O Efeito da Heterogeneidade no Desempenho de Sistemas Paralelos de Banco de Dados *

Luiz Henrique Gomes Virgílio Almeida Alberto Henrique Frade Laender

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702
30161-970 - Belo Horizonte - MG

Resumo

A enorme capacidade computacional provida pelas arquiteturas paralelas juntamente com a facilidade de se paralelizar as operações sobre bancos de dados relacionais incentivaram o desenvolvimento de sistemas paralelos de banco de dados. Além disso, o uso de microprocessadores de alto desempenho e baixo custo para a construção de máquinas paralelas torna esses sistemas muito competitivos quando comparados aos *mainframes*, até então a principal plataforma para grandes aplicações de banco de dados. No entanto, os sistemas paralelos apresentam alguns problemas que inibem o aproveitamento real de sua grande capacidade de processamento. Entre eles, encontra-se o *data skew* que se refere à dificuldade de se dividir igualmente o processamento de uma consulta entre os processadores do sistema, devido a características inerentes aos dados. O objetivo deste artigo é apresentar uma proposta de arquitetura, denominada heterogênea, e analisar, através de simulação, o seu desempenho na presença do problema de *data skew*.

Abstract

The enormous computational power provided by parallel architectures and the possibility of parallelizing database operations have motivated the development of parallel database systems. Successful parallel database systems built from conventional microprocessors, memories, and disks are becoming more attractive than mainframes. However, there are generic barriers to linear speedup. For example, partitioned data is the key to parallel execution. The problem with data partitioning is that it risks data skew, where all the data is placed in one partition, and execution skew in which most of the the execution occurs in one processor. This paper addresses the problem of data and execution skew and proposes an architecture-based solution to the problem. The paper shows, through a simulation model, that a heterogeneous parallel architecture composed of processors of different speed can minimize the skew problem.

*Trabalho parcialmente financiado pelo CNPq e pela FAPEMIG (contrato TEC-1113/90)

1 Introdução

A enorme capacidade computacional provida pelas arquiteturas paralelas juntamente com a facilidade de se paralelizar as operações sobre bancos de dados relacionais incentivaram o desenvolvimento de sistemas paralelos de banco de dados. Além disso, o uso de microprocessadores de alto desempenho e baixo custo para a construção de máquinas paralelas torna esses sistemas muito competitivos quando comparados aos *mainframes*, até então a principal plataforma para grandes aplicações de banco de dados [4, 12, 5].

Podem ser naturalmente identificados dois tipos de paralelismo nas aplicações de banco de dados: *entre-transações*, que diz respeito a várias transações em execução simultânea, e *intra-transações*, que se refere ao particionamento de uma transação em tarefas para execução em paralelo. No entanto, os sistemas paralelos de banco de dados apresentam alguns problemas que inibem o aproveitamento real de sua grande capacidade de processamento. Entre eles, encontra-se o *data skew* que se refere à dificuldade de se dividir igualmente o processamento de uma consulta entre os processadores do sistema, devido a características inerentes aos dados [9].

O objetivo deste artigo é apresentar uma proposta de arquitetura, denominada heterogênea, e analisar, através de simulação, o seu desempenho na presença do problema de *data skew*. O artigo está organizado da seguinte forma. A próxima seção apresenta as características gerais dos sistemas paralelos de banco de dados, dando ênfase às arquiteturas e algoritmos paralelos de junção. A terceira seção discute a natureza do problema de *data skew*. A quarta seção descreve o modelo global de simulação, detalhando a carga de trabalho e os módulos do simulador. Na quinta seção, são apresentados os resultados numéricos, comparando as arquiteturas homogênea e heterogênea. Finalmente, na última seção, estão as conclusões junto com propostas para trabalhos futuros.

