
SE R < M então
    <processamento em memória sem problemas>;
SENÃO
    SE R = M então
        ❑ Passo 1 – calcular K;
        ❑ Passo 2 – calcular F;
        SE K <= F então
            <reservar (K + F)/2 partições em disco>;
        SENÃO
            < solicitar processamento colaborativo>;
        FIM-SE;
    SENÃO
        < solicitar processamento colaborativo>;
    FIM-SE;

FIM.
```

6.3.4 Regra para Indisponibilidade de Recurso de Processamento

Conforme demonstrado na tabela, para este evento existe uma condição relacionada: memória insuficiente da MU para processamento da consulta ou fragmento. Na arquitetura AMDB, o Agente Executor é o responsável pela verificação da existência de recursos necessários para a execução das operações de processamento. Por exemplo, caso uma MU não possua recursos necessários ao processamento de um fragmento este será executado numa unidade de armazenamento temporária; bem como a reorganização do PEC.

É importante ressaltar que na mudança entre os operadores, devem ser garantidas as fases de processamento efetuadas pelo operador que foi substituído e entregues ao novo operador escolhido. As ações propostas na mudança de operador, visando dar continuidade ao processamento, são:

a) Garantir os repasses das tuplas já lidas e não processadas;

b) Garantir o repasse das tuplas já processadas;

Informar a posição corrente de cada fonte de dados (endereço), isto é, os endereços das últimas tuplas lidas de cada fonte de dados.

6.4 Regras para Desconexões de MU na arquitetura AMDB

Nesta seção, analisaremos duas situações referentes a conexões/desconexões das unidades no cenário a seguir:

Suponha as unidades móveis MU1, MU2 e MU3, e que MU1 deseja iniciar a composição de uma CBDM. O passo inicial é MU1 declare-se como coordenadora ao seu agente Mantenedor. A seguir, o agente Mantenedor de MU1 envia esta informação sobre a nova comunidade a todos os computadores móveis que estão na área de cobertura do serviço de comunicação sem fio utilizado por MU1. A partir do ingresso de novos participantes, MU2 e MU3, na CBDM, o papel

de coordenador exercido por MU1 não é mais necessário, pois a coordenação pode ser executada de forma colaborativa e local por cada agente mantenedor das unidades móveis participantes.

Toda unidade que desejar participar da CBDM deverá efetuar uma declaração explícita para que possa conhecer os esquemas de bancos de dados disponíveis na comunidade. A essa declaração explícita, chamaremos de inscrição. Assim, se uma unidade móvel desconectar-se da comunidade, por qualquer motivo, ao retornar a comunidade deverá fazer novamente sua Inscrição. O agente promotor da nova unidade móvel, consulta às unidades participantes quanto aos esquemas de BD de cada participante da CBDM. Nesse momento, é verificado se o esquema da MU visitada já foi armazenado e caso positivo, não haverá necessidade de copiar novamente o esquema. Caso contrário, repassará seu esquema de BD e copiará os esquemas de BD de cada participante visitado.

A partir do cenário acima descrito, analisaremos duas situações de Desconexão. Ressaltamos que a desconexão caso não disponha de nenhum tratamento, ocasionará dentre os principais problemas, que os participantes da CBDM, por falta de informação, podem efetuar consultas a unidades móveis que não fazem mais parte da CBDM, ocasionando processamento desnecessários. A tabela a seguir descreve um resumo das regras (evento-condição-ação) para as desconexões:

TABELA 7 – REGRAS PARA DESCONEXÕES

Evento	Condição	Ação
Desconexão	Voluntária	Sequência de passos da MU que sai.
		Sequência de passos das MUs que permanecem.
	Involuntária	Enviar *broadcast* para CBDM
		Sequência de passos das MUs que permanecem.
		Efetuar nova Inscrição

6.4.1 Regras para Desconexão Voluntária

Entende-se desconexão voluntária aquela onde a MU deseja sair, por vontade própria, da CBDM. Por exemplo, o usuário da MU deseja explicitamente se desconectar da CBDM. Conforme a tabela resumo, existem dois grupos de ações: para a MU que está saindo da CBDM, e para as MUs que permanecem na CBDM.

- **Ações da MU que saindo da CBDM**

Passo1: Avisar ao Agente Mantenedor para não receber mais requisições; **Passo2:** Enviar *broadcast* informando saída da comunidade; **Passo3:** Avisar ao agente executor que a MU vai ser desconectada; **Passo4:** Caso a MU esteja participando de um processamento de consulta, aguardar o processamento até o final;

- **Ação das MUs que permanecerem na CBDM**

Ao receber *broadcast* informando a saída da CBDM de uma MU, eliminar da lista local, o esquema da unidade móvel correspondente.

