Mineração Assíncrona de Regras de Associação em Sistemas de Memória Compartilhada-Distribuída

A. Veloso [1], B. Coutinho [1], B. Pôssas [1], G. Menezes [1] W. Meira Jr. [1], M. Carvalho [1], C. Amorim [2]
[1] Departamento de Ciência da Computação, Universidade Federal de Minas Gerais
Av. Antônio Carlos, 6627, Belo Horizonte, Brasil
{adrianov,coutinho,bavep,gms,meira,mlbc}@dcc.ufmg.br
[2] Fundação Coppetec, Universidade Federal do Rio de Janeiro, Rio de Janeiro
Cidade Universitária bloco I 2000, Ilha do Fundão, Rio de Janeiro, Brasil
amorim@cos.ufrj.br

Resumo—

Encontrar as regras de associação presentes em grandes bases de dados é um importante problema em Mineração de Dados. Existe uma grande necessidade de desenvolver algoritmos paralelos para esse problema, uma vez que ele corresponde a um processo computacional muito custoso. No entanto, a maioria dos algoritmos propostos para minerar tais regras seguem uma busca iterativa, que impõe a necessidade de sincronização ao final de cada iteração, degradando o desempenho. Outra deficiência desses algoritmos é proveniente da contenção que ocorre no barramento de entrada e saída, uma vez que todos os processadores devem acessar simultaneamente suas respectivas porções da base de dados. Mais ainda, esses algoritmos usam somente esquemas de balanceamento de carga estático, baseados na decomposição inicial dos dados, e depois disso eles assumem uma carga homogênea, o que está longe da realidade, já que a carga pode variar a cada iteração do algoritmo. Nesse artigo nós apresentamos um eficiente algoritmo paralelo para minerar regras de associação em sistemas de memória Distribuída-Compartilhada. Cada processador realiza sua tarefa de mineração sem efetuar nenhuma sincronização, e a carga é continuamente balanceada entre os processadores. Mais importante, nosso algoritmo realiza apenas um acesso à base de dados, evitando o problema de contenção no sistema de entrada e saída. Os experimentos mostram que nosso algoritmo paralelo proporciona ganhos significativos quando comparado com sua parte sequencial.

Palavras-chave— **Regras de Associação, Mineração de Dados, Balanceamento de Carga, Escalabilidade, Computação Paralela.**

Abstract—

Mining association rules from large databases is an important problem in data mining. There is a need to develop parallel algorithms for this problem because it is a very costly computation process. However, almost all proposed algorithms for mining such rules follow an iterative bottom-up search approach, which imposes a synchronization in every iteration, degrading greatly their performance. Another deficiency of these algorithms comes from the contention on the shared I/O channel, since all processors may access their database partitions in the shared storage synchronously. Further, all extant algorithms use only a static load-balancing scheme based on the initial data decomposition, and after this they assume a homogeneous workload. This is far from reality, since the worload may change after each iteration of the algorithm. We proposed an asynchronous and efficient parallel algorithm for mining association rules on a distributed-shared memory enviroment. Each processor performs its task without any synchronization, and the workload can be continously balanced. Furthermore, our algorithm scans exactly once the database, avoiding the problem with I/O contention. The experiments show that our algorithm has greatly speed-up over its sequential counterpart.

Keywords— **Association Rules, Data Mining, WorkLoad Balance, Scalability, Parallel Computing.**

I. Introdução

A ampla difusão da utilização de computadores em empresas e no comércio, e a disponibilidade de melhores e maiores sistemas de armazenamento, causaram uma explosão no crescimento da quantidade de dados coletados e armazenados por empresas e organizações. Recentemente essas organizações vêm apresentando um grande interesse em encontrar padrões interessantes em seus dados, e utilizar esses padrões em diversas aplicações, seja para melhorar seus serviços ou para automatizar a realização de decisões. *Mineração de Dados* e *Descoberta de Conhecimento em Bases de Dados* é um novo e interdisciplinar campo de pesquisa que mescla idéias e conceitos de estatística, inteligência artificial, banco de dados e computação paralela. O principal desafio da Mineração de Dados é o processo de extração de conhecimento implícito em grandes e densas bases de dados. Em particular, nos interessamos em um aspecto de tal processo, mineração de *Regras de Associação*, cuja definição é encontrar o conjunto de todos os conjuntos de itens que freqüentemente ocorrem nas transações de uma base de dados, e desses conjuntos extrair regras que mostram como um conjunto de itens influencia na presença de outro conjunto. A mineração de regras de associação tem uma grande aplicabilidade em diversas áreas, incluindo suporte à decisões, formulação de estratégias de $marketing$ e previsões financeiras.

Embora a mineração de regras de associação tenha uma definição muito simples, ela representa um processo computacional extremamente intensivo, como veremos nas próximas seções. Uma vez que os dados estão crescendo tanto em dimensão (número de itens) quanto em tamanho (número de transações), uma das principais necessidades dos algoritmos de mineração de regras de associação é escalabilidade, ou seja, a habilidade de manipular densas e grandes bases de dados. Algoritmos sequenciais não podem prover escalabilidade, em termos de dimensão e tamanho dos dados, e desempenho. Dessa forma, precisamos focar em eficientes algoritmos paralelos. Neste artigo propomos um efi-

ciente algoritmo paralelo para minerar regras de associação, o qual necessita de apenas um acesso à base de dados e que, após uma fase de inicialização, comporta-se de maneira assíncrona. Também propomos duas formas de balanceamento de carga. A primeira é estática e baseada em uma heurística para a decomposição dos dados. A outra dinamicamente divide os dados entre os processadores.