2 Sistemas Paralelos de Banco de Dados

Sistemas paralelos de banco de dados começam a se tornar uma alternativa viável para os *mainframes* tradicionais no caso de grandes aplicações de banco de dados. Esses sistemas fazem uso de arquiteturas paralelas construídas a partir de processadores, memórias e discos convencionais, possibilitando grande capacidade de expansão (*scaleup*) e ganho de desempenho (*speedup*) [4, 5]. Uma outra razão para o grande interesse despertado pelos sistemas paralelos de banco de dados se deve ao modelo de dados relacional que hoje é adotado pela grande maioria dos sistemas disponíveis comercialmente. Consultas relacionais são ideais para o processamento paralelo, uma vez que consistem em operações bem definidas que são aplicadas a um conjunto uniforme de dados. Cada operação relacional produz como resultado sempre uma nova relação, de modo que toda operação pode ser decomposta em sub-operações que são executadas em paralelo [5].

Nesta seção, apresentamos uma classificação das principais arquiteturas paralelas, como também uma visão geral dos principais algoritmos paralelos para a execução da operação de junção, que é a operação predominante nas consultas a bancos de dados relacionais.

2.1 Classificação das Arquiteturas Paralelas

As arquiteturas paralelas que utilizam processadores convencionais podem ser classificadas de acordo com a seguinte taxonomia [5, 14]:

- *shared memory* ou *shared everything* - Este tipo de arquitetura consiste de processadores, módulos de memória e discos interligados através de uma rede de interconexão, de modo que todos os processadores compartilham diretamente uma memória global e todos os discos. O principal problema deste tipo de arquitetura é a contenção de recursos, causada devido ao grande compartilhamento existente.
- *shared disk* - Neste tipo de arquitetura, cada processador tem seu próprio módulo de memória, porém compartilha os discos com todos os outros processadores. Assim os problemas de contenção de recursos que ocorrem neste tipo de arquitetura são análogos aos existentes nas arquiteturas do tipo *shared everything.*
- *shared nothing* - Neste tipo de arquitetura, os processadores não compartilham nenhum recurso. Cada processador possui seus próprios módulos de memória e discos, de modo que a comunicação entre eles é feita por meio de mensagens através da rede de interconexão. A principal vantagem deste tipo de arquitetura é que a introdução de novos processadores na rede nunca interfere negativamente na performance do sistema.

Essas arquiteturas possibilitam a exploração de dois tipos de paralelismo nas aplicações de banco de dados:

- *Paralelismo entre-transações* - Consiste em se executar simultaneamente várias transações. Nos *mainframes* convencionais, este tipo de paralelismo é alcançado usando-se processadores de E/S e/ou controladores de rede, de modo que enquanto uma transação executa uma operação de E/S, uma outra utiliza o processador, possibilitando que mais de uma transação seja executada ao mesmo tempo. Nas máquinas paralelas, esse paralelismo é alcançado alocando-se processadores distintos para atender simultaneamente várias transações, além de usar processadores de E/S, quando disponíveis.
- *Paralelismo intra-transações* - Consiste em se particionar uma transação em tarefas independentes que são executadas ao mesmo tempo em processadores distindos, de forma que o tempo de execução da transação é o tempo de execução da maior tarefa. Este tipo de paralelismo é pouco explorado nos *mainframes* convencionais, porém é muito utilizado nas máquinas paralelas.

Dentre as arquiteturas citadas, a mais adequada para aplicações de banco de dados é a do tipo *shared nothing* [4, 9]. Arquiteturas *shared nothing* possuem uma capacidade de expansão e atingem um ganho de desempenho quase-linear na execução de operações relacionais e no processamento de transações *on line* [9]. Atualmente, diversos sistemas em desenvolvimento adotam uma arquitetura do tipo *shared nothing* [3, 4, 5, 6].

