ANÁLISE DE AGRUPAMENTOS FCM
UTILIZANDO PROCESSAMENTO PARALELO

Marta Vidal Modenesi

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA CIVIL.

Aprovada por:

_____
Prof. Alexandre Gonçalves Evsukoff, D.Sc

_____
Prof. Myrian Christina de Aragão Costa, D.Sc

_____
Prof. Luiz Landau, D.Sc

_____
Dr. Sebastião Cesar Assis Pereira, D.Sc

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2008

MODENESI, MARTA VIDAL

Análise de Agrupamentos FCM Utilizando Processamento Paralelo [Rio de Janeiro] 2008

IX, 110, 29,7 cm (COPPE/UFRJ,M.Sc., Engenharia Civil, 2008)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Mineração de Dados
2. Análise de Agrupamentos
3. Otimização
4. Processamento Paralelo

I. COPPE/UFRJ II. Título (série)

DEDICATÓRIA

Dedico este trabalho a todas as mulheres que têm filhos, marido, pais idosos, trabalho, cachorro e tudo mais o que a vida acaba nos dando de presente e que do meio de tudo isso, resolvem fazer mestrado.

AGRADECIMENTOS

À Agência Nacional do Petróleo (ANP) pela bolsa durante boa parte desse trabalho.

Na COPPE, onde eu sonhava em fazer o mestrado....

Ao Centro de Computação de Alto Desempenho da UFRJ (NACAD-COPPE/UFRJ) onde os testes do trabalho foram realizados.

Aos funcionários da COPPE, NACAD, LAB2M e do LAMCE pelo apoio e ajuda que prestaram.

Aos amigos que fiz durante o curso, pela sua postura de cooperação e camaradagem.

À amiga Jaci Guigon que me incentivou a fazer o mestrado na área de petróleo. Por suas orientações e apoio nas matérias.

Ao professor Luiz Landau, que tem o dom de ouvir os alunos com o coração, pelo apoio durante todo o curso e pela oportunidade de fazer o mestrado na área de petróleo que tantas oportunidades me abriu.

Ao apoio e suporte da professora e orientadora Myrian Christina de Aragão Costa, pessoa muito humana que se tornou uma amiga.

Ao professor e orientador Alexandre G. Evsukoff por todo incentivo, apoio, idéias e ajuda nesse trabalho. Também por ser sempre um parceiro, pelas opiniões equilibradas, pela paciência e por responder sempre a todos os pedidos de socorro com muita presteza.

Na PETROBRAS ....

Ao amigo Jorge Luiz Sued, gerente da área onde trabalho na PETROBRAS, cujo apoio foi fundamental para que eu terminasse o mestrado, me liberando em vários dias para que pudesse escrever a dissertação.

À Martha Gomes de Souza e a PETROBRAS pelo apoio e incentivo durante o período do mestrado.

Na família ....

À minha amiga Regina Lúcia Marin pelo apoio nas revisões dos artigos que foram escritos a partir desse trabalho.

À minha tia Maria Regina, pelo incentivo e palavras de ânimo.

Aos meus pais por me ensinarem a procurar fazer sempre o melhor e pelo apoio e atenção dedicados à minha filha caçula durante o tempo do mestrado.

Aos meus filhos por me ensinarem a amar as pessoas como elas são. Também por assumirem mais responsabilidades para ajudar a mãe a estudar.

Ao meu marido e eterno amor que tive a sorte de conhecer tão jovem, Ricardo, pelo apoio incondicional em todas as circunstâncias, por assumir tantas coisas para que eu pudesse realizar esse trabalho.

E acima de tudo

Agradeço à Deus pelo dom da minha vida, de toda minha família e de todas as pessoas que conheci e me apoiaram durante o mestrado.

“Quando o homem se põe como medida de todas as coisas, converte-se em escravo de sua própria finitude."

Papa João Paulo II

"Não existe verdadeira inteligência sem bondade."

Ludwig van Beethoven

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANÁLISE DE AGRUPAMENTOS FCM UTILIZANDO PROCESSAMENTO PARALELO

Marta Vidal Modenesi

Março / 2008

Orientadores: Alexandre Gonçalves Evsukoff
Myrian Christina de Aragão Costa

Programa: Engenharia Civil

Este trabalho propõe o uso de paralelismo para tratar o problema da análise de agrupamentos no processamento de grandes volumes de dados integrando seus dois maiores desafios: o cálculo dos centros de agrupamentos feito pelo algoritmo de particionamento Fuzzy c- means e a seleção do melhor padrão para os dados através da avaliação dos agrupamentos pelo índice de validação de partições PBM.

Duas estratégias são propostas: a do paralelismo pela divisão do conjunto de dados e a do paralelismo pela divisão do conjunto de partições. Neste último caso, uma estratégia de balanceamento de carga modelada pelo problema das múltiplas mochilas é acrescentada ao algoritmo para lidar com os diferentes pesos das partições. Sua implementação é feita por uma heurística que incorpora as restrições relativas à análise de agrupamentos ao algoritmo *first-fit decreasing*.

Testes realizados com dados sintéticos e um estudo de caso com dados sísmicos da Petrobras referentes ao campo de óleo e gás de Marlim são apresentados com as análises de eficiência e escalabilidade do algoritmo paralelo de análise de agrupamento Fuzzy c-means em diversos cenários.

Abstract of Dissertação presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

FCM CLUSTER ANALYSIS
USING PARALLEL PROCESSING

Marta Vidal Modenesi

March / 2008

Advisors: Alexandre Gonçalves Evsukoff
Myrian Christina de Aragão Costa

Department: Civil Engineering

This work proposes the use of parallelism to deal with the cluster analysis problem in the processing of large volumes of data by the integration of its two major challenges: the clusters calculations done by the Fuzzy c-means algorithm and the selection of the best pattern for the data through the clusters evaluation by the PBM clusters validity index.

Two approaches are proposed: parallelism by the data set division and parallelism by the partitions set division. In this last case, a load balance strategy modeled by the multiple knapsacks problem is added to the algorithm to deal with the different weights partitions. Its implementation is made by a heuristic that incorporates the restrictions related to the cluster analysis to the first-fit decreasing algorithm.

Tests made with synthetic data and a seismic data case study from Petrobras referring to the Marlim oil and gas field are presented with the Fuzzy c-means cluster analysis parallel algorithm efficiency and scalability analysis over diverse scenarios.

1 Introdução

A imensa quantidade de dados gerados por empresas em diversos setores, tais como, telecomunicações, seguros, exploração de gás e óleo, dentre outras, direcionou os algoritmos de mineração de dados para as implementações paralelas [1][2].

Diversas implementações paralelas de mineração de dados são usadas ou experimentadas, tanto em máquinas com memória distribuída como compartilhada [3] e também em ambientes de grid [4]. Todos os principais algoritmos de mineração de dados têm sido investigados, tais como: indução por árvore de decisão [5], classificadores baseados em regras fuzzy [6][7], redes neurais [8][9], mineração por regras de associação [10][11] e de agrupamentos [12][13].