O resto do artigo está organizado da seguinte forma. A Seção 2 apresenta uma descrição formal da mineração de regras de associação. A Seção 3 traz uma breve descrição do algoritmo **Eclat** [ZAK 97], que é nossa versão sequencial. Na Seção 4, apresentamos nosso algoritmo paralelo. Na Seção 5, descrevemos os resultados experimentais. Na Seção 6, discutimos os trabalhos correlatos. Seção 7 é a conclusão.

II. Mineração de Regras de Associação

Um importante problema em Mineração de Dados é encontrar associações presentes em densas bases de dados. O objetivo é verificar se a ocorrência de certos itens em uma transação pode ser usada para deduzir a ocorrência de outros itens, ou em outras palavras, encontrar relações de associatividade entre os itens baseando-se em suas ocorrências simultâneas. Caso essas associações sejam encontradas, elas podem ser usadas de diversas formas, como posicionamento de produtos, manutenção de estoques, etc. Dessa forma, foi dado início ao estudo das regras de associação [AGR 93]. Dado um conjunto de itens, devemos prever a ocorrência de outro conjunto de itens com um certo grau de confiança. Este problema está longe de ser trivial por causa do número exponencial de formas com que podemos agrupar tais itens.

Seja $I = \{I_1, I_2, ..., I_m\}$ o conjuto de m atributos, os chamados itens. Seja T o conjunto de transações onde cada uma delas é um subconjunto de I, e é unicamente identificada por um tid. Seja C um subconjunto de I, também chamado de $itemset$. Se C possuir k itens ele é chamado de k-itemset. Definimos o $suporte$ de um $itemset$ C de acordo com T como sendo: $\sigma(C) = \mid \{t \mid t \in T, C \subseteq t\} \mid$. Estatisticamente, o suporte representa a significância do $itemset$. Então $\sigma(C)$ é o número de transações que contêm C. Por exemplo, considere o conjunto de transações mostrado na Tabela I. O conjunto de itens I para essas transações é $\{A, B, C, D, E\}$. O suporte de $\{D, E\}$ é $\sigma(D, E) = 3$, e $\sigma(D, E, A) = 2$. Enquanto não sabemos qual o suporte de um $itemset$, ele é chamado de conjunto $candidato$.

Uma regra de associação é uma expressão da forma $X \Rightarrow Y$, onde $X \subseteq I$ e $Y \subseteq I$. O suporte s de uma regra $X \Rightarrow Y$ é definido como $\sigma(X \cup Y)/ \mid T \mid$, e a *confiança* c é definida como $\sigma(X \cup Y)/\sigma(X)$. Por exemplo, considere a regra $\{D, E\} \Rightarrow \{A\}$ (i.e., a presença de frauda e de leite em uma transação indica a presença de cerveja na mesma transação). O suporte dessa regra é $\sigma(D, E, A)/5 = 40\%$. A confiança dessa regra é $\sigma(D, E, A)/\sigma(D, E) = 66\%$. Uma

Tabela I
Base de Dados com as Transações

TID	Itens
1	B, C, E
2	A, B
3	A, C, D, E
4	A, B, D, E
5	C, D, E

regra que possui um grande grau de confiança (i.e. perto de 100%) geralmente é muito importante, porque ela provê uma previsão muito precisa da associatividade dos itens nessa regra. O suporte de uma regra também é importante, uma vez que ele indica o quão freqüente nas transações essa regra é. Regras com baixo suporte geralmente são pouco interessantes, uma vez que elas não descrevem significantemente grandes populações. Esta é uma das razões pela qual vários algoritmos descartam qualquer regra que não satisfaça uma condição de suporte mínimo, especificada pelo usuário. Esta filtragem feita pelo suporte mínimo requerido, também é importante para reduzir o número de regras de associação derivadas para um número manuseável. Note que o número de regras possíveis é proporcional ao número de subconjuntos de I, que é $2^{|I|}$. Por isso a filtragem é absolutamente necessária na maioria dos casos práticos.

A tarefa de descobrir as regras de associação resume-se em encontrar todas as regras $X \Rightarrow Y$, tal que seu suporte é maior ou igual a um dado suporte mínimo e sua confiança é maior ou igual a uma dada confiança mínima. O processo de descoberta de regras de associação é dividido em duas etapas. A primeira etapa é encontrar todos os $itemsets$ freqüentes (conjuntos de itens que possuem suporte maior ou igual ao suporte mínimo). A segunda etapa é gerar as regras de associação desses $itemsets$. O custo computacional requerido para encontrar todos os $itemsets$ freqüentes é muito maior que o de gerar as regras desses $itemsets$ freqüentes. Por isso neste artigo só trataremos da primeira etapa.