6.4.2 Regras para Desconexão Involuntária

Entende-se desconexão involuntária aquela onde a MU não deseja sair, por vontade própria, da CBDM (desconexão por problemas de descarregamento da bateria, por exemplo). Neste caso teremos um grupo de ações, a saber:

- **Ação da primeira unidade móvel que detectar a desconexão**

Quando uma unidade móvel detectar a desconexão de outra, enviar um *broadcast* informando a saída daquela unidade a comunidade.

- **Ações das MUs que permanecerem na CBDM**

Passo1: Caso a MU esteja participando de um processamento de consulta com a MU que se desconectou, avisar ao agente Executor para usar a semântica de *outer-join*; **Passo2:** Eliminar da sua lista local, o esquema da unidade móvel que saiu da CBDM.

- **Ação das MUs que retornarem a CBDM**

Efetuar nova Inscrição.

Apresentamos a seguir, um quadro resumo de análise das regras propostas para desconexões de unidades móveis da arquitetura AMDB:

TABELA 8 – VANTAGENS X DESVANTAGENS DO CONTROLE DE DESCONEXÃO

VANTAGENS	DESVANTAGENS
A comunidade continua sendo colaborativa	Caso mais de uma unidade móvel detectar a desconexão de uma unidade móvel, simultaneamente, ocasionará duplicidade de *broadcast*
A arquitetura continua a mesma	-
Evitar consultas inválidas, visto que a unidade móvel não mais pertence a CBDM.	-

6.5 Resumo

Neste capítulo apresentamos um conjunto de regras (evento-condição-ação) para uso na arquitetura AMDB, divididas em ações para o processamento de consulta e ações para desconexões de unidades móveis. No capítulo seguinte, faremos as considerações finais sobre a contribuição deste trabalho.

CAPÍTULO 7

CONCLUSÃO

7.1 Considerações Sobre as Regras Propostas

A computação móvel ainda enfrenta desafios a serem superados, como problemas de hardware e *software*, visando o compartilhamento e o gerenciamento das informações disponíveis.

A realização deste trabalho envolveu o estudo dos principais conceitos relacionados ao ambiente de computação móvel, incluindo recentes trabalhos propostos de operadores adaptativos de processamento de consulta, com o intuito de escolher uma estratégia adequada para o processamento de consulta da arquitetura AMDB, observando-se:

- Reduzir o tempo de resposta de uma consulta;

- Processar continuamente as tuplas, evitando o bloqueio da consulta, independente do atraso ou baixa taxa de entrega de dados;

- Fornecer dados parciais de consulta ao usuário;

A partir desses problemas expostos, verifica-se que as técnicas ou operadores tradicionais de processamento de consulta, não são aplicáveis a uma CBDM, como proposto na arquitetura AMDB. Este trabalho propôs, eventos a serem disparados quando identificados problemas no processamento de consulta, tais como restrição de memória e atraso na entrega de tuplas.

A tabela abaixo, apresenta um resumo comparativo das principais características dos operadores estudados.

TABELA 9 – RESUMO COMPARAÇÕES OPERADORES ADAPTATIVOS

OPERADOR	CARACTERÍSTICAS					
	Simetria	Fases	*Bucket*	Acesso Disco	*Overflow*	*Pipelining*
Ajax	Sim	2	Variável	2 situações	Prevenção	Sim
MobiJoin	Sim	3	Fixo	3 situações	BTID	Sim
Xjoin	Sim	3	Fixo	2 situações	*timestamp*	Sim

Conforme visto no Capítulo 4, ao submetemos uma consulta na arquitetura AMDB, o agente Executor, procura o plano de consulta que apresente o melhor desempenho considerando-se a vazão de tuplas e utilização de memória. Caso ocorram atrasos no fornecimento de tuplas, o agente Executor, após uma reavaliação, pode escolher um outro operador, solucionando o problema de vazão de tuplas.

Uma outra contribuição deste trabalho, refere-se a desconexões voluntárias ou não de unidades móveis usuários da arquitetura AMDB. As ações propostas, objetivam:

- Reduzir as atividades de atualização de esquemas dos bancos de dados das MU participantes, quanto ao compartilhamento de bancos de dados;

- Garantir o ambiente colaborativo proposto na AMDB;

- Evitar consultas inválidas; isto é, consultas a membros que já deixaram a comunidade;

Espera-se que, as ações propostas aos eventos de *Estatísticas Imprecisas e Fonte de Dados Indisponível,* em conjunto com o controle das desconexões de membros da comunidade, contribuam efetivamente para um melhor desempenho da funcionalidade da arquitetura AMDB. O uso da flexibilidade na substituição dos planos e operadores durante o processamento na arquitetura, associado a discriminação das ações efetivas do agente Executor durante a reformulação do PEC e rota, garantem a integridade dos dados manipulados.

