3
Trabalhos Relacionados

Neste capítulo serão apresentados os trabalhos mais relevantes na área de banco de dados paralelo e processamento de consulta adaptativa que contribuíram para o desenvolvimento desta dissertação.

Os trabalhos referentes a banco de dados paralelos são em sua maioria da década de 80 e 90, época em que este assunto foi tema ativo de pesquisa. Durante a década de 80 foram desenvolvidos vários protótipos de banco de dados paralelos como Teradata (F.CARINO & P.KOSTAMAA, 1992), Bubba (HARAN BORAL, WILLIAM ALEXANDER et al., 1990), GAMMA (D.J.DEWITT, SHAHRAM GHANDEHARIZADEH et al., 1990), XPRS (M.STONEBRAKER, R.KATZ et al., 1988), dois quais resultaram várias técnicas de particionamento dos dados e processamento de consulta (i.e. modelo de execução baseado em iteradores). Já na década de 90 as pesquisas foram direcionadas para o processo de otimização e tratamento do problema de *skew*.

As pesquisas em processamento de consulta adaptativo são mais recentes e tiveram início no final da década de 90. Inicialmente os trabalhos tratavam da utilização de informações coletadas dinamicamente no processo de otimização. Mais recentemente, com o aparecimento de processamento de consultas contínuas foram desenvolvidas técnicas que permitem adaptação durante a execução.

A seguir são apresentados os trabalhos na área de banco de dados paralelos e em seguida os de processamento de consulta adaptativo. Para cada um destes tópicos é proposta uma classificação e, em seguida, uma descrição de cada trabalho. No final de cada seção uma tabela sumariza as principais características de cada abordagem.

3.1. Banco de dados Paralelos

No contexto de banco de dados paralelos, os trabalhos existentes podem ser classificados em três grandes grupos: desenho, otimização e técnicas de

processamento paralelo. Neste trabalho, foca-se o processo de otimização de consultas paralelas, em especial àqueles com paralelismo intra-operador.

3.1.1. Otimização de Consultas Paralelas

O processo de otimização em um ambiente paralelo, conforme descrito na seção 2.1.3.4, pode ser realizado em uma ou duas fases. O espaço de busca para a otimização de um plano paralelo é muito maior que o espaço de busca de um plano centralizado em função dos vários formatos de árvores a serem considerados e da alocação de operadores do plano nas máquinas alocadas. Dada esta complexidade, a maioria dos trabalhos encontrados realiza o processo em duas fases. A primeira utiliza uma abordagem tradicional para produção de um plano de execução de consultas para um ambiente centralizado. Em seguida, a segunda fase estende o plano inicial com a análise de paralelismo e, caso assim decida, introduz operadores para gerência da execução paralela. No entanto, em ambas as abordagens existe a necessidade do algoritmo de escalonamento que definirá quantas máquinas serão utilizadas e em que máquina cada fragmento do plano de execução deverá ser executado. Nesta seção apresentam-se alguns algoritmos encontrados na literatura classificados segundo o tipo de paralelismo considerado no processo de otimização e a arquitetura utilizada.

A primeira decisão a ser tomada durante o desenvolvimento de um algoritmo de escalonamento para um sistema de banco de dados é a identificação do número de consultas que podem ser executadas concorrentemente no sistema, ou seja, se o sistema deve ou não suportar o paralelismo inter-consulta. A utilização deste tipo de paralelismo normalmente implica em algoritmos mais complexos, pois requerem a utilização de informações dinâmicas do sistema (i.e. carga de trabalho dos nós e disponibilidade de memória).

Um segundo passo é a decisão de qual o tipo de paralelismo será suportado durante a execução de uma consulta e, neste caso, podem ser utilizados o paralelismo intra-operador, independente e *pipeline*. O paralelismo intra-operador pode ser utilizado para reduzir o tempo de execução de um operador e uma questão muito importante em sua utilização é a definição do número de processadores a serem alocados.

(A.N.WILSCHUT, J.FLOKSTRA et al., 1992) apresenta um modelo para determinar o nível ideal de paralelismo para uma operação considerando um banco de dados em memória sem a utilização de paralelismo inter-consulta. De acordo com experimentos realizados o nível de paralelismo é limitado pelo tempo de inicialização dos nós a serem utilizados, o custo de uma operação e o tamanho da entrada. Quanto maiores forem a entrada e o custo de realização de uma operação, maior será o número de nós ideal a serem alocados.