III. O Algoritmo ECLAT

Nesta seção apresentamos o algoritmo Eclat, que é nossa parte sequencial. Eclat (Equivalence CLAss Transformation) usa uma elegante técnica que diminui o número de candidatos gerados agrupando $itemsets$ relacionados, e que facilita a contagem do suporte dos $itemsets$ utilizando projeções verticais da base de dados. Detalharemos essas etapas nas próximas seções.

A. Decomposição do Espaço de Busca

Vamos considerar a base de dados de exemplo na Tabela I. Podemos enumerar todos os $itemsets$ freqüentes simples-

mente restringindo o espaço de busca.

No entanto, na prática temos apenas um limitado poder de processamento e uma limitada quantidade de memória, e todos os conjuntos candidatos não poderão ser armazenados simultaneamente em memória. Esta questão revela a necessidade de decompor o espaço de busca original em partições menores de tal forma que cada partição possa ser processada independentemente.

A seguir descrevemos como podemos realizar essa decomposição. Suponha que os itens em um *itemset* estejam lexicograficamente ordenados. Dois *itemsets* pertencem a uma mesma classe de equivalência, F_k, se eles compartilham um prefixo de tamanho k, onde o tamanho é o número de itens que compõem o prefixo. A Figura 1 mostra quatro classes de equivalência com prefixo de tamanho 1: $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$. Por exemplo, os *itemsets* $\{A, B\}$, $\{A, D\}$ e $\{A, E\}$ pertencem a uma mesma classe de equivalência, já que compartilham um prefixo de tamanho 1 (i.e., A). Ao final de cada iteração, uma classe pode ser dividida em sub-classes de tamanho menor. A motivação para esta definição é que cada classe de equivalência pode ser processada independentemente, de tal forma que os *itemsets* produzidos por uma classe de equivalência são independentes dos produzidos por outra classe qualquer.

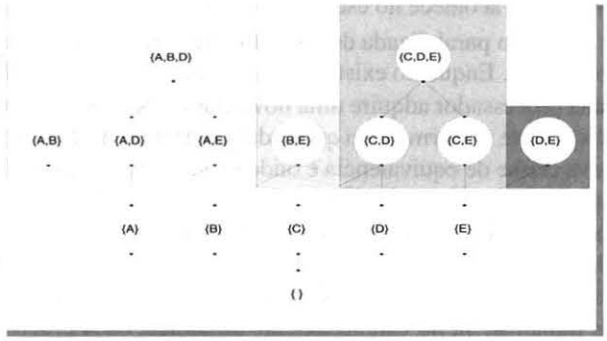

Figura 1. Decomposição do espaço de busca usando classes de equivalência com prefixo de tamanho 1

B. Projeção Vertical da Base de Dados

A base de dados de exemplo na Tabela I está no formato horizontal, com cada *tid* seguido pelos itens que formam a transação. Esse formato de base de dados impõe algumas restrições computacionais durante a fase de contagem do suporte dos *itemsets*, uma vez que ele nos força a acessar a base de dados pelo menos uma vez a cada iteração.

Eclat usa uma projeção vertical, que consiste de uma lista de *itemsets*, e cada *itemset* possui sua *tidlist* - uma lista de todos os identificadores das transações onde esse *itemset* ocorreu na base de dados. O formato vertical da base de dados possui três principais vantagens comparado ao formato horizontal. Primeiro, um k-itemset pode ser encontrado simplesmente unindo os itens e realizando a interseção das *tidlists* de quaisquer dois de seus $(k-1)$-subconjuntos. Por exemplo, se unirmos os itens e interseccionarmos as *tidlists* de $\{A, B\}$ e $\{A, E\}$ nós obtemos o *itemset* $\{A, B, E\}$, o qual aparece somente na quarta transação da base de dados da Tabela I. Segundo, as *tidlists* contêm toda a informação relevante sobre um *itemset*, e nos possibilita evitar o acesso de toda a base de dados para computar o suporte de um *itemset*. Terceiro, quanto maior for o *itemset*, menor será sua *tidlist*, fato que acontece praticamente sempre, e que resulta em interseções cada vez mais rápidas.

C. Busca por Itemsets Freqüentes

Eclat usa um esquema de busca *bottom-up* para enumerar todos os *itemsets* freqüentes. O ponto de partida são os itens freqüentes, os quais são determinados com apenas um acesso à base de dados. Para cada item encontrado na transação, atualiza-se sua *tidlist* e incremeta-se seu suporte. Os itens são usados para gerar os candidatos de tamanho 2. Os candidatos de tamanho 2 que forem infreqüentes são descartados e todos os 2-*itemsets* freqüentes são usados para criar as classes de equivalência com prefixo de tamanho 1.

Partindo de uma classe de equivalência, somos capazes de enumerar todos os *itemsets* freqüentes com o prefixo que forma a classe de equivalência. Então, para cada classe de equivalência criada, Eclat gera os candidatos de tamanho 3. Na próxima iteração, os candidatos classificados como freqüentes são usados para determinar o próximo nível de *itemsets* freqüentes (i.e., os $(k-1)$-*itemsets* freqüentes são usados para determinar os k-*itemsets* freqüentes). O processo termina quando não existem mais candidatos, e todos os *itemsets* freqüentes foram encontrados.