A análise de agrupamentos é a classificação não supervisionada dos dados em grupos e é uma das abordagens de mineração de dados mais fortemente consumidora de recursos computacionais, devido ao cálculo das distâncias de registros a centros de agrupamentos.

Muitos algoritmos paralelos e distribuídos têm sido recentemente estudados na área da análise de agrupamentos e usados em diversas aplicações que processam grandes volumes de dados, incluindo classificação de imagem, recuperação de documentos, segmentação de consumidores, dentre outros.

Os algoritmos de agrupamento geralmente seguem abordagem hierárquica ou particional [16]. Na abordagem particional, dois principais problemas de otimização são endereçados: encontrar os padrões nos dados, ou seja, determinar qual o número de agrupamentos que emergem dos dados representando um padrão natural para eles, e, calcular estes padrões, o que significa, calcular os centros de cada agrupamento. Para o segundo problema, o cálculo dos centros dos agrupamentos, algoritmos de otimização iterativos como o k-means [14] e suas variações, dentre elas o fuzzy c-means [19], são amplamente utilizados, sendo muito populares na comunidade de mineração de dados.

O algoritmo *fuzzy* c-means (FCM), generalizado a partir do algoritmo k-means por Bezdek no início dos anos oitenta [19], calcula partições chamadas nebulosas (*fuzzy* em inglês) devido ao fato de permitir que os mesmos dados possam pertencer a mais de um agrupamento, mas com diferentes valores de pertinência. A solução FCM é muito útil

em aplicações reais porque propõe limites flexíveis para as fronteiras dos agrupamentos onde a incerteza da classificação é levada em consideração.

No entanto, o problema mais desafiador da análise de agrupamentos é descobrir quais padrões são, de fato, válidos para o conjunto de dados. Isto, na prática, significa determinar quantos agrupamentos emergem dos dados. Em geral, a abordagem usada para essa investigação é a repetição do algoritmo de cálculo dos centros dos agrupamentos para diferentes partições, sendo cada resultado avaliado por um índice de validação, e o resultado final selecionado a partir das partições com os melhores índices.

Vários índices de validação de agrupamentos têm sido propostos na literatura nos últimos anos [20][21][22]. Índices de validação tentam responder a duas questões importantes na análise de agrupamentos: quantos agrupamentos estão realmente presentes nos dados e qual a qualidade da partição realizada. O conceito norteador dos índices de validação baseia-se na estrutura geométrica da partição, considerando que amostras dentro do mesmo agrupamento devem ser compactas e que agrupamentos diferentes devem estar bem separados.

O índice PBM recentemente proposto por Pakhira et al. [22], cujo nome é acrônimo para as iniciais dos nomes dos autores, é um índice de validação que investiga partições avaliando sua estrutura geométrica e se os agrupamentos gerados são bem definidos e separados. É um índice de maximização, o que significa dizer que quanto maior o índice PBM calculado, melhor é a qualidade da partição gerada.

Usualmente, implementações paralelas de algoritmos de agrupamento [12][13] somente consideram estratégias para distribuição do processo iterativo para encontrar os centros dos agrupamentos. Com o objetivo de tratar a questão da análise de agrupamentos na sua totalidade este trabalho propõe a integração do cálculo de um conjunto de partições com sua avaliação a fim de possibilitar eleger a melhor partição para os dados.

Esta proposta é implementada através de algoritmos onde estão integrados, tanto o cálculo dos centros de agrupamentos das partições pelo algoritmo FCM, quanto a avaliação das partições pelo índice PBM, cujos resultados são comparados para indicar qual é o padrão mais adequado para os dados. Duas abordagens distintas foram utilizadas: a primeira baseada na divisão do conjunto de dados pelos processadores e a segunda, na divisão do conjunto de partições.

No primeiro caso o conjunto de dados é dividido igualmente pelos processadores que calculam iterativamente o intervalo de partições e sua qualidade, sendo a consolidação dos resultados feita ao final por um dos processadores. Essa estratégia, na prática, resulta em um balanceamento de carga natural pela divisão dos dados, mas tem um alto custo de comunicação devido à necessidade de freqüente interação entre os processadores que calculam de forma síncrona as partições.

A fim de validar a eficiência desta solução, a segunda abordagem paralela foi implementada baseada na divisão das partições pelos processadores. Essa alternativa, entretanto, necessita de uma estratégia de balanceamento de carga para gerar uma divisão equilibrada do processamento paralelo das partições por processadores devido ao diferente peso computacional das partições. Essa questão pode ser compreendida como o problema de dividir o processamento de um conjunto de partições, com diferentes pesos computacionais, por processadores de mesma capacidade de processamento. É um caso prático do problema das múltiplas mochilas, uma variação do clássico problema da mochila [23], onde os processadores podem ser considerados como múltiplas mochilas computacionais, cujas idênticas capacidades devem ser preenchidas pelo processamento das partições que serão calculadas pelos algoritmos FCM [19] e PBM [22].

Para implementar o balanceamento de carga modelado pelo problema das múltiplas mochilas, uma heurística foi desenvolvida e acrescentada ao algoritmo paralelo. Essa heurística trata o problema da distribuição das partições por processadores como equivalente ao preenchimento do menor número de mochilas (filas de processamento) com itens (partições) de forma a respeitar uma dada capacidade computacional média calculada para o processamento. Este é um caso especial do problema das múltiplas mochilas, conhecido como o problema do empacotamento, para o qual há algoritmos que garantem soluções aproximadas, como o popular algoritmo *first-fit decreasing* [27]. A heurística implementada neste trabalho incorpora as restrições relativas à análise de agrupamentos ao algoritmo *first-fit decreasing* [27] resultando em um algoritmo específico para a análise de agrupamentos.

Finalmente, para avaliar a aplicabilidade e eficiência das abordagens paralelas de análise de agrupamentos FCM propostas neste trabalho no processamento de grandes volumes de dados foram realizados testes em diversos cenários com dados sintéticos e feito um estudo de caso com dados sísmicos da Petrobras referentes ao campo de óleo e

gás de Marlim. Os resultados com análises de desempenho e eficiência são apresentados permitindo a comparação das duas abordagens.

O trabalho segue a seguinte organização: os capítulos dois, três e quatro são referentes ao embasamento teórico do trabalho, com apresentação da questão da análise de agrupamentos e formulação dos algoritmos FCM e PBM, paradigmas de processamento paralelo e a formulação matemática do problema das múltiplas mochilas e do problema do empacotamento. No quinto capítulo são apresentados os algoritmos paralelos de análise de agrupamentos FCM por divisão de dados e por divisão das partições, este último modelado pelo problema das múltiplas mochilas.