2.2 Algoritmos Paralelos de Junção

Um banco de dados relacional consiste de um conjunto de *relações*, que podem ser vistas como tabelas, nas quais cada linha (*tupla*) possui o mesmo conjunto de *atributos*. O modelo de dados relacional define um conjunto de operações que permitem combinar e comparar duas ou mais relações. Dentre essas operações, a junção é uma das mais freqüentes e requer grande demanda computacional. A operação de junção combina duas relações R e S dando origem a uma terceira relação na qual cada tupla é formada pela concatenação de uma tupla de R e uma tupla de S que satisfazem uma determinada condição θ definida sobre os atributos de R e S. Neste trabalho, estamos interessados apenas em um caso especial da operação de junção, denominada *equi-junção*, cuja condição θ é uma igualdade.

Diversos algoritmos de junção encontram-se descritos na literatura [2, 13, 15, 11]. No caso de sistemas paralelos de banco de dados, análises de desempenho indicam que os algoritmos mais importantes são o de junção por espalhamento (*hash join algorithm*) e o de junção por intercalação (*sort merge join algorithm*) [13, 9, 16, 17]. A seguir, descrevemos apenas o algoritmo de junção por espalhamento que é usado neste trabalho.

Seja uma operação de (equi-)junção sobre duas relações R e S. O algoritmo paralelo de junção por espalhamento consiste dos seguintes passos:

1. Uma função *hash* é aplicada ao atributo de junção de cada tupla de R e S de forma a distribuí-las entre os diversos processadores, de modo que tuplas em processadores distintos não satisfaçam a condição de junção;
2. Cada processador aplica uma função *hash* às tuplas das partes das relações R e S que ele detém, agrupando-as de forma que tuplas de R em um mesmo grupo satisfaçam a condição de junção com relação a tuplas de apenas um grupo correspondente de S;
3. cada processador executa a junção dos grupos correspondentes de tuplas de R e S, devolvendo os resultados ao coordenador da transação que combina os resultados parciais e produz o resultado final da junção.

O primeiro passo do algoritimo faz o espalhamento das tuplas para que o processamento possa ser dividido em tarefas independentes em processadores distintos. A única restrição quanto à função *hash* usada é que seu contra-domínio seja o conjunto dos processadores com dispositivo de armazenamento secundário. As tuplas são remetidas de um processador a outro por mensagens através da rede. O segundo passo do algoritmo objetiva fazer com que a junção seja feita em partes que possam ser totalmente carregadas na memória dos processadores. Logo o número de partes em que se deve dividir as tuplas das relações depende da memória disponível nos processadores. Finalmente, o último passo do algoritmo consiste na execução da junção propriamente dita. Ao término da junção de cada grupo de tuplas de R com o grupo correspondente de tuplas de S, os resultados são remetidos via rede ao coordenador da transação, que então produz o resultado final da junção.

Uma condição básica para que o algoritmo de junção por espalhamento funcione satisfatoriamente, é que a distribuição das tuplas entre os vários processadores seja feita uniformemente, de

modo que a carga de cada processador seja aproximadamente a mesma. Entretanto, essa distribuição uniforme nem sempre é possível. Na maioria dos bancos de dados reais, certos valores de determinados atributos ocorrem com mais freqüência do que outros, resultando em uma distribuição desbalanceada. Este fenômeno, denominado *data skew* [9], tem um impacto significativo no tempo de execução de uma junção e será discutido com mais detalhe a seguir.

3 Data Skew

Consideremos um banco de dados de uma empresa contendo as seguintes duas relações: Departamento, com os atributos NumDept, NomeDept e Ramal, e Empregado, com os atributos NumEmp, NomeEmp, Salario e NumDept. O atributo NumDept é a chave da relação Departamento, possuindo, portanto, valores únicos. Entretanto, este mesmo atributo na relação Empregado pode não assumir valores únicos, uma vez que em um departamento trabalham geralmente vários empregados. Como em qualquer empresa nem todos os departamentos possuem o mesmo número de empregados, é bastante provável que determinados valores do atributo NumDept apareçam com muito mais freqüência na relação Empregado do que outros, caracterizando o *data skew*.