IV. Mineração Paralela de Regras de Associação

Como mencionado anteriormente, descobrir as regras de associação é uma tarefa extremamente intensiva computacionalmente, e a partir de um certo tamanho da base de dados, torna-se crucial empregar o poder computacional combinado de vários processadores para uma rápida resposta e escalabilidade. Nesta seção apresentamos um novo algoritmo paralelo para a descoberta dos *itemsets* freqüentes.

Nosso algoritmo é baseado nos conceitos apresentados na seção passada. Ele usa a decomposição do espaço de busca e a projeção vertical da base de dados. A carga de trabalho é distribuída entre os processadores de forma com que cada processador possa determinar os *itemsets* freqüentes independentemente, utilizando simples interseções de *tidlist*. Nosso algoritmo acessa a base de dados apenas uma vez, diminuindo drasticamente os custos impostos pela contenção na entrada e saída. Após uma fase de inicialização, ele não necessita de nenhuma sincronização. A seguir descrevemos

cada etapa de nosso algoritmo mais detalhadamente.

A. *Fase de Inicialização*

A fase de inicialização consiste de três etapas. Primeiro, a base de dados é acessada, todos os itens freqüentes são encontrados e suas *tidlists* são criadas. Segundo, os 2-*itemsets* são computados realizando interseções entre as *tidlists* dos itens freqüentes. Terceiro, os 2-*itemsets* são agrupados em classes distintas aplicando-se a decomposição por classes de equivalência, e o conjunto de classes independentes é criado.

As classes são distribuídas entre os processadores, e um certo grau de balanceamento de carga é atingido por associar um determinado peso para cada classe de equivalência baseado no número de elementos na classe. Ordena-se as classes de equivalência pelos pesos, e atribui-se cada classe iterativamente para o processador com menos carga (menor soma de pesos). Se dois processadores possuem uma mesma estimativa de carga de trabalho, o empate é desfeito selecionando o processador com o maior identificador. Esta é apenas uma heurística de custo, e podem existir heurísticas melhores que essa. O processo relativo a fase de inicialização está representado na Figura 2.

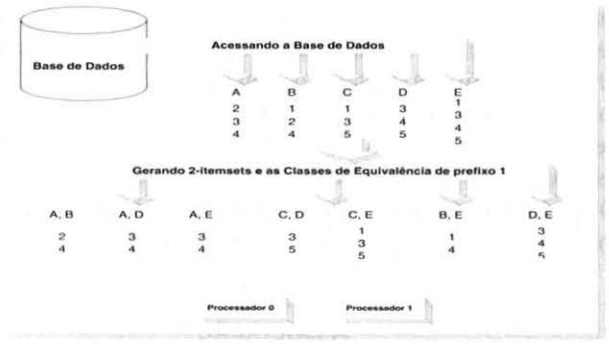

Figura 2. A Fase de Inicialização

B. *Fase Assíncrona*

Ao final da fase de inicialização, as classes de equivalência relevantes estão disponíveis localmente em cada processador. Agora, cada um deles pode gerar independentemente todos os *itemsets* freqüentes provenientes das classes que foram atribuídas a ele. Cada classe é totalmente processada antes de passar para a próxima classe.

Embora tenhamos diminuído o desbalanceamento de carga entre os processadores ao aplicar nossa heurística de custo, graças à natureza do algoritmo, quase sempre a carga de trabalho atribuída a cada processador é bastante diferente. Em um esquema estático de balanceamento de carga, nenhuma migração de carga para corrigir o desbalanceamento é permitida. O balanceamento de carga dinâmico procura resolver este problema distribuindo a carga pertencente a processadores mais carregados para os processadores menos carregados no momento. Mas tudo isso tem um preço, e observamos que o movimento de computação possui um vínculo com a movimentação de dados, uma vez que o processador responsável por uma determinada tarefa necessita dos dados associados a ela. Por isso, esta forma de balanceamento dinâmico de carga implica em custos adicionais de trabalho e movimentação de dados, que num sistema de memória distribuída pode causar grande degradação devido à alta taxa de comunicação. No entanto, o balanceamento de carga é um fator crucial para o desempenho se existe um grande desbalanceamento inerente às tarefas, se os processadores possuem diferentes capacidades computacionais ou se a carga muda com o passar do tempo.

Portanto, o principal desafio que procuramos resolver é a investigação de uma forma de balanceamento dinâmico de carga que não necessite de muita movimentação de dados e comunicação entre os processadores. Nossa técnica funciona da seguinte forma: cada processador trabalha com apenas uma classe de equivalência, e assim que essa classe for totalmente processada, esse processador busca outra classe, que ainda não está sendo processada, para trabalhar. Essa técnica busca evitar que alguns processadores fiquem sem carga alguma enquanto outros permaneçam muito carregados, problema que acontece no esquema de balanceamento estático.

A sessão paralelizada do algoritmo funciona como descrito a seguir. Enquanto existem classes ainda não processadas, cada processador adquire uma nova classe e a processa completamente. A forma com que cada processador adquire uma nova classe de equivalência é onde reside a diferença entre o balanceamento de carga estático e o dinâmico. No balanceamento estático, cada processador só pode adquirir uma nova classe dentre as classes inicialmente atribuídas a ele, e, caso não existam mais classes disponíveis, esse processador pára de trabalhar. Já no balanceamento dinâmico, cada processador pode adquirir qualquer classe, uma vez que não ocorre nenhuma distribuição inicial das classes aos processadores. Contudo, para assegurar que os processadores trabalharão com classes de equivalência distintas, torna-se necessária a utilização de um semáforo. Conforme mencionado anteriormente, o processamento de cada classe de equivalência é independente.