No sexto capítulo os testes realizados usando dados sintéticos e reais com os dados sísmicos da Petrobras referentes ao campo de óleo e gás de Marlim são descritos e seus resultados apresentados, sendo feita uma comparação dos resultados obtidos com cada abordagem.

Por fim, no capítulo sete, são apresentadas as conclusões do trabalho e reflexões sobre trabalhos futuros.

2 Análise de Agrupamentos

A análise de agrupamentos é o estudo formal de algoritmos e métodos para agrupar ou classificar objetos físicos ou abstratos em classes de objetos similares. Funciona pela atribuição de itens em grupos distintos e homogêneos, chamados de agrupamentos, de forma que os objetos em cada agrupamento compartilham características comuns – freqüentemente medidas por proximidade, de acordo com alguma métrica definida.

Seu principal objetivo é encontrar uma estrutura conveniente, uma organização válida para os objetos, onde estes estejam organizados numa representação eficiente que caracteriza a população que está sendo amostrada.

É uma importante técnica de mineração de dados e vem sendo aplicada a diversas áreas científicas como engenharia, biologia, psicologia, medicina, marketing, processamento de imagens, sensoriamento remoto e processamento sísmico, dentre outros [2].

A análise de agrupamentos encontra-se no campo da classificação não supervisionada, que se caracteriza por realizar classificações apenas a partir das informações ou padrões presentes nos próprios objetos. Um conjunto de dados a ser analisado pela análise de agrupamentos contém apenas informações relativas a seus próprios atributos, ou seja, o conjunto de dados $T = \{(x(t), t = 1..N)\}$ contém em cada registro somente a informação representada pelo seu vetor de atributos $x(t) = (x_1(t),..., x_p(t)) \in X^p$. É aplicável a situações onde o conhecimento sobre a relação entre as observações e suas classes não é conhecido e utiliza técnicas e algoritmos específicos para este fim.

Um desafio da análise de agrupamentos é que não existe uma definição precisa do que é um agrupamento. Não é fácil encontrar um critério que traduza a intuitiva noção de agrupamento através de uma formulação matemática. Esta definição é dependente dos parâmetros do problema que está sendo analisado. Conforme o tipo de informação e o objetivo da análise, agrupamentos podem ter formas e tamanhos arbitrários em espaços multidimensionais e o número dos agrupamentos pode ser diferente. O número de possíveis formas de organizar os dados é enorme até para a avaliação de um pequeno número de padrões.

Portanto, para encontrar uma organização válida para os dados, a análise de agrupamentos deve realizar duas tarefas distintas: identificar qual é o número de agrupamentos adequado para representar um determinado conjunto de dados e calcular esses agrupamentos a partir das informações nos próprios dados.

Na prática, o que se faz é calcular diferentes agrupamentos e avaliar qual deles representa melhor um conjunto de dados. Diferentes técnicas são usadas em cada etapa: algoritmos de agrupamento são usados para a etapa de cálculo dos agrupamentos e algoritmos de cálculo de índices de validação de partições são usados para avaliar a qualidade dos agrupamentos gerados.

Os algoritmos que implementam métodos de classificação não-supervisionada são desenvolvidos a partir de técnicas de estatística, redes neurais, algoritmos genéticos e conjuntos nebulosos (*fuzzy*). O processo de organizar coleções de padrões, usualmente representados por vetores de características, gerou diferentes e variadas técnicas para representar os dados, medir a similaridade entre seus elementos e para representar os agrupamentos, mas apesar do grande número de técnicas existentes, os algoritmos de agrupamento mais utilizados se dividem basicamente em dois grupos: hierárquicos e particionais. Os algoritmos hierárquicos classificam os dados em uma hierarquia de grupos e os particionais organizam os dados classificando-os em grupos formados por similaridades.

Um agrupamento hierárquico é uma seqüência de partições na qual cada partição é aninhada dentro da próxima partição na seqüência [15]. O agrupamento hierárquico gera uma estrutura em forma de árvore aonde pontos vão sendo aninhados em grupos de acordo com alguma métrica, que geralmente representa algum tipo de distância entre os pontos. A representação de um agrupamento hierárquico permite um entendimento bastante claro da formação dos agrupamentos. A forma mais comum de representação do agrupamento hierárquico é o dendograma (Figura 1), que é um tipo especial de estrutura em árvore que consiste em camadas de nós e linhas que conectam os nós aos pontos compreendidos em cada nó, representando grupos que estão aninhados uns dentro dos outros.

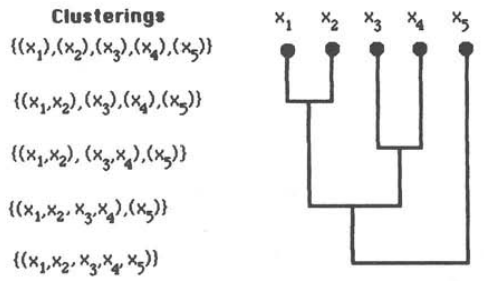

Figura 1 - Exemplo de dendograma

Cortar um dendograma horizontalmente determina os grupos aos quais estão atribuídos os dados, ou seja, os agrupamentos.

Os métodos de agrupamento particionais, como o próprio nome indica, geram um particionamento dos dados, ou seja, os dados são divididos em um determinado número de grupos segundo algum padrão natural existente neles.

Métodos de agrupamento hierárquicos geralmente supõem apenas a matriz de proximidade entre os objetos, enquanto que métodos particionais esperam trabalhar com dados dispostos em uma matriz de padrões.

Técnicas hierárquicas são populares em biologia, ciências sociais e comportamentais porque estas necessitam construir taxonomias. Técnicas particionais são usadas freqüentemente em aplicações de engenharia aonde partições únicas são importantes. Métodos de agrupamento particionais são especialmente apropriados para a representação eficiente e para compreensão de grandes bases de dados. A visualização do agrupamento dos dados nos dendogramas é impraticável quando se tem um número muito grande de padrões.

Os algoritmos de agrupamento particionais são aprofundados na próxima seção, pois o algoritmo de agrupamento utilizado neste trabalho se encontra nesta categoria. Em seguida é apresentado o índice de validação de partições PBM também utilizado neste trabalho.

2.1 Algoritmos de Agrupamento Particionais

A idéia de grupos de objetos ou dados implica em que padrões em um mesmo agrupamento são mais similares entre si do que padrões em agrupamentos distintos.

Um item ou objeto é geralmente descrito como um conjunto de medições ou de relações entre este objeto e outros objetos. Um agrupamento é um aglomerado de pontos no espaço vetorial de forma que as distâncias entre quaisquer dois pontos em um agrupamento são menores do que a distância entre qualquer ponto do agrupamento e outro ponto que não pertence a ele [28] (Figura 2). Portanto, o problema do agrupamento particional pode ser estabelecido da seguinte forma: *Dados n padrões em um espaço d dimensional, determinar um particionamento de padrões em K agrupamentos, de forma que os padrões em um agrupamento são mais similares uns aos outros do que aos padrões fora do agrupamento*.