Em (ERHARD RAHM, 1996) são apresentados algoritmos de escalonamento que utilizam informações dinâmicas do sistema (i.e. memória disponível e utilização de CPU) para um ambiente com paralelismo inter-consulta. No entanto, estas soluções não fazem uso de paralelismo vertical durante a execução das consultas, pois consideram a utilização de um operador que materializa os resultados intermediários.

(MANISH MEHTA & DAVID J.DEWITT, 1995) (ERHARD RAHM & ROBERT MAREK, 1995) realiza experimentos em um sistema com diferentes cargas de trabalho (1, 15, 35, e 55 consultas por segundo) e conclui que quanto maior a utilização do sistema menor o nível de paralelismo a ser utilizado. A partir desta observação é desenvolvido um algoritmo para determinar o nível de paralelismo que é baseado no número de nós ideal em um ambiente mono-usuário. Dependendo da utilização do sistema o número de nós escalonados é reduzido. A alocação é definida com base na utilização dos processadores que é atualizada constantemente. Quando um nó é selecionado sua utilização é incrementada para evitar que ele seja escalonado novamente. (ERHARD RAHM, 1996) estende este algoritmo considerando que disco e memória também podem representar um gargalo para a execução.

(MANISH MEHTA & DAVID J.DEWITT, 1995) critica a metodologia utilizada em (ERHARD RAHM & ROBERT MAREK, 1995), pois segundo o autor as consultas utilizadas seriam pequenas e o nível de paralelismo utilizado não deveria estar associado ao ambiente mono-usuário, uma vez que isso pode ser difícil de prever. Em contraproposta é apresentado um algoritmo baseado nas taxas de entrada e saída dos operadores. O nível ideal seria aquele em que a soma da capacidade dos produtores fosse igual à de seus consumidores, desta forma não haveria disputa por dados. Para realizar a alocação, foram comparados vários

algoritmos e o que apresentou melhor resultado foi o algoritmo baseado na utilização da CPU proposto anteriormente.

(ANASTASIOS GOUNARIS, RIZOS SAKELLARIOU et al., 2006) propõe um algoritmo de escalonamento para a execução de consultas paralelas no ambiente de grade, que tem como objetivo definir uma alocação para a execução paralela de operadores com elevado custo de avaliação. Para cada operador a ser paralelizado o algoritmo tenta incluir um novo nó, se a diferença entre o custo atual e o anterior for maior que certo limiar então este nó será utilizado. Um modelo de custo independente é utilizado para estimar o custo de avaliação da consulta e de cada operador físico do plano de execução.

(MING-SYAN CHEN, PHILIP S.YU et al., 1996) apresenta uma proposta para a otimização de consultas com múltiplas junções que considera o uso de paralelismo independente e intra-operador em uma arquitetura de memória compartilhada. Uma vez que o paralelismo do tipo *pipeline* não é utilizado, existe uma forte dependência entre a execução de um operador de junção e seus produtores, pois aquele não pode iniciar antes que todos seus filhos na árvore de execução tenham terminado. Levando em consideração este problema é proposto um algoritmo de escalonamento *top-down* que tenta equiparar o tempo de execução dos produtores de forma a minimizar o tempo de execução das sub-árvores e o tempo em que um conjunto de processadores poderia ficar ocioso.

Em (HUI-I HSIAO, 1994) é utilizado o algoritmo de escalonamento proposto em (MING-SYAN CHEN, PHILIP S.YU et al., 1996) para a otimização de um pipeline de junções baseadas em hash com implementação de duas fases síncronas. Neste caso, o problema é que o *pipeline* só pode começar a executar quando todos os operadores tiverem a fase de *build* finalizada.

Em (WAQAR HASAN & RAJEEV MOTWANI, 1995) é realizado um estudo sobre a influência da comunicação entre operadores sobre o processo de otimização de consultas que utilizam o paralelismo do tipo *pipeline*. Diferente do proposto em volcano (G.GRAEFE, 1990), onde o custo de comunicação é modelado por um operador, neste trabalho o custo de comunicação é representado como restrição de tempo nas arestas que ligam os operadores em uma árvore de execução. A partir desta representação, uma heurística é utilizada para eliminar os casos em que o custo de comunicação é maior que o custo de processamento.