C. *O Processo Paralelo*

Para ilustrar as idéias por trás de nosso algoritmo, vamos considerar o exemplo na Figura 3. Por simplicidade e sem perda de generalidade, vamos assumir uma base de dados com apenas uma transação. Nesse caso especial, todos os possíveis *itemsets* são freqüentes, não importanto o valor do suporte mínimo. Vamos assumir também apenas dois processadores.

Durante a fase de inicialização, acessamos a base de dados e encontramos os itens freqüentes, geramos os 2-$itemsets$ freqüentes e agrupamô-los em classes de equivalência com prefixos de tamanho 1: [A], [B], [C], [D] e [E]. Essas classes foram atribuídas aos dois processadores, de tal forma que o processador 0 adquiriu as classes [A] e [D], e o processador 1 adquiriu as classes [B], [C] e [E]. Um certo grau de balanceamento foi atingido, já que as classes assinaladas ao processador 0 possuem 7 $itemsets$, e as classes assinaladas ao processador 1 possuem 8 $itemsets$.

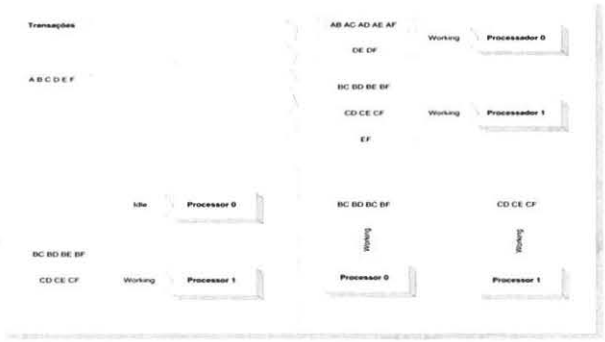

Figura 3. O Processo Paralelo e o Balanceamento de Carga

A fase assíncrona tem início, e os dois processadores começam seu trabalho. Suponha que, simultaneamente, o processador 0 enumerou todos os $itemsets$ freqüentes provenientes das classes [D] e [A] e o processador 1 enumerou todos os $itemsets$ freqüentes provenientes da classe [E]. Nesse momento o processador 0 fica ocioso, já que terminou todo seu trabalho, mas o processador 1 ainda possui as classes [B] e [C] para computar. No esquema de balanceamento estático, a situação ociosa do processador 0 continua até o final de todo o processo. Já no balanceamento dinâmico, o processador 0 adquire a classe [B], deixando apenas a classe [C], tornando o processo bem mais eficiente na maioria dos casos, uma vez que todos os processadores tendem a terminar suas tarefas ao mesmo tempo. Quando não existe mais trabalho a ser feito, todos os processadores são sincronizados, e o processo termina.

D. Fase de Geração de Regras

Como todos os $itemsets$ freqüentes foram encontrados, podemos obter relações entre diferentes itens, as regras de associação. Os $itemsets$ enumerados por cada processador são todos agrupados e as regras são facilmente geradas.

V. Avaliação Experimental

Nesta seção avaliamos o desempenho de nosso algoritmo e o comparamos com sua parte sequencial, Eclat. Além disso, empregamos duas métricas básicas para estudar a escalabilidade e o balanceamento de carga: tempo de execução total e $itemsets$ processados. A primeira métrica é o tempo de execução total, decorrido durante todo o processo de mineração, excluindo a geração de regras. Essa métrica nos dá uma boa noção dos ganhos realmente providos por nosso algoritmo. A segunda métrica é o número de $itemsets$ processados, ou seja, o número de conjuntos candidatos gerados durante a mineração. Essa métrica pode ser considerada uma boa aproximação da quantidade de trabalho realizado, e irá nos ajudar a entender os diferentes tempos de execução observados, mais ainda, em ambientes homogêneos nos ajuda a entender os efeitos do balanceamento de carga.

A. Bases de Dados Sintéticas

Em nossos experimentos empregamos bases de dados geradas sinteticamente, seguindo o procedimento descrito em [AGR 94]. Essas bases de dados procuram imitar as transações em um ambiente comercial, onde pessoas tendem a comprar conjuntos de itens. Preferimos utilizar as bases de dados sintéticas pela facilidade de associarmos a elas os efeitos causados pela variação da dimensão (número de itens) e do tamanho (número de transações) da base de dados. As bases de dados são chamadas de T.I.D., onde T corresponde ao tamanho médio das transações (dimensão da base de dados), I corresponde ao tamanho máximo dos $itemsets$ freqüentes (chamados de $itemsets\ maximais$), e D corresponde ao número de transações (tamanho da base de dados). As bases de dados utilizadas possuem de 500 a 1000 itens distintos.

B. ThreadMarks^{TM}

O desenvolvimento de nosso algoritimo, bem como todo o estudo de seu comportamento e desempenho, foi baseado numa arquitetura chamada $ThreadMarks^{TM}$.