Neste caso o termo padrão é empregado representando as ocorrências dos objetos dentro do modelo que o particionamento está tentando estabelecer. As coordenadas identificam os objetos pela sua localização no espaço vetorial.

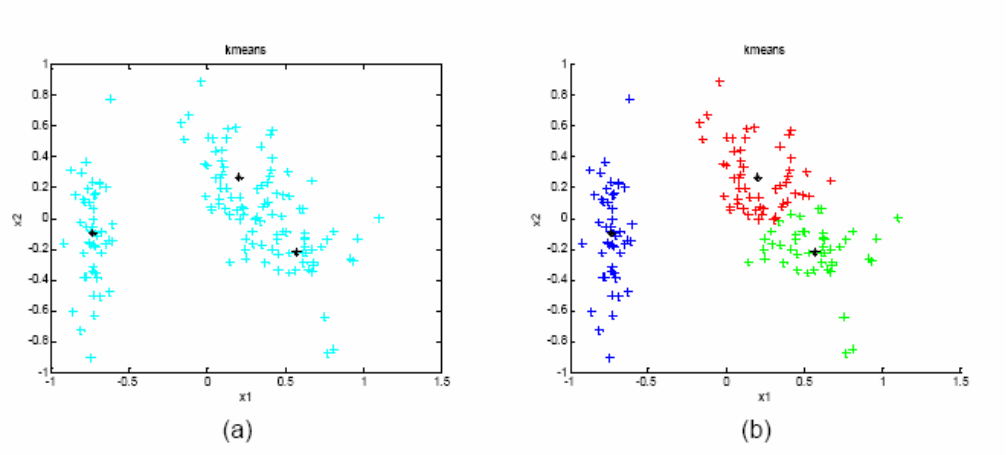

Figura 2 – Possível organização de agrupamentos.

Os algoritmos de particionamento têm como objetivo encontrar a matriz de coordenadas dos centros de agrupamentos $W = [w_i,...,w_K\}$, onde cada coluna $w_i \in X^p$ define as coordenadas do centro de agrupamento representativo da classe w_i, $i = 1...K$.

Para encontrar e definir um agrupamento, algoritmos particionais calculam o centro das coordenadas de cada agrupamento e atribuem cada registro a um determinado agrupamento a partir de alguma métrica de distância desta ocorrência em relação ao centro do agrupamento.

O modelo gerado pelo algoritmo de particionamento procura identificar grupos que caracterizem um significado sobre a ocorrência dos padrões, onde seja posível associar um significado semântico ao fato de um padrão pertencer ou não a um agrupamento. Por exemplo: a análise de um conjunto de atributos referentes a peso e altura de crianças. Um algoritmo de particionamento calcula três agrupamentos para os dados. Relacionando cada grupo a um padrão no mundo real, podemos identificar que os registros agrupados onde há grandes valores para altura e valores baixos para o peso caracterizam o grupo das crianças magras, o agrupamento referente aos registros onde ocorrem altos valores de peso e pequenos valores de altura caracterizam as crianças gordas e o agrupamento com os valores de peso e altura intermediários caracterizam as crianças que não são gordas nem magras.

A partir de exemplos reais fica mais fácil perceber que os métodos de particionamento visam otimizar um determinado critério de custo definido em função da distância dos registros em relação aos centros de agrupamentos:

$$J(W) = \frac{1}{N} \sum_{t=1...N} \sum_{x(t) \in w_i} d(x(t), w_i)^2$$

onde $d(x(t), w_i)$ é, geralmente, a distância euclidiana, embora outras métricas, tais como a quadrado da distância euclidiana, a distância de Mahalanobis, a distância de Minkowski e medidas de similaridade criadas especialmente para variáveis em diferentes escalas de medida, dentre outras, também possam e sejam muitas vezes empregadas, conforme a natureza do problema que esteja sendo analisado.

De uma forma geral, a distância Euclidiana pode ser calculada como:

$$d(x(t), w_i)^2 = [x(t) - w_i] P [x(t) - w_i]^T$$

onde a matriz de ponderação P, que define os pesos de uns atributos em relação aos outros, é geralmente a identidade I^P.

Os métodos baseados em distância são afetados pela diferença de escala entre valores dos atributos, sendo necessário normalizar os atributos que estejam em escalas diferentes para que os resultados sejam coerentes.

2.1.1 Critério de Agrupamento do Erro Quadrático

A estratégia de agrupamento é baseada no critério do erro quadrático. O objetivo geral do método é obter a partição que minimiza o erro quadrático, dado um número fixo de agrupamentos.

Suponhamos que um dado conjunto de N padrões em d dimensões tenha sido particionado em K agrupamentos $\{w_1, w_2, ..., w_K\}$ de forma que o agrupamento w_K tem n_K padrões e que cada padrão está em exatamente um grupo, de forma que:

$$\sum_{k=1}^{K} n_k = n$$

O vetor médio do agrupamento w_k é definido como o centróide do agrupamento, ou

$$m^{(k)} = (1/n_k)\sum_{i=1}^{n_k} x_i^{(k)}$$

onde x_i^k é o $i$ésimo padrão pertencendo ao agrupamento w_K. O erro quadrático para o agrupamento w_K é a soma do quadrado das distâncias euclidianas entre cada padrão em w_K e o seu centro de agrupamento é m^k. Este erro quadrático é também chamado de variação intra-grupo.

$$e_k^2 = \sum_{i=1}^{n_k} (x_i^{(k)} - m^k)^T (x_i^{(k)} - m^{(k)})$$

O objetivo de método de agrupamento pelo erro quadrático é encontrar um particionamento dos dados com K agrupamentos que minimiza E_K^2. O método pelo

critério do erro quadrático enxerga os centróides como protótipos dos agrupamentos. O erro representa os desvios dos padrões em relação aos centróides.

2.1.2 O algoritmo K-means

Um grande número de algoritmos de particionamento está presente na literatura. Os mais populares são o K-means [14] [15] e seu grande número de variações, como o Fuzzy c-means [19], o *hill-climbing* [17] e o *density-based* DBSCAN [18], dentre outros.

O algoritmo K-means é um dos mais simples algoritmos de classificação não supervisionada que resolve bem o problema de agrupamentos. Segue uma forma simples e fácil de classificar um conjunto de dados através de um certo número de agrupamentos fixados a priori.

A idéia central é definir k centróides, que são os pontos centrais dos agrupamentos, um para cada agrupamento. A forma como estes centróides são inicialmente estabelecidos em diferentes localizações pode causar diferentes resultados. A melhor opção é colocá-los, tanto quanto possível, distantes uns dos outros.