Assim, ao se executar a junção das relações Departamento e Empregado sobre o atributo NumDept, após a aplicação da função *hash* as tuplas com os mesmos valores serão enviadas para os mesmos processadores, acarretando uma carga desbalanceada desses processadores em função do *data skew* que limitará as vantagens do processamento paralelo [9]. Um aspecto importante do *data skew* é que ele é um problema inerente aos dados, de modo que o desbalanceamento de carga entre os processadores ocorre mesmo que uma função *hash* ideal seja utilizada, ou seja, a função *hash* não é uma solução para o problema de *data skew* [9].

Várias alternativas têm sido propostas para a solução do problema de *data skew* em sistemas paralelos de banco de dados. Em [8], procura-se diminuir o efeito do *data skew* através de uma arquitetura híbrida na qual arquiteturas *shared everything* são interconectadas por meio de uma rede de comunicação formando uma estrutura *shared nothing*. Nessa abordagem, o processamento fica restrito aos componentes com arquitetura *shared everything* de modo a minimizar o efeito do *data skew*. Em [16] e [17] são propostas novas versões para os algoritmos paralelos de junção por espalhamento e por intercalação, nas quais é introduzida uma fase de escalonamento que, utilizando um algoritmo de otimização heurística, procura balancear a carga entre os processadores.

Neste trabalho, apresentamos uma proposta para minimização do problema de *data skew* baseada no uso de processadores heterogêneos. De acordo com esta proposta, um processador heterogêneo, com maior capacidade de processamento, é alocado para receber os grupos de tuplas com maior concentração de dados. Assim, o tempo de processamento para a execução da junção envolvendo essas tuplas é reduzido e aproxima-se dos tempos de execução das junções dos grupos de tuplas que foram espalhados pelos demais processadores, resultando na redução do tempo global de execução da junção. A substituição de vários processadores idênticos por um de maior capacidade é feita de maneira a preservar o custo total das configurações homogêneas e heterogêneas [10].

Nas próximas seções apresentamos resultados numéricos indicando que o uso de um processador heterogêneo minimiza o problema causado pelo efeito *data skew*.

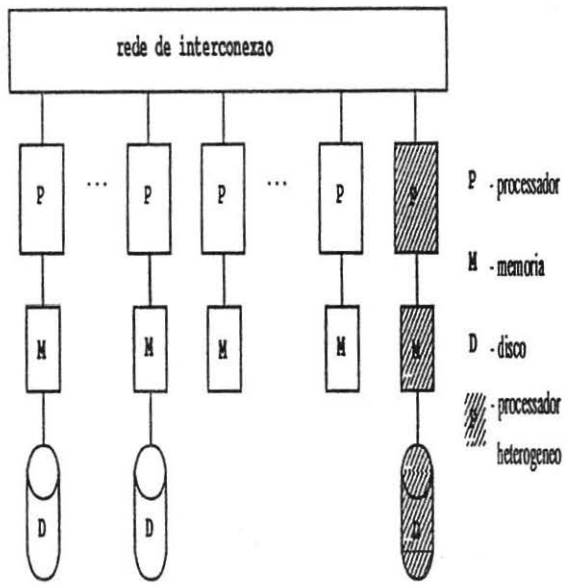

Figura 1: Arquitetura Paralela Heterogênea

4 Modelo Global do Sistema

Esta seção descreve o modelo global do sistema, que consiste de três componentes: a arquitetura paralela, a carga de trabalho composta das transações de consulta ao banco de dados e o simulador usado para comparar o desempenho das arquiteturas existentes com o da arquitetura proposta.

4.1 Arquitetura

O modelo adotado refere-se a um sistema do tipo *shared nothing* onde vários processadores completamente independentes são interligados através de uma rede de interconexão. No modelo, existem duas classes de processadores: interface e execução. Os processadores de interface não possuem dispositivos de armazenamento secundário e têm como função principal a interface com o usuário. Já os s processadores de execução possuem sistema de armazenamento secundário, que pode ser do tipo *array of disks* ou apenas discos individuais. Estes são os processadores que efetivamente executam as transações. Quando todos os n processadores de execução são idênticos, no que se refere à velocidade computacional, diz-se que a arquitetura é homogênea. A proposta do artigo consiste em substituir um subconjunto α dos processadores idênticos por um de maior velocidade computacional, de tal forma que o custo total da configuração permaneça inalterado. Essa arquitetura composta de $n - \alpha$ processadores idênticos e um mais rápido que os demais é denominada heterogênea. A figura 1 apresenta um diagrama dessa arquitetura.