$ThreadMarks^{TM}$ é um software que permite uma programação concorrente no estilo de memória compartilhada em um ambiente multiprocessado com memória distribuída, ou em uma rede de computadores. Quando a memória "compartilhada" é acessada por um processador, $ThreadMarks^{TM}$ determina quais os dados estão presentes naquele processador, e, se necessário, transmite os dados para o processador sem a necessidade da intervenção do programador. Quando a memória "compartilhada" é modificada em um processador, $ThreadMarks^{TM}$ assegura que os outros processadores serão notificados da mudança, de tal forma que eles não usem dados obsoletos. Essa notificação não é realizada imediatamente nem globalmente. Em vez disso, essa notificação ocorre entre dois processadores quando esses são sincronizados. Essa abordagem não prejudica o resultado de programas livres de condição de corrida, e a acumulação de muitas dessas notificações em uma só mensagem durante a sincronização, reduz bastante os custos causados pelas comunicações entre processadores.

C. Ambiente de Execução

Todos os experimentos foram realizados em dois ambientes: homogêneo e heterogêneo. Em ambos os casos as bases de dados foram armazenadas em disco local.

Nosso ambiente de execução homogêneo é formado por um *cluster* de 8 $PentiumIII$, com 512MB de memória, todos rodando a 650MHZ. As máquinas estão conectadas por uma rede $GigaNet$ de 1.2 Giga/bits.

Nosso ambiente de execução heterogêneo é formado por 8 $Athlons$, 4 deles possuem 1GB de memória e rodam a 1GHZ, enquanto os outros possuem 256MB de memória e rodam a 768MHZ. As máquinas estão conectadas por uma rede $Ethernet$ de 100Mb/s.

D. Speedup

De forma a estudarmos a eficiência de nosso algoritmo paralelo, investigamos seu *speedup*, que é definido como a redução do tempo de resposta com o aumento do número de processadores. Quanto mais processadores, mais rápida deve ser a computação. O *speedup* "ideal" é uma função linear com o número de processadores. Para avaliar o *speedup*, executamos o algoritmo para apenas uma base de dados e variamos o número de processadores. Antes dos resultados, vamos apresentar algumas características da base de dados utilizada nesse experimento. A Figura 4 mostra o grau de desbalanceamento apresentado por essa base de dados (T25.I20.D10K) em termos de conjuntos candidatos processados em cada classe de equivalência. Observamos a necessidade da realização do balanceamento de carga. O tempo de execução base foi obtido executando o algoritmo Eclat.

D.1 Ambiente Homogêneo

A Figura 5 apresenta os ganhos de desempenho com o aumento do número de processadores utilizados, para minerar a base de dados T25.I20.D10K. Podemos perceber que, mesmo em ambientes homogêneos, os resultados obtidos com o balanceamento dinâmico são melhores que os obtidos com balanceamento estático. A explicação para esse fato é que a heurística utilizada para decompor os dados só leva em consideração a quantidade de $itemsets$ que inicialmente estão naquela classe (i.e., os 2-$itemsets$ freqüentes de cada classes). Contudo, a quantidade de $itemsets$ candidatos de maior tamanho gerados pode variar de classe para classe, como observamos na Figura 4. Dessa forma, alguns processadores terminarão sua tarefa mais cedo que outros, o que é comprovado pelas Figuras 6 e 7, que mostram o número de $itemsets$ realmente processados por cada processador. Podemos perceber que com a utilização do balanceamento dinâmico, os processadores possuem tempos de execução mais próximos e o número de $itemsets$ processados é melhor distribuído.

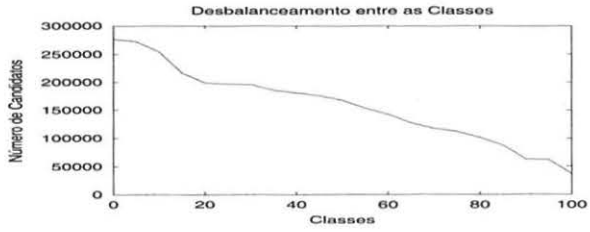

Figura 4. Desbalanceamento entre as Classes da Base de Dados T25.I20.D10K

Figura 5. Ganhos de Desempenho obtidos com os Balanceamentos Estático e Dinâmico

Figura 6. Relação do Tempo de Execução de cada Processador com o Número de Candidatos enumerados por ele, utilizando Balanceamento Estático

Figura 7. Relação do Tempo de Execução de cada Processador com o Número de Candidatos enumerados por ele, utilizando Balanceamento Dinâmico

D.2 Ambiente Heterogêneo

A Figura 8 apresenta o tempo de execução obtido por cada um dos oito processadores utilizados para minerar a base de dados T25.I20.D10K a um suporte mínimo de 1%, utilizando-se o balanceamento de carga estático. O número associado a cada vértice corresponde a quantidade total de $itemsets$ candidatos processados por aquele processador. Podemos perceber claramente que, para sistemas onde suas unidades possuem diferente poder de processamento, a quantidade de trabalho não pode ser modelada baseando-se ex-

clusivamente no número de *itemsets* processados, o que diminuiu a eficácia de nossa heurística. A Figura 9 apresenta o experimento para a mesma base de dados, mas dessa vez aplicando-se o balanceamento de carga dinâmico. Podemos perceber que os tempos de execução gastos por cada processador ficaram mais próximos, embora o número de *itemsets* candidatos tenha divergido mais. Isso mostra que o balanceamento dinâmico adaptou o número de *itemsets* candidatos ao poder de processamento do processador. Os resultados mostram a validade do balanceamento dinâmico, principalmente para ambientes heterôgeneos.