7.2 Trabalhos Futuros

Como trabalhos futuros, sugere-se:

- Estender o controle de desconexões das MU participantes da comunidade;

- Desenvolver novos operadores de processamento de consulta adaptativos dinamicamente ao contexto atual do processamento;

- Definir mecanismos efetivos de estatísticas durante a execução da consulta na arquitetura AMDB;

- Identificar novos eventos que contribuam com a otimização do processamento de consulta da arquitetura AMDB.

REFERÊNCIAS BIBLIOGRÁFICAS

ANCIAUX, N., et al. **PicoDBMS: Validation and Experience.** Proceedings of the 27th International Conference on Very Large Databases. September 11/14. Roma, Italy, 2001, p.709-710.

ARIDOR, Y. LANGE, Danny B. **Agent Design Patterns: Elements of Agent Application Design.** Proceedings of the second International Conference on Autonomous Agents. May, 10/13. Minneapolis, Minnesota, United States, 1998, p.108-115.

AVNU, R., HELLERSTEIN, Joseph M. **Eddies: Continously Adaptative Query Processing.** Proceedings of the 2000 ACM SIGMOD International conference on Management of Data. May, 15/18. Dallas, Texas, United States, 2000, p.261-272.

BARISH, G., et al. **An Efficient Plan Execution System for Information Management Agents.** Proceedings of the second International Workshop on Web Information and Data Management. November, 02/06. Kansas City, Missouri, United States, 1999, p.1-5.

BOBINEAU, C., et al. **PicoDBMS: Scaling down Database Techniques for the Smartcard.** Proceedings of the 26th International Conference on Very Large Databases. September 10/14, Cairo, Egypt, 2000, p.11-20.

BOUGANIM, L. et al**. A Dynamic Query Processing Architecture for Data Integration.** IEEE Data Engineering Bulletin, V.23 Nº 2, June 2000, p.42-48.

CHEN, J. et al. **NiagaraCQ: A Scalable Continuous Query System for Internet Databases.** Proceedings of the 2000 ACM SIGMOD International conference on Management of Data. May, 15/18. Dallas, Texas, United States, 2000, p.379-390.

DAS, S. et al. **ACQUIRE: Agent-based Complex Query and Information Retrieval Engine.** Proceedings of the first International joint Conference on Autonomous Agents and Multiagent Systems. July 15/19. Bologna, Italy, 2002, p.631-638.

DATE, C. J. **Introdução a Sistemas de Banco de Dados.** [S.n.], Campus, 1998.

DRAPER, D., et al. **The Nimble Integration Engine.** Proceedings of the 2001 ACM SIGMOD International conference on Management of Data. May 21/24. Santa Barbara, California, United States, 2001, p.567-568.

ELMAGARMID, A., Bouguettaya, A., Benatallah, B. **Interconnecting Heterogeneous Information Systems.** Kluwer Academic Publishers, 1998.

FLORESCU, D., et al. **Query Optimization in the Presence of Limited Access Patterns.** Proceedings of the 1999 ACM SIGMOD International conference on Management of Data. May 31 – Jun 03. Philadelphia, Pennsylvania, United States,1999, p.311-322.

GOLDMAN, R. WIDOM, J. **WSQ/DSQ: A Pratical Approach for Combined Querying of Databases and the Web.** Proceedings of the 2000 ACM SIGMOD International conference on Management of Data. May, 15/18, Dallas, Texas, United States, 2000, p.285-296.

HAAS, et al. **Ripple Joins for Online Aggregation.** Proceedings of the 1999 ACM SIGMOD International conference on Management of Data. May, 31 – Jun, 03. Philadelphia, Pennsylvania, United States,1999, p.287-298.

_____. **A Scalable Hash Ripple Join Algorithm.** Proceedings of the ACM SIGMOD International Conference on Management of Data. June 4/6, Madison, Wisconsin, USA, 2002.

HELLERSTEIN, J. M., FRANKLIN, M.J., CHANDRASEKARAN, S., DESHPANDE, A., HILDRUM, K., MADDEN, S., RAMAN, V., E SHAH, M. A. **Adaptative query**

processing: Technology in evolution. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 23,*2 (Junho 2000), 7-18.

HAMEURLAIN, A. et al. Mobile query optimization based on agent-technology for distributed datawarehouse and OLAP applications. Proceedings of the 13th International Workshop on Database and Expert Systems Applications. September, 02/06. Aix-en-Provence, France, 2002, p.795-799.

HONG, W., STONEBRAKER, M. **Optimization of Parallel Query Execution Plans in XPRS.** Distributed and Parallel Databases. V.1 Nº 1, 1993, p.9-32.