O algoritmo começa pela definição do número de agrupamentos e pela atribuição aleatória dos centros de agrupamentos iniciais. É calculado um particionamento inicial para os K centros de agrupamentos iniciais. A partir dessa etapa, o algoritmo entra em um ciclo iterativo onde atribui cada ponto do conjunto de dados ao centróide mais próximo naquela iteração. Ao final de cada iteração calcula novos centróides pela média de todos os pontos atribuídos em cada agrupamento. O algoritmo termina quando é minimizada a função de critério, neste caso a função de erro quadrático:

$$J(W) = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - w_j \right\|^2$$

onde $\left\| x_i^{(j)} - w_j \right\|^2$ é escolhida como uma medida de distância entre um ponto do conjunto de dados $x_i^{(j)}$ e o centro do agrupamento w_j, é um indicador da distância dos *n* pontos aos seus respectivos centros de agrupamentos.

O algoritmo K-means pode ser descrito pelas seguintes etapas:

1. Definir K , sendo $K \geq 2$ e inicializar as coordenadas dos centros de agrupamentos aleatoriamente;
2. Atribuir cada registro $x(t)$ ao agrupamento w_i mais próximo de acordo com a menor distância $d(x(t), w_i)$ do registro ao centro do agrupamento da iteração corrente:

$$x(t) \in w_i \leftrightarrow d(x(t), w_i) \leq d(x(t), w_j), \forall j \neq i$$

3. Atualizar a matriz de centros de agrupamentos pela média das coordenadas dos pontos em cada agrupamento:

$$w_i = \frac{\sum_{t=1...N} \chi_i(x(t).x(t)}{\sum_{t=1...N} \chi_i(x(t))}$$

4. Se a diferença entre os centros de agrupamentos não mudar mais ou ficar menor do que o erro mínimo desejado parar; caso contrário voltar à etapa 2.

A função $\chi_i(x(t))$ é a função característica do agrupamento w_i definida no passo 2 do algoritmo como:

$$\chi_i(x(t)) = \begin{cases} 1 \rightarrow x(t) \in w_i \\ 0 \rightarrow x(t) \in w_i \end{cases}$$

Embora Selim e Ismail [31] tenham provado rigorosamente a convergência do algoritmo K-means e seus derivados, o algoritmo não encontra necessariamente a melhor configuração de agrupamento dos dados, que corresponde ao mínimo global da função objetiva, pois o algoritmo é bastante sensível a seleção aleatória dos centros dos agrupamentos iniciais, podendo convergir para mínimos locais dependendo dos valores atribuídos inicialmente.

2.1.3 O algoritmo Fuzzy c-means (FCM)

O *Fuzzy* c-means (FCM) é um algoritmo de análise de agrupamentos particional que utilizando lógica nebulosa permite que um objeto pertença a mais de um agrupamento com diferentes graus de pertinência. Proposto por Bezdek em 1981 [19] como uma generalização do K-means, o método é freqüentemente utilizado na área de reconhecimento de padrões.

Nos agrupamentos nebulosos (*fuzzy*), a associação dos elementos aos agrupamentos é ponderada através de um conjunto de graus de pertinência. Esses valores de pertinência indicam a força de associação entre o elemento e um agrupamento específico.

O particionamento fuzzy é um processo de atribuição de níveis de pertinência dos elementos a agrupamentos.

Graficamente podemos ter uma noção bem intuitiva do particionamento FCM através do exemplo da Figura 3. No exemplo, duas funções distintas classificam um conjunto de pontos nos grupos A e B. No primeiro caso, a função de classificação sobre uma descontinuidade que define a fronteira entre os conjuntos A e B. No segundo caso, a função de classificação apresenta uma região de transição entre os dois conjuntos e os pontos situados nessa região pertencem ao conjunto A em um determinado grau e ao conjunto B em outro. Este segundo exemplo, retrata o comportamento de uma classificação *fuzzy*, que considerada os pontos localizados na região intermediária entre dois agrupamentos com valores de pertinência intermediários aos dois agrupamentos.

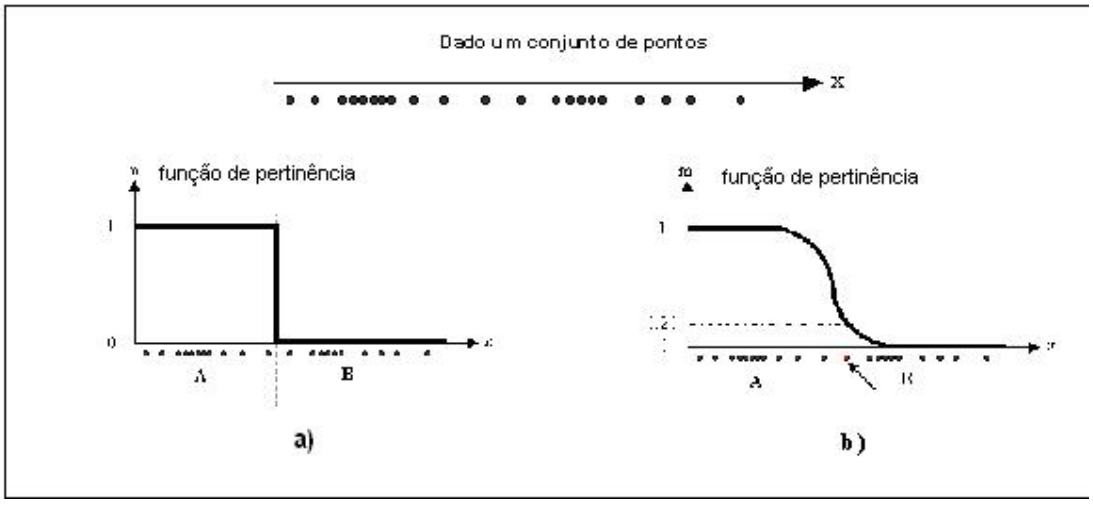

Figura 3 - Exemplificação gráfica do FCM.

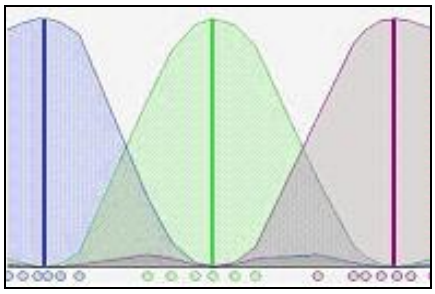

Figura 4 - Representação gráfica bidimensional de uma partição FCM em três agrupamentos.

O grau de pertinência FCM de um ponto a um agrupamento é inversamente proporcional a distância do ponto ao centro do agrupamento. Quanto mais próximo do centro do agrupamento, maior a pertinência do ponto ao agrupamento, portanto, pontos localizados na fronteira de um agrupamento fazem parte do agrupamento com grau de pertinência menor do que pontos localizados no centro do agrupamento.