Figura 8. Relação do Tempo de Execução de cada Processador com o Número de Candidatos enumerados por ele, utilizando Balanceamento Estático

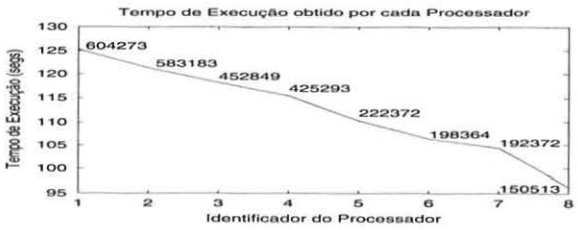

Figura 9. Relação do Tempo de Execução de cada Processador com o Número de Candidatos enumerados por ele, utilizando Balanceamento Dinâmico

E. Scaleup

Scaleup é definido como o desempenho obtido variando-se o número de processadores quando o tamanho da base de dados é aumentado proporcionalmente ao número de processadores utilizados. Se o algoritmo é eficiente seu desempenho deve manter-se uniforme quando o número de processadores e o tamanho da base de dados aumentam proporcionalmente. Para avaliarmos o *scaleup*, executamos o algoritmo variando proporcionalmente (no caso seguindo uma função linear) o tamanho da base de dados e o número de processadores. O número de processadores envolvidos variou de 1 a 8, e o tamanho da base de dados variou de 2000 a 16000 transações.

E.1 Ambiente Homogêneo

A Figura 10 mostra o tempo de execução do algoritmo paralelo relativamente ao tempo de execução do algoritmo sequencial para bases de dados de diferentes tamanhos. O número sobre cada vértice do gráfico corresponde ao número de transações de cada base de dados utilizada. Podemos perceber que o balanceamento dinâmico aproveita melhor o paralelismo. Justificamos a defasagem observada na utilização do balanceamento estático por imprecisões na decomposição inicial dos dados.

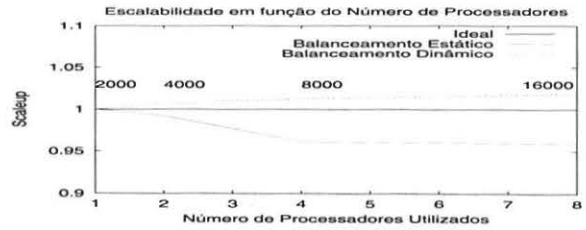

Figura 10. Relação entre o Tamanho da Base de Dados e Número de Processadores com o Tempo de Execução em Ambientes Homôgeneos

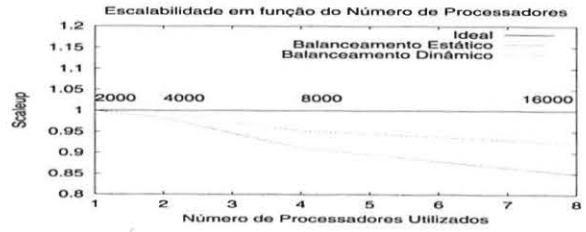

Figura 11. Relação entre o Tamanho da Base de Dados e Número de Processadores com o Tempo de Execução em Ambientes Heterôgeneos

E.2 Ambiente Heterogêneo

A Figura 11 apresenta o tempo de execução do algoritmo paralelo relativamente ao tempo de execução do algoritmo sequencial. Variamos o número de processadores e o tamanho das bases de dados. Percebemos que o algoritmo torna-se menos escalável ao empregarmos o balanceamento estático. Esse comportamento ocorre porque a precisão de nossa heurística é muito dependente da base de dados. O balanceamento dinâmico não é tão dependente da base de dados, possibilitando uma adaptação contínua e mais flexível, capaz de corrigir eventuais desbalanceamentos que não seriam corrigidos pelo balanceamento estático.

F. Discussão

Os ganhos de desempenho providos por nosso algoritmo paralelo são dependentes das características da base de dados e do suporte mínimo empregado na mineração. Esses dois fatores determinam a quantidade e o tamanho das classes de equivalência geradas, e conseqüentemente determinam o aumento ou a diminuição da flexibilidade e eficácia do balanceamento, que é baseado nessas classes. Observamos que para bases de dados mais densas, e com a utilização de suportes mínimos menores, nosso algoritmo paralelo obtém os

melhores resultados, uma vez que o tamanho da sessão paralela e as oportunidades de balanceamento aumentam.

VI. Trabalhos Correlatos

Nesta seção provemos uma visão geral dos algoritmos paralelos já desenvolvidos para minerar regras de associação.

A. Memória Distribuída

Vários outros algoritmos de mineração de associações em bases de dados já foram paralelizados. O mais famoso é chamado **Apriori**. Um dos primeiros trabalhos nesse sentido em memória distribuída, surgiu em [PAR 95], onde os autores propuseram uma abordagem paralela baseada na geração de candidatos, mas a alta taxa de comunicação realizada não permitiu resultados bons tanto em termos de desempenho como em termos de escalabilidade. Em [JOS 00] os autores propuseram a utilização de agregação de memória, aumentando a eficiência e diminuindo a taxa de comunicação.