IVES, Zachary G., et al. **An Adaptative Query Execution System for Data Integration.** Proceedings of the 1999 ACM SIGMOD International conference on Management of Data. May 31 – Jun 03. Philadelphia, Pennsylvania, United States,1999, p.299-310.

_____. **Integrating Network-Bound XML Data.** IEEE Data Engineering Bulletin. V.24, Nº 2, June, 2001, p.20-26.

LANGE, D. B., OSHIMA M. **Programming and Deploying Java Mobile Agents with Aglets.** Addison-Wesley. Massachusetts, USA. 1998. ISBN 0201325829.

LEE, C-H., CHEN, M-S. **Processing Distributed Mobile Queries with Interleaved Remote Mobile Joins.** IEEE Transactions on Computers, V. 51, Nº 10. October, 2002, p.1182-1195.

_____. **Using Remote Joins for the Processing of distributed Mobile Queries.** Proceeding of the 7th International Conference on Database Systems for Advanced Applications, April 18/21, Hong Kong, China, 2001, p.226-233.

LIMTHANMAPHON, B., et al. **An Agent-Based Negotiation Model supporting transactions in Electronic Commerce.** Proceedings of the 11th International Workshop on Database and Expert Systems Applications. September 06/08. Greenwich, London, UK, 2000, p.440-444.

MACKER, J. P., M. S. Corson. **Mobile Ad Hoc Networks and the IETF.** Internet Engineering Task Force, MANET Working Group. Available online at http://www.ietf.org/html.charter/manet-charter.html.

MADEN, S., et al. **Continuously Adaptive Continuous Queries over Streams.** Proceedings of the 2002 ACM SIGMOD International conference on Management of Data. Jun, 03/06. Madison, Wisconsin, United States, 2002, p.49-60.

MARGARET, Dunham H.; ABDELSALAM, HELAL (Sumi). **Mobile Computing and Databases: Anything New?** SIGMOD Record, V. 24. Nº 4, December, 1995.

MENDONÇA, N., et al. **Mobile Database Communities: An Approach for Sharing Autonomous Mobile Databases in Ad Hoc Networks**, [S.n.], 2002.

_____. **Sharing Móbile in Dynamically Configurable Environments.** [S.n.t.].

MCHUGH, J., WIDOM, J. **Query Optimization for XML.** Proceedings of the 25th International Conference on Very Large Databases. September 7 – 10. Edinburgh, Scotland, UK, 1999, p.315-326.

MORAES FILHO, José Aguiar. **AMDB – An approach for Sharing Mobile Databases. Master Dissertation.** Universidade de Fortaleza (UNIFOR), 2003.

MURPHY, A. L., et al. **A Middleware for Physical and Logical Mobility.** Proceedings of the 21st International Conference on Distributed Computing Systems. April 16/19. Mesa, AZ. 2001, p. 524-533.

NAUGTON, J. et al. **The Niagara Internet Query System.** IEEE Data Engineering Bulletin. V.24, Nº 2, June, 2001, p.27-33.

RAMAN, V., HELLERSTEIN, Joseph M. **Partial Results for Online Query Processing.** Proceedings of the 2002 ACM SIGMOD International conference on Management of Data. Jun 03/06. Madison, Wisconsin, United States, 2002, p.275-286.

SATTLER, K., et al. **Adding Conflict Resolution Features to a Query Language for Databases Federations.** Proceedings of the 3rd International Workshop on Engineering Federated Information Systems. June, 19/21, Dublin Ireland, 2000.

SHANMUGASUNDARAM, J., et al**. Architecting a Network Query Engine for Producing Partial Results.** Proceedings of the Third International Workshop on the Web and Databases. May, 18/19, Dallas, Texas, United States, 2000.

SILBERCHATZ, A.; et al. **Sistema de Banco de Dados.** [S.n.] Makron Books, 1999.

TUFTE, K., MAIER, D. **Aggregation and Accumulation of XML Data.** IEEE Data Engineering Bulletin. V.24, Nº 2, June, 2001, p.34-39.

URHAN, T., FRANKLIN, Michael J. **XJoin: A Reactively-Scheduled Pipelined Join Operator.** IEEE Data Engineering Bulletin. V. 23 Nº 2. June, 2000, p.27-33.

VASCONCELOS, E., Brayner, A. **MobiJoin: Um Operador de Junção para Bancos de Dados Móveis.** VI Workshop de Comunicação Sem Fio e Computação Móvel (WCSF), 2004.
.

XQUERY 1.0 Formal Semantics. Available online at http://www.w3.org/TR/query-semantics.

WERBET, E., at al. **AJAX – Um Algoritmo de Junção Adaptativo para Bancos de Dados Móveis.** 2004.

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