Sendo $u_i(x)$ a pertinência de x ao agrupamento w_i, temos que:

$$u_i(x) = \frac{1}{d(w_i, x)}$$

No FCM, cada agrupamento é representado pelo seu centróide, sendo que o centróide de um agrupamento é a média de todos os pontos, ponderados pelo seu grau de pertinência ao agrupamento:

$$w_k = \frac{\sum_x u_k(x)x}{\sum_x u_k(x)}$$

O grau de pertinência de um registro ou objeto a um agrupamento toma valores no intervalo [0,1]. A pertinência de todos os registros a todos os agrupamentos forma a matriz de partição. É a matriz de partição que contém o valor de pertinência de cada registro a cada agrupamento. A matriz de partição tem as seguintes propriedades:

a) A soma dos valores de pertinência de cada registro a cada agrupamento deve ser 1, ou seja, a soma dos valores de cada linha deve ser 1;

b) Cada registro deve pertencer a pelo menos um agrupamento e nenhum agrupamento pode conter todos os registros, o que quer dizer que a soma de cada coluna da matriz de partição deve ser maior que zero e menor que o número de linhas da matriz.

O algoritmo FCM pode ser definido da seguinte maneira:

Seja o conjunto $T = \{(\mathbf{x}(t)),\ t = 1..N\}$, onde cada amostra é uma coordenada contida no vetor $\mathbf{x}(t) \in R^p$, o algoritmo tem como objetivo realizar uma partição *fuzzy* do domínio em K agrupamentos de centros $\{C_1 \ldots C_K\}$, onde cada agrupamento C_i é representado pelo vetor de coordenadas de seu centro $\mathbf{w}_i \in R^p$.

O algoritmo FCM calcula as coordenadas dos centros de agrupamentos minimizando a função objetiva J definida como:

$$J(m, \mathbf{W}) = \sum_{t=1..N} \sum_{i=1..K} u_i(t)^m \, d(\mathbf{x}(t), \mathbf{w}_i)^2 \qquad (1)$$

onde $m > 1$. A variável *m* é usualmente conhecida como o "parâmetro de fuzzificação". Sua função é ajustar o efeito da função de pertinência. A medida de distância $d(\mathbf{x}(t), \mathbf{w}_i)$ geralmente utilizada é a distância euclidiana, que mede a distância da amostra $\mathbf{x}(t)$ ao centro de agrupamento $\mathbf{w}_i$.

A pertinência de todas as amostras a todos os agrupamentos define uma matriz de partição como:

$$U = \begin{bmatrix} u_1(1) & \cdots & u_K(1) \\ \vdots & \ddots & \vdots \\ u_1(N) & \cdots & u_K(N) \end{bmatrix}. \qquad (2)$$

A matriz de partição é definida pela seguinte equação:

$$\forall \mathbf{x}(t) \in T, \sum_{i=1..K} u_i(t) = 1 \qquad (3)$$

O algoritmo FCM calcula interativamente as coordenadas dos centros de agrupamento a partir de uma estimativa prévia da matriz de partição de forma que:

$$\mathbf{w}_i = \frac{\sum_{t=1..N} u_i(t)^m . \mathbf{x}(t)}{\sum_{t=1..N} u_i(t)^m}. \quad (4)$$

A matriz de partição é atualizada pela seguinte equação como em 5:

$$u_i(t) = \frac{1}{\sum_{j=1..K} \left(\frac{d(\mathbf{x}(t), \mathbf{w}_i)}{d(\mathbf{x}(t), \mathbf{w}_j)} \right)^{\frac{2}{(m-1)}}}. \quad (5)$$

O algoritmo FCM é descrito a seguir:

1) Seja $m > 1$, $K \geq 2$. Inicialize as coordenadas dos centros de agrupamentos aleatoriamente e a matriz de partição como em (5).

2) Para cada agrupamento i onde $(2 \leq i \leq K)$ atualize a coordenada do centro do agrupamento como em (4).

3) Para cada amostra t $(1 \leq t \leq N)$ e para cada agrupamento i $(2 \leq i \leq K)$, atualize a matriz de partição como em (5).

4) Se a diferença entre os centros de agrupamentos não mudar mais ou ficar menor do que o erro mínimo desejado pare; caso contrário volte à etapa 2.

O resultado de um algoritmo FCM não inclui apenas a partição dos dados, mas também a informação do grau de pertinência de um registro ou objeto a cada agrupamento.

Considerações

As principais vantagens deste algoritmo são sua simplicidade e velocidade que permitem que ele processe satisfatoriamente em grandes bases de dados. Sua limitação é que ele não alcança o mesmo resultado a cada processamento, pois os agrupamentos resultantes dependem dos valores aleatórios iniciais. A forma como os centróides são inicialmente estabelecidos em diferentes localizações pode causar diferentes resultados.

Embora possa ser provado que o algoritmo sempre termina, pois converge para um mínimo local [31], o algoritmo FCM não encontra necessariamente a melhor configuração de agrupamento dos dados, o que corresponderia ao mínimo global da função objetiva, pois o algoritmo depende fortemente da inicialização aleatória dos centros de agrupamentos e do peso para ponderação, podendo na prática retornar uma solução bem pior que o mínimo global. Como o algoritmo é extremamente rápido, uma forma muito usada para garantir o melhor resultado é executá-lo diversas vezes com diferentes centróides iniciais e escolher o melhor resultado, em uma tentativa de assegurar que foi alcançado o mínimo global da função objetiva, ou seja, a melhor partição do conjunto.

O tempo gasto por um algoritmo *Fuzzy c-means* é proporcional ao produto do número de padrões e do número de agrupamentos por iteração, e de fato, tem sido observado que o número de iterações é usualmente bem menor que o número de objetos. Recentemente, entretanto, Arthur and Vassilvitskii [30] mostraram que existem alguns conjuntos de dados nos quais o algoritmo leva tempo superpolinomial para convergir.

Na prática, para a grande maioria dos conjuntos de dados, os algoritmos do tipo *Fuzzy* c-means convergem rapidamente.

2.2 Validação de Partições e o índice PBM

Embora o algoritmo FCM sempre atinja a convergência, nem sempre alcança o mínimo global da função de critério, pois seu resultado depende fortemente dos valores de inicialização, que geralmente são atribuídos aleatoriamente. Além disso, no caso de bases de dados para as quais não se tenha conhecimento de quais padrões podem ser válidos para os dados, podem estar sendo realizadas partições que não são as mais representativas dos agrupamentos naturais existentes nos dados. Portanto, para que possa ser feita uma correta investigação dos padrões existentes nos dados, é necessário avaliar que partições são de fato significativas. Para que o processo de análise de

agrupamentos possa alcançar resultados práticos, devem ser respondidas as seguintes perguntas:

1 - Qual é a partição mais adequada para os dados? Ou seja, quantos agrupamentos estão presentes nos dados?