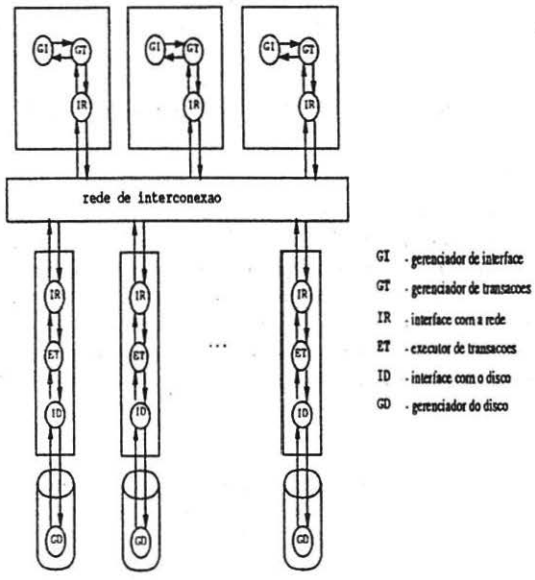

Figura 2: Modelo do Simulador

4.2 Modelo do Simulador

O simulador construído para avaliar o efeito da heterogeneidade em arquiteturas *shared nothing* compõe-se de vários módulos, conforme mostra a figura 2. Suas funções estão descritas a seguir.

- Gerenciador de Interface com Usuário

  Executa a interface com o usuário recebendo novas consultas aos bancos de dados e devolvendo os resultados das transações já processadas. Ao receber uma transação do usuário este módulo imediatamente a coloca na fila do *gerenciador de transações* e ao receber uma resposta de transação do *gerenciador de transações*, o módulo retorna o resultado para o usuário. Este módulo é executado no processador coordenador da transação.

- Gerenciador de Transações

  Este módulo faz a coordenação das transações vindas do *gerenciador de interface* e se encarrega de repassar os resultados parciais das transações vindas dos executores, controlando o fim de cada transação. Ao receber uma transação vinda do *gerenciador de interface* este módulo prepara a requisição de emissão de mensagens aos executores avisando o início da transação. Para controlar o fim da transação, abre-se uma entrada no conjunto de transações em execução.

- Interface com a Rede

Este é o módulo responsável por controlar as mensagens a serem enviadas pela rede ou a serem entregues aos processos destinos. Ao receber uma mensagem a ser enviada para a rede, ele prepara a mensagem com os controles e endereços conhecidos pela rede e requisita seu envio. Quando recebe uma mensagem vinda da rede para um módulo executando no processador em que ele está, ele retira as informações que não interessam ao módulo destino e a repassa a este.

- Executor da Transação

  Este módulo cuida da execução da transação. Ao receber uma mensagem de início de transação ele faz o espalhamento inicial das tuplas das relações que participam da transação e que ele detém. Usando uma função *hash* pré-estabelecida, o módulo se prepara para receber tuplas que compõem a parte da transação que ele executará. Ao receber essas tuplas a função *hash* de divisão em grupos é aplicada ao atributo de junção e os grupos são escritos nos discos do processador onde ele está executando. Ao reconhecer o fim das tuplas vindas dos outros processadores, ele dispara a leitura dos grupos de tuplas e inicia a execução da junção, rementendo os resultados ao coordenador da transação.

- Interface com o Disco

  Este módulo gerencia os *buffers* de disco, de forma que ao receber uma requisição de dados do executor verifica se o dado está disponível na memória e, caso contrário, a requisição é repassada ao disco.