A tabela 1 sumariza os trabalhos apresentados nesta seção.

Trabalho	Dinâmico	Informações Monitoradas	Intra-Operador	Inter-Operador	
				Pipeline	Independente
DeWitt	Sim	CPU, memória	Sim	Não	Não
Rahm	Sim	CPU, memória, disco	Sim	Não	Não
Gounaris	Não	-	Sim	S/N	Não
Anita					
MS Chen	Não	-	Sim	Não	Sim
HSiao	Não	-		Sim	
Hasan	Não	-	Não	Sim	Não

Tabela 1. Sumário dos trabalhos paralelos

3.2. Consultas Adaptativas

Nesta seção serão apresentados os trabalhos relacionados à área de adaptatividade. Seguindo a organização proposta em (ANASTASIOS GOUNARIS, NORMAN W.PATON et al., 2002) classificaremos os trabalhos quanto a sua meta, o nível das alterações realizadas e as informações utilizadas no processo de adaptação. A figura 4 apresenta um resumo dos diversos aspectos da classificação utilizada:

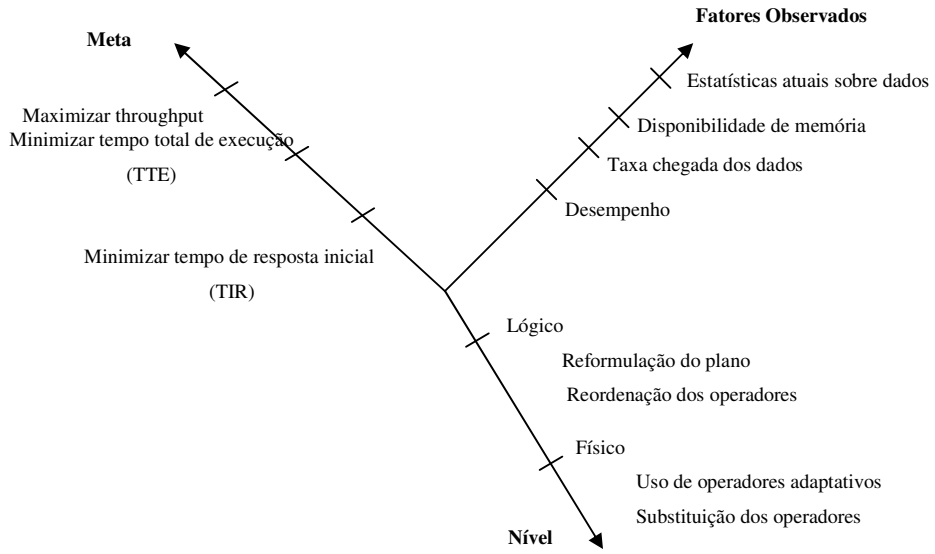

Figura 4. Eixos da adaptatividade

As técnicas de adaptatividade devem ser utilizadas levando em consideração o objetivo do sistema que, geralmente visa minimizar o tempo total de execução (MTTE), o tempo de resposta inicial (MTRI) ou a vazão (*throughput*) de cada consulta (MT).

As técnicas de adaptação podem variar significativamente de acordo com o fator ao qual tentam reagir. Os trabalhos analisados, em sua maioria, apresentam técnicas para se adaptar à: flutuações em memória, desempenho, taxa de chegada dos dados, existência de estatísticas sobre os dados.

O nível de adaptação está relacionado aos componentes do sistema alterados pelas ações corretivas realizadas. Estas últimas podem ser classificadas em lógica e física. As alterações lógicas estão relacionadas à organização do plano de execução e podem envolver desde a reformulação de todo o plano de execução (alterando sua forma), ou a simples re-ordenação dos operadores. Já as alterações no nível físico estão relacionadas aos operadores do plano de execução. Elas podem envolver a troca dos operadores ou a utilização de operadores adaptativos, que conseguem mudar seu comportamento de acordo com as características do ambiente. A seguir são apresentados os trabalhos mais relevantes.