2 - A partição gerada pelo algoritmo é de boa qualidade, ou seja, na partição surgem agrupamentos distintos e bem caracterizados?

Com esse objetivo existem na literatura diversos trabalhos sobre índices de validação de partições cujos objetivos são exatamente investigar a qualidade das partições geradas e realizar comparações entre diversas partições para selecionar as mais adequadas para os dados.

Nos últimos anos, diversas métricas de validação de agrupamentos têm sido propostas na literatura [20][21][22].

A idéia geral presente na maioria dos índices de validação de agrupamentos é baseada em uma visão geométrica da partição, de forma que uma boa partição dos dados apresenta grupos compactos onde agrupamentos diferentes devem estar bem separados. No caso da análise de agrupamentos em que um valor de pertinência é associado a cada registro, o grau de superposição entre grupos também deve ser levado em conta, de forma que a melhor partição é aquela que apresentar menor superposição.

Recentemente, o índice de validação de agrupamentos PBM [22] tem obtido um bom resultado em um grande número de aplicações.

O índice PBM [22] é usado para avaliar o número de agrupamentos ou subconjuntos formados a partir de um conjunto de dados. É definido como o produto de três fatores, conforme em (6), dos quais a maximização assegura que a partição tem um pequeno número de grupos compactos com grande separação entre pelo menos dois deles.

O índice PBM é definido como a seguir:

$$PBM(K) = \left(\frac{1}{K} \cdot \frac{E_1}{E_K} . D_K \right)^2 \qquad (6)$$

onde K é o número de agrupamentos.

O fator E_1 é a soma das distâncias de cada amostra ao centro geométrico de todas as amostras $\mathbf{w}_0$. Esse fator não depende do número de agrupamentos e é calculado como a seguir:

$$E_1 = \sum_{t=1..N} d(\mathbf{x}(t), \mathbf{w}_0). \tag{7}$$

O fator E_K é a soma das distâncias intra–grupos de K agrupamentos, ponderada pelo valor de pertinência correspondente de cada amostra ao agrupamento:

$$E_K = \sum_{t=1..N} \sum_{i=1..K} u_i(t) d(\mathbf{x}(t), \mathbf{w}_i)^2 \tag{8}$$

E D_K representa a máxima separação de cada par de agrupamentos:

$$D_K = \max_{i,j=1..K} \left(d(\mathbf{w}_i, \mathbf{w}_j) \right) . \tag{9}$$

O procedimento para calcular o índice PBM pode ser descrito com a seguir:

1) Selecione o número máximo de agrupamentos M ;

2) Calcule o fator E_1 (7) ;

3) Para $K = 2$ a $K = M$, faça:

   a. Rode o algoritmo FCM;

   b. Calcule os fatores E_K (8) and D_K (9);

   c. Calcule o índice $PBM(K)$ (6).

4) Selecione o melhor número de agrupamentos K^* como:

$$K^* = \arg\max\left(PBM(K)\right) \tag{10}$$

Quanto maior o índice PBM, melhor a partição. Assim como outros índices, o índice PBM é um índice de otimização, de forma que para se obter a melhor partição, deve-se executar o algoritmo para diversos valores de K e escolher aquele que resultar no maior valor de índice PBM.

O problema mais desafiador da análise de agrupamentos é descobrir quais padrões são, de fato, válidos para o conjunto de dados, o que na prática, significa determinar quantos agrupamentos existem de fato nos dados. Em geral, a abordagem usada para

essa investigação é a repetição do algoritmo de cálculo dos centros dos agrupamentos para diferentes partições, sendo cada resultado avaliado por um índice de validação, e o resultado final selecionado a partir das partições com os melhores índices.

Os algoritmos de análise de agrupamentos FCM propostos neste trabalho, integram o algoritmo de agrupamentos FCM e o do índice de validação de partições PBM para responder ao problema da análise de agrupamentos, com o primeiro calculando os centros dos agrupamentos e o segundo, avaliando a qualidade da partição gerada.

No entanto, para que o processo da identificação da melhor partição para os dados possa ocorrer, muitas vezes um grande número de partições deve ser examinado. Os algoritmos propostos neste trabalho foram elaborados então, para realizar o cálculo e avaliação de um conjunto de partições de forma automática. A idéia era a de oferecer uma ferramenta que investigasse um grande número de partições para tentar garantir a seleção de uma partição realmente representativa para os dados. Como com o aumento do número de partições há também um aumento proporcional do tempo de processamento do algoritmo, a fim de que os algoritmos tivessem boa aplicabilidade foi percebida a necessidade de que fossem desenvolvidos algoritmos paralelos para que, mesmo quando fosse avaliado um grande número de partições, o tempo de processamento pudesse ser controlado com o aumento do número de processadores.

A seguir, no capítulo 3, são analisadas questões relativas ao projeto de algoritmos paralelos para que possam ser compreendidas as decisões tomadas na paralelização da análise de agrupamentos FCM.

3 Processamento Paralelo

Ao longo de toda a história da ciência da computação, novas aplicações demandam computadores cada vez mais velozes e com maior capacidade de tratar grandes volumes de informações. Resultado natural da própria utilização da computação, a aquisição de grandes volumes de dados e a sempre crescente utilização de aplicações computacionais faz com que empresas e instituições científicas acumulem grandes bases de dados e desenvolvam inúmeros programas e sistemas para automatizar seus negócios.

A comparação do custo versus desempenho nos computadores seriais ao longo das últimas décadas mostra a saturação desta curva após um determinado ponto. O processamento de determinadas aplicações científicas ou empresarias, principalmente as que envolvem o processamento de grandes volumes de informações, necessitam de grande tempo de processamento, sendo muitas vezes impossível carregar o volume de informações necessárias na memória de um único processador.

Uma alternativa largamente adotada tanto em termos de custo quanto desempenho para elevar o poder computacional, é a idéia de conectar microprocessadores formando sistemas paralelos.

Um dos atuais desafios para os sistemas de mineração de dados é poder processar grandes volumes de dados [12]. Há diversas abordagens para o problema na literatura. Algumas abordagens sugerem amostrar os dados [32][33]. No entanto, há indicações de que nos casos de classificação a utilização de um conjunto de treinamento completo aumenta a acurácia enquanto que sua redução leva a *over-fitting* [34]. Uma abordagem para processar grandes volumes de dados é a da paralelização do processamento.

Nos algoritmos de mineração de dados baseados em agrupamentos, que lidam tanto com o cálculo intensivo das medidas de similaridade dos objetos quanto com o número de objetos e atributos a analisar, há também necessidade de intensa computação. Os algoritmos tratam o cálculo das distâncias entre padrões e centros de agrupamentos em um ciclo iterativo, o que, no processamento de grandes volumes de dados causa grande impacto no tempo de processamento.

Novamente, uma abordagem viável para redução do tempo de processamento nesses casos é a implementação de algoritmos paralelos de análise de agrupamento escaláveis para um grande número de processadores.