- Gerenciador de Disco

  Este módulo recebe as requisições de dados vindas da interface com o disco e enfileira para que o disco as execute em ordem de chegada. Ao receber dados dos discos os repassa ao módulo de *interface com o disco*.

- Gerenciador da Rede

  Este módulo gerencia a emissão de mensagens pela rede. Ele recebe as mensagens e as encaminha ao processador de comunicação do nó destino.

4.3 Modelo da Carga de Trabalho

A carga de trabalho do sistema compõe-se de transações que chegam ao sistema com uma taxa λ. Cada transação é composta de uma equi-junção entre duas relações R e S. No espalhamento inicial supõe-se uma taxa de *skew* γ, onde γ é a fração das tuplas enviadas para um processador executor da rede. O restante é dividido igualmente entre os outros processadores executores da rede.

5 Parâmetros do Modelo

A seguir são descritos os parâmetros que caracterizam cada um dos três modelos descritos acima, junto com os valores usados em nossas simulações.

- Parâmetros Relativos à Arquitetura

- Vel_Proc

  Velocidade dos Processadores, indica o número de instruções que cada processador pode executar por segundo. Os valores usados foram: processadores homogêneos 8 MIPS e processador heterogêneo quando existe 32 MIPS.

- Num_Coord

  Número de processadores coordenadores. O valor usado foi 4.

- Num_Exec

  Número de processadores executores de transações. O valor usado para este parâmetro variou entre 20 e 76 para o caso em que se usou um processador heterogêneo e entre 23 e 79 quando não se usou processador heterogêneo. O número de processadores foi incrementado sempre de 8 em 8.

  O caso em que se usa processador heterogêneo é derivado do caso em que não se usa. Aplicando uma função de custo por MIPS constante [10], trocou-se 4 processadores homogêneos por um processador 4 vezes mais rápido.

- Parâmetros Relativos à Carga de Trabalho

  - Taxa_skew

    A taxa de *skew* usada foi de 10%, isto é, 10% das tuplas sempre são enviadas para apenas um processador da rede.

  - Taxa_Selet

    Este parâmetro nos diz qual a percentagem das tuplas de cada relação que efetivamente participam da junção. O valor usado foi 30%.

  - Tm_Chegadas

    O tempo médio entre chegadas nos indica com que frequência transações chegam ao sistema. O valor usado foi 16 segundos, ou seja, a cada 16 segundos em média chega uma transação ao sistema.

  - Tam_Tupla

    Tamanho de cada tupla da relação. Foi usado o valor 128 bytes.

  - Num_Tuplas_Pag

    Número de tuplas por página. Página é a unidade usada para leitura e escrita no disco. O valor usado foi 32 tuplas por página, ou seja, foi usado uma página de 4 kbytes.

  - Num_Tup_Grupo

    Número de tuplas por grupo. Este parâmetro diz respeito aos grupos de tuplas formados para que a junção possa ser feita em partes que sempre caibam na memória dos processadores. O valor usado foi 64 tuplas por grupo.

  - Tam_Rel

    Tamanho, em páginas, das relações componentes da transação. O valor usado foi 2560 páginas por relação, o que significa uma relação de 10 Mbytes.

- Parâmetros Relativos ao Simulador

  - Instr_Interface

    Este parâmetro nos diz o número de instruções executadas devido à interface com o usuário. Foi usado o valor 100000 sugerido em [7].