Em [CHE 99] os autores apresentaram um trabalho interessante, onde propuseram técnicas de balanceamento de carga baseadas no particionamento da base de dados. A forma como os dados devem ser particionados é uma função da entropia observada em cada partição. Os resultados da paralelização do algoritmo Apriori apresentados nesse trabalho são muito bons. Infelizmente não podemos compará-los com os nossos, uma vez que os algoritmos sequenciais são diferentes. A saber, Eclat possui um desempenho aproximadaamente 10 vezes melhor que Apriori. Os autores conseguiram resultados surpreendentemente bons, porque a métrica de balanceamento mudou também o comportamento do algoritmo Apriori, mas não sabemos quantificar essa melhoria.

B. Memória Compartilhada

Tanto em [ZAK 96] como em [ZAK 98], M. J. Zaki e seus companheiros propuseram o paralelismo do algoritmo Eclat em memória compartilhada. O algoritmo apresentou bons resultados, mas o balanceamento dinâmico proposto em nosso trabalho apresentou resultados ainda melhores. Em [CHE 98] os autores propuseram um paralelismo do algoritmo Apriori. Sua proposta é baseada na geração de candidatos. Essa técnica apresentou resultados bem melhores em ambientes de memória compartilhada do que em ambientes de memória distribuída.

VII. Conclusões

Neste trabalho propusemos uma nova paralelização para o algoritmo Eclat, cuja finalidade é minerar regras de associação, um problema real bastante complexo que necessita de soluções eficientes.

Utilizamos o paradigma de memória compartilhada-distribuída, no qual nosso algoritmo apresentou bons resultados tanto em termos de escalabilidade quanto em termos de desempenho. Propusemos duas formas de balanceamento de carga, uma estática e uma dinâmica. A abordagem dinâmica mostrou-se sempre melhor, e ideal para ambientes heterôgeneos, que são os disponíveis na maioria dos casos. A abordagem estática mostrou-se competitiva em ambientes homogêneos, revelando que, para esse tipo de ambiente, a quantidade de trabalho efetivamente realizado pode ser bem estimado pelo número de $itemsets$ candidatos.

Pretendemos explorar abordagens paralelas mais poderosas, que utilizem um $cluster$ de màquinas multi-processadas. Acreditamos que essas arquiteturas possam oferecer melhores oportunidades de balanceamento, e conseqüentemente melhores resultados.

Referências

[AGR 93] AGRAWAL, R. IMIELINSKI, T. and SWAMI, A. Mining association rules between sets of items in large databases. In **Proc. of 1993 ACM-SIGMOD Int. Conf. on Management of Data**, Washington, D.C., 1993.

[AGR 94] AGRAWAL, R. and SWAMI, A. Fast Algorithms for Mining Association Rules. In **Proc. of the 20th Int. Conf. on Very Large Databases**, Santiago, Chile, 1994.

[CHE 98] CHEUNG, D. W., HU, K. and XIAO, Y. et al. Asynchronous Parallel Algorithm for Mining Association Rules on a Shared-Memory Multi-processors. In **10th ACM Symp. Parallel Algorithms and Architectures**, 1998.

[CHE 99] CHEUNG, D. W. and XIAO, Y. et al. Effect of Data Distribution in Parallel Mining of Associations. In **the 12nd Pacific-Asia Conference on Knowledge Discovery and Data Mining**, New York, 1998.

[ZAK 97] ZAKI, M., PARTHASARATHY, S., OGIHARA, M. and LI, W. et al. New Algorithms for Fast Discovery of Association Rules. In **Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining**, New Port Beach, California, 1997.

[ZAK 98] ZAKI, M., PARTHASARATHY, S. and LI, W. et al. A Localized Algorithm for Parallel Association Mining. In **9th ACM Symp. Parallel Algorithms and Architectures**, 1998.

[ZAK 96] ZAKI, M., PARTHASARATHY, S. and LI, W. et al. Parallel Data Mining for Association Rules on Shared-Memory Systems. In **Supercomputing'96**, Pittsburg, PA, Nov 1996.

[PAR 98] PARTHASARATHY, S., ZAKI, M. and LI, W. et al. Memory Placement Techniques for Parallel Association Mining. In **4th Intl. Conf. Knowledge Discovery and Data Mining**, 1998.

[PAR 95] PARK, J., CHEN, M. and YU, P. et al. Efficient Parallel Data Mining for Association Rules. In **Proc. of 1995 Int. Conf. on Information and Knowledge Management**, Baltimore, MD, Nov 1995.

[HAN 97] HAN, E., KARYPIS, G., and KUMAR, V. et al. Scalable Paralell Data Mining for Association Rules. In **Proc. of 1997 ACM-SIGMOD Int. Conf. on Management of Data**, Tucson, Arizona, 1997.

[JOS 00] JOSHI, M., HAN, E., KARYPIS, G., and KUMAR, V. et al. Efficient Parallel Algorithms for Mining Associations. In **M. J. Zaki and C.-T. Ho, editors, Lecture Notes in Computer Science: Lecture Notes in Artificial Inteligence**, volume 1759, Springg-Verlag, To appear.