Diversos trabalhos neste sentido foram realizados sem, no entanto, esgotar a questão da análise de agrupamentos, pois tratam apenas da versão paralela do algoritmo de agrupamento sem abordar a questão da descoberta do número correto de padrões para classificar o problema sendo pesquisado.

Este trabalho propõe a utilização de processamento paralelo para implementar uma solução para a questão da análise de agrupamentos, tratando tanto do cálculo dos agrupamentos quanto da descoberta da melhor partição para os dados. O presente trabalho apresenta uma solução completa e eficiente para o problema da análise de agrupamentos.

3.1 Arquiteturas de Computadores em Processamento Paralelo

Computadores podem ser basicamente classificados quanto a sua arquitetura, segundo a taxonomia de Flynn, em quatro categorias: Single Instruction Single Data Stream (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD) e Multiple Instruction Multiple Data (MIMD) [40].

A categoria SISD ou modelo de computador seqüencial realiza uma única seqüência de instruções e opera em uma única seqüência de dados de cada vez. Esta é a arquitetura mais comumente usada de computadores.

Os computadores com arquitetura SIMD, MISD e MIMD pertencem às categorias de computadores para processamento paralelo.

A arquitetura SIMD possui uma única unidade de controle que despachas as instruções para cada processador no sistema. Os processadores executam o mesmo conjunto de instruções em um conjunto distinto de dados.

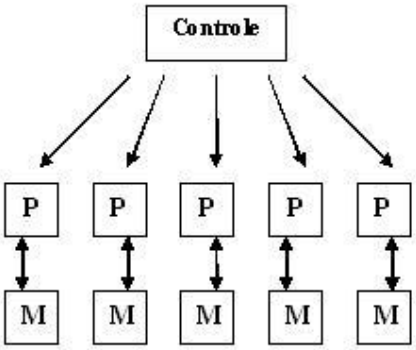

Figura 5 – Arquitetura SIMD.

A arquitetura SIMD é capaz de aplicar o mesmo fluxo de instruções a múltiplos fluxos de dados simultaneamente como mostrado na Figura 5. Para problemas onde há necessidade de paralelismo dos dados, esta arquitetura é perfeitamente adequada para alcançar taxas de processamento muito altas, pois os dados podem ser divididos em pedaços e múltiplas unidades de instrução podem operar sobre eles ao mesmo tempo. A arquitetura SIMD é síncrona e determinística.

A arquitetura MIMD tem uma unidade de controle em cada unidade de processamento onde cada processador é capaz de executar um programa diferente de forma independente dos demais. Cada processador executa seu conjunto de instruções em seu conjunto de dados.

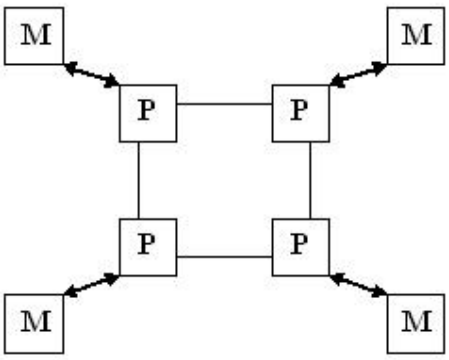

Figura 6 – Arquitetura MIMD.

Esta arquitetura pode operar de modo síncrono ou assíncrono, determinístico ou não, e é adequada para paralelismo em diversos níveis de granularidade, tais como: blocagem, laços, subrotina, paralelismo de múltiplas instruções e paralelismo de programas.

Entre os exemplos de máquinas MIMD podem ser citados os clusters com estações de PCs usando PVM/MPI.

Dependendo do tipo de estrutura de memória do sistema, a arquitetura MIMD é classificada em memória compartilhada ou distribuída. Na arquitetura de memória compartilhada, a mesma memória é acessada por múltiplos processos e a sincronização é alcançada através de tarefas de controle lendo e escrevendo na memória compartilhada.

Na arquitetura de memória distribuída cada processador tem a sua própria memória local e o sincronismo é alcançado pela comunicação entre processadores através da rede de interconexão. A maior preocupação na arquitetura de memória distribuída é a decomposição dos dados, ou seja, como distribuir dados entre processadores (memórias locais) para minimizar a comunicação entre processadores. Sistemas de memória distribuída podem ser vistos como modelos de passagem de mensagem.

Nos sistemas de passagem de mensagens ("message-passing") os programas se comunicam por explicitamente enviar e receber dados uns dos outros. O processo de envio e recebimento dos dados é especificado e controlado pela aplicação através da utilização de bibliotecas de passagem de mensagem, por exemplo, MPI, PVM, etc...

3.2 Projeto de Algoritmos Paralelos

A criação de um programa paralelo eficiente depende da identificação das partes computacionais de um problema que podem ser executadas simultaneamente.

Diferentes modelos de programação são utilizados para obter diferentes soluções. As características de cada aplicação orientam a seleção de uma arquitetura de hardware paralela e o tipo de paralelismo adequado.

A programação paralela para sistemas com memória distribuída tem aspectos diferentes da programação em sistemas com memória compartilhada. Na programação

com memória compartilhada, os programadores vêem seus programas como um conjunto de processos acessando um grupo central de variáveis compartilhadas.

Na programação para memória distribuída os programadores vêem seus programas como um conjunto de processos com variáveis locais privadas e com a habilidade de enviar e receber dados entre processos através da passagem de mensagens.

Em grande parte dos casos de processamento paralelo há mais de um algoritmo (ou pelo menos variações de um mesmo algoritmo) para resolver o problema. Elaborar um algoritmo paralelo é uma atividade que pode ser desenvolvida de diferentes formas e para a qual não há uma receita. Entretanto, o processo de projetar algoritmos paralelos pode se beneficiar de uma metodologia que permita analisar a variedade de opções de paralelismo disponíveis e que forneça mecanismos para avaliação dessas alternativas.

Uma forma de projetar bons algoritmos paralelos é utilizar uma metodologia de desenvolvimento que favoreça a utilização correta do paralelismo e que trate de questões inerentes ao aproveitamento desse tipo de processamento. O uso de uma metodologia permite ao programador se concentrar nas oportunidades de paralelismo relativas ao problema analisado, adiando aspectos relativos à escolha da arquitetura paralela para mais adiante no projeto [35].

Muito difundida no projeto de algoritmos paralelos é a metodologia de Foster [36], que propõe quatro passos para, a partir da especificação do problema, se obter um algoritmo paralelo que o resolva. A metodologia de Foster, cujo acrônimo é PCAM, como apresentado na Figura 7 ajuda a considerar as questões envolvidas na elaboração de um algoritmo paralelo objetivando obter a melhor eficiência para o processo.

Seus quatros passos para o projeto de algoritmos paralelos propostos são:

- Particionamento;
- Comunicação;
- Aglomeração e
- Mapeamento.