- Instr_Msgs_Rede
  Número de instruções executadas pelo sistema operacional para processamento de uma chegada ou emissão pela rede. O valor usado foi 1000, o qual foi obtido de [1]
- Instr_ES
  Número de instruções executadas pelo sistema operacional para a realização de uma operação de E/S. O valor usado foi 1000.
- Instr_Msgs_Cpu
  Número de instruções executadas para se mandar uma mensagem a um processo executando na mesma CPU. O valor usado foi 500.
- Inst_Hash
  Número de instruções executadas para se aplicar uma função *hash* ao atributo de junção de uma tupla. O valor usado foi 1000, segundo [15].
- Tm_Ser_Disco
  Tempo médio de serviço do disco. Os valores usados foram 0.002 seg para os discos dos processadores homogêneos e 0.001 seg para o do processador heterogêneo. Estes valores foram obtidos de [7].
- Taxa_acerto
  Taxa de acerto nos *buffers* dos discos nas leituras/escritas. O valor usado foi de 40%, o que siginifica que em cada 10 vezes que se tenta ler/escrever no disco 6 vezes são realmente feitas no disco, as outras quatro são feitas em um *buffer* na memória.
- T_Simulacao
  Tempo de simulação. O valor usado foi de 20000 seg.

6 Resultados Numéricos

Os gráficos das figuras 3 e 4 apresentam um resumo dos resultados numéricos obtidos através de simulação. No primeiro deles, está mostrado o tempo de resposta das transações executando no sistema paralelo de banco de dados em função do número de processadores executores. Nesse caso, todos os processadores são idênticos; é uma configuração homogênea. Pode-se notar as seguintes observações:

- O tempo de resposta decresce com o aumento do número de processadores que atuam em paralelo;
- Existe um ponto para o qual o aumento de processadores não traz diminuição significativa no tempo de resposta; esse ponto para os parâmetros estudados ocorre por volta de 30 processadores. A existência desse ponto deve-se à concentração de dados em um processador (*data skew*), que faz com que o tempo de resposta da transação passe a ser dominado pelo mais longo componente. A partir daí, produz pouco efeito o acréscimo de mais processadores.

O gráfico da figura 4 exibe o *speedup* da arquitetura heterogênea em função do número de processadores. Neste contexto, *speedup* é definido como a razão entre o tempo de resposta de uma

transação no sistema homogêneo pelo tempo de resposta da mesma transação numa arquitetura heterogênea de custo equivalente. Observa-se no referido gráfico que o *speedup* é maior que 1 em todos os casos estudados, o que indica a superioridade da proposta heterogênea. É interessante observar também que o *speedup* cresce com o aumento do número de processadores. Isto ocorre pois o gargalo representado pelo processador que tem a maior concentração de dados é minimizado pelo processador mais rápido da configuração heterogênea.

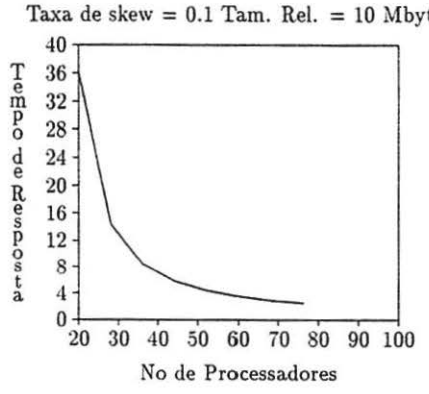

Figura 3: Tempo de Resposta × Número de Processadores

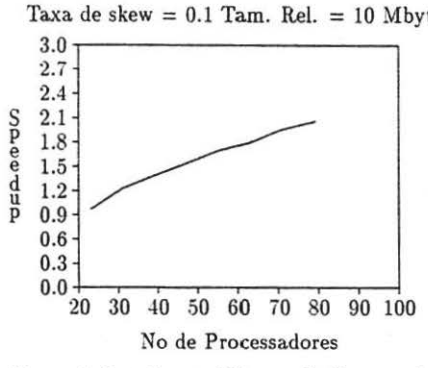

Figura 4: Speedup × Número de Processadores

7 Conclusões

Este trabalho apresenta o problema de *data skew* que limita o desempenho de sistemas paralelos de banco de dados. Através do desenvolvimento de um simulador de tais sistemas, algumas alternativas de solução foram consideradas. Em especial, o artigo trata da proposta de uma arquitetura heterogênea, composta de vários processadores idênticos e um de maior capacidade computacional. A partir dos resultados numéricos obtidos verifica-se que um sistema paralelo heterogêneo

tem melhor desempenho que um homogêneo de mesmo custo. Deve ainda ser ressaltado que a proposta aqui apresentada é uma abordagem completamente inédita para o problema de *data skew* em sistemas paralelos de banco de dados. A simulação desenvolvida considera apenas o algoritmo de junção por espalhamento (*hash join algorithm*). Um extensão prevista para este trabalho será a simulação de outros algoritmos, entre eles o algoritmo de junção por intercalação (*sort merge join algorithm*) [16].