Kabra e DeWitt (NAVIN KABRA & DAVID J.DEWITT, 1998) propõem um modelo dinâmico para se detectar a utilização de planos não ótimos criados a partir de informações estatísticas imprecisas. O plano de execução utilizado é enriquecido com as informações estatísticas utilizadas no processo de otimização e operadores especiais são inseridos em pontos estratégicos do plano de forma a capturar informações dinâmicas sobre os resultados intermediários produzidos. Durante a execução, as informações coletadas pelos operadores podem ser comparadas com as informações utilizadas inicialmente e, se alguma diferença que possa levar a um custo maior que o esperado for observado os resultados produzidos até então são materializados e o restante do plano é otimizado. Um esquema de classificação das informações que podem ser capturas é utilizado para garantir que seu tempo de obtenção não incorra em um *overhead* substancial no tempo de execução da consulta.

Tukwila (ZACHARY G.IVES, ALON Y.LEVY et al., 2000) é um sistema de execução de consulta adaptativo de alto desempenho para a integração de fontes de dados autônomas e heterogêneas. Seu processo de adaptação é realizado

a partir de eventos lançados pela máquina de execução que podem levar desde uma simples substituição do operador a re-otimização do plano. O processo de otimização utilizado é bem flexível e realiza a intercalação das fases de otimização e execução. De acordo com as informações disponíveis sobre as relações envolvidas na consulta, o otimizador pode optar por definir o plano de execução dinamicamente ou adaptá-lo após a execução em *pipeline* de fragmentos do plano. A adaptação do plano de execução é realizada conforme (NAVIN KABRA & DAVID J.DEWITT, 1998) e a principal diferença é que os pontos de adaptação podem ser definidos dinamicamente. Para reduzir o impacto causado pela variação das taxas de chegada dos dados o sistema faz uso dos operadores: *Colector* e *Double Pipeline Hash-Join* (DPHS). O operador *Colector* é responsável por realizar a união de dados advindos de diversas fontes e, se houver sobreposição, o operador pode definir qual a melhor fonte a ser utilizada. Já o DPHJ realiza as fases de construção e investigação simultaneamente e, desta forma, se uma das fontes de dados não responder o operador não fica bloqueado.

Em (TOLGA URHAN, MICHAEL J.FRANKLIN et al., 1998) é apresentada uma proposta para minimizar o retardo inserido na espera pela primeira tupla. Este problema é comum na integração de dados em que pode haver dificuldade de conexão, sobrecarga das fontes de dados, ou porque alguma tarefa deve ser realizada antes que os dados sejam enviados. Durante uma primeira fase o sistema tenta escalonar outros operadores e materializa seu resultado para processamento futuro, quando isso não for mais possível, o sistema entra em uma segunda fase na qual o plano pode ser reformulado e novos operadores podem ser adicionados. Uma vez que a materialização dos resultados intermediários e a alteração do plano podem resultar em trabalho extra a ser realizado, o sistema utiliza um modelo baseado na relação entre o custo e o benefício de cada alteração para guiar o processo de adaptação.

O projeto *Telegraph* combina as técnicas utilizadas em rivers (REMZI H.ARPACI-DUSSEAU, ERIC ANDERSON et al., 1999) e eddies (RON AVNUR & JOSEPH M.HELLERSTEIN, 2000) para formar um *dataflow* adaptativo. Um ambiente de *dataflow* movimenta uma grande massa de dados entre operadores que podem estar sendo executados em diferentes máquinas, e pode ser visto como um sistema de processamento de consulta.

Eddies é um mecanismo de processamento de consulta que re-ordena constantemente os operadores de um plano para se adaptar às variações que podem ocorrer nos dados durante a execução. Inicialmente são verificadas as restrições de re-ordenabilidade existentes e é criado um plano de execução com as informações disponíveis. Neste plano, os operadores são interligados por um operador de controle n-ário denominado *Eddy,* que realiza a leitura dos dados das fontes e determina um roteamento para cada tupla segundo as informações observadas (i.e. seletividade de cada operador). Em especial, este mecanismo requer operadores de junção que apresentem momentos de adaptação constantes nos quais a ordem dos operadores pode ser alterada com pouca ou nenhuma alteração em seu estado.

River é um ambiente de programação em *dataflow* que tem como objetivo manter o desempenho do sistema mesmo na presença de heterogeneidade dos componentes de *hardware* utilizados. Suas principais contribuições são as técnicas de *Graduated Declustering* (GD) e *Distributed Queue* (DQ) que podem ser utilizadas no balanceamento de carga na utilização de paralelismo intra-operador. O mecanismo DQ é responsável por balancear a carga entre consumidores com diferentes taxas de consumo e, é baseada na inserção de filas no produtor e nos consumidores. A distribuição de dados é realizada pelo produtor segundo uma política que se beneficia das vantagens das estratégias *put* e *get* e, pode ser descrita da seguinte maneira: Sempre que houver dados a serem distribuídos, o produtor verifica o número de mensagens enviadas e para os quais não se recebeu uma confirmação, se este valor for maior que um certo limiar o produtor aguarda até que créditos estejam disponíveis. Quando houver créditos disponíveis o produtor obtém um consumidor de forma aleatória e realiza um teste idêntico ao apresentado anteriormente. Se houver créditos, o dado é enviado senão, um novo consumidor é sorteado. Sempre que um consumidor começa a processar uma mensagem do produtor uma confirmação é enviada e indica que o consumidor está realizando progresso. Já o mecanismo GD proporciona a utilização máxima de toda a banda de comunicação disponível.

CACQ (SAMUEL MADDEN, MEHUL A.SHAH et al., 2002) estende o projeto *Telegraph* para execução de consultas contínuas que são realizadas sobre dados captados após o seu registro. O diferencial deste trabalho é apresentar uma maneira mais efetiva para o compartilhamento de computação e armazenamento

entre consultas simultâneas. Para alcançar tal objetivo as tuplas de uma relação são compartilhadas entre as várias consultas e um *Eddy* é responsável por roteá-las entre os diversos operadores, um mecanismo de controle é inserido em cada tupla para identificar a quais consultas uma tupla ainda participa. Em adição a este mecanismo são utilizados *SteMs* (*State Modules*) (VIJAYSHANKAR RAMAN, AMOL DESHPANDE et al., 2003) e filtros agrupados. Como veremos adiante os *SteMs* realizam a divisão dos operadores de junção em módulos de consulta mais refinados que não guardam informações específicas de uma junção, mas sobre uma relação. Isso permite o compartilhamento das operações de junções entre as consultas. Já um filtro agrupado é um operador capaz de realizar eficientemente várias operações de seleção sobre um atributo de uma relação e identifica para quais consultas uma dada tupla deve prosseguir.

A utilização de *SteM* tem o objetivo de oferecer maior flexibilidade de adaptação à arquitetura do *Eddies*, de forma que o método de acesso, o algoritmo de junção e o formato do plano também possam ser alterados dinamicamente.

A tabela 2 sumariza a característica dos trabalhos apresentados nesta seção de acordo com o nível das ações corretivas, os fatores observadas durante a execução, e o objetivo e o ambiente para as quais se aplicam.

Trabalho	**Nível Correção**		**Fatores**	**Objetivo**	**Ambiente**
	Lógico	**Físico**			
Mid-Query Re-optimization	Reform.	Subst/Adição de op.	Novas Estatísticas	TTE	Mono
Query Scrambling	Reform.	Subst/Adição de op.	Tx chegada dados	TTE	Dist.
Tukwila	Reform.	Op. adap.	Qq alteração no ambiente.	TTE TRI	Dist.
Telegraph	Reord. op.	Op. adap.	Qq alteração no ambiente.	TTE	Dist.
Eddies	Reord. op.	-	Qq alteração no ambiente	TTE.	Mono
River	-	Op. adap.	Desempenho	TTE	Paralelo
CACQ	Reform.	Op. Adap.	Qq alteração no ambiente	TTE	Dist.

STem	Reform.	Op. Adap.	Qq alteração no ambiente	TTE	Mono

Tabela 2. Características dos trabalhos adaptativos

3.3. Síntese do Capítulo

Neste capítulo foram apresentados os trabalhos mais relevantes na área de otimização de consultas paralelas e processamento de consulta adaptativo que influenciaram no desenvolvimento deste trabalho. Os trabalhos analisados na área de otimização de consultas paralelas, em especial (MANISH MEHTA & DAVID J.DEWITT, 1995), foram de suma importância para a compreensão dos problemas associados à definição do nível de paralelismo ideal, e guiaram a construção do algoritmo de escalonamento G2N. Já os trabalhos na área de processamento de consulta adaptativo, em especial (RON AVNUR & JOSEPH M.HELLERSTEIN, 2000) (REMZI H.ARPACI-DUSSEAU, ERIC ANDERSON et al., 1999), serviram de base para o desenvolvimento da arquitetura de execução proposta neste trabalho.