Referências

[1] Bastos, A.M.C.; Menascé, D.A.; Soares, L.F.G. Comparação de desempenho de paradigmas para acesso distribuído a bancos de dados em redes locais. (Comunicação pessoal).

[2] Bitton, D.; Boral, H.; Dewitt, J. D.; Kevin, W. W. Parallel algorithms for the execution of relational database operations. ACM Transaction on Database Systems 8, 3(September 1983), pp 325-353.

[3] Boral, H. et al. Prototyping BUBBA: A highly parallel database system. IEEE Transactions on Knowledge and Data Engineering 2, 1(March 1990), 4-24.

[4] Dewitt, D.J.; Gray, J. Parallel database systems: The future of database processing or a passing fad? SIGMOD Record 19, 4(December 1990), 104-112.

[5] DeWitt, D.; Gray, J. Parallel database systems: The future of high performance database systems. Communications of the ACM 35, 6(June 1992), 85-98.

[6] Dewitt, D. et al. The Gamma database machine project. IEEE Transactions on Knowledge and Data Engineering 2, 1 (Mar. 1990),pp 44-62.

[7] Heidelberger, P. A performance comparison of multimicro and mainframe database architetures. IEEE Transactions on Software Engineering 14, 4(April 1988), 522-531.

[8] Hua, K.A.; Lee, C.; Peir, J. Interconnecting shared-everyting systems for efficient parallel query processing. Proceedings of the First International Conference on Parallel and Distributed Information Systems, Miami Beach, Florida, December 1991, pp 262-270.

[9] Lakshmi, M.S.; Yu, P.S. Effectiveness of parallel joins. IEEE Transactions on Knowledge and Data Engineering 2, 4(December 1990), 410-424.

[10] Menasce, D.; Almeida, V.A.F. Cost-performance analysis of heterogeneity in supercomputer architectures. Proceedings of Supercomputing'90, New York, November 1990, pp 169-177.

[11] Mishra, P.; Eich, M. Join processing in relational databases. ACM Computing Surveys 24, 1(March 1992), 63-113.

[12] Pirahesh, H.; Mohan, C.; Cheng, J.; Liu, T.S.; Selinger, P. Paralellism in relational data base systems: Architetural issues and design approaches. Proceedings of the Second International Symposium on Databases in Parallel and Distributed Systems, Dublin, Ireland, July 1990, pp 4-29.

[13] Schneider, D.A.; DeWitt, D.J. A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. Proceedings of the ACM SIGMOD International Conference on Management of Data, Portland, Oregon, June 1989, pp 110-121.

[14] Stonebraker, M.M. The case for shared nothing. Database Engineering. 9,1 (1986).

[15] Valduriez, P.; Gardarin, G. Join and semijoin algorithms for a multiprocessor database machine. ACM Transactions on Database Systems 9, 1(March 1984), 133-161.

[16] Wolf, J.L.; Dias, D.M.; Yu, P.S. An effective algorithm for parallelizing sort merge joins in the presence of data skew. Proceedings of the Second International Symposium on Databases in Parallel and Distributed Systems, Dublin, Ireland, July 1990, pp 103-115.

[17] Wolf, J.L.; Dias, D.M.; Yu, P.S.; Turek, J. An effective algorithm for parallelizing hash joins in the presence of data skew. Proceedings of the 7th International Conference on Data Engineering, Kyoto, Japan, 1991, pp 200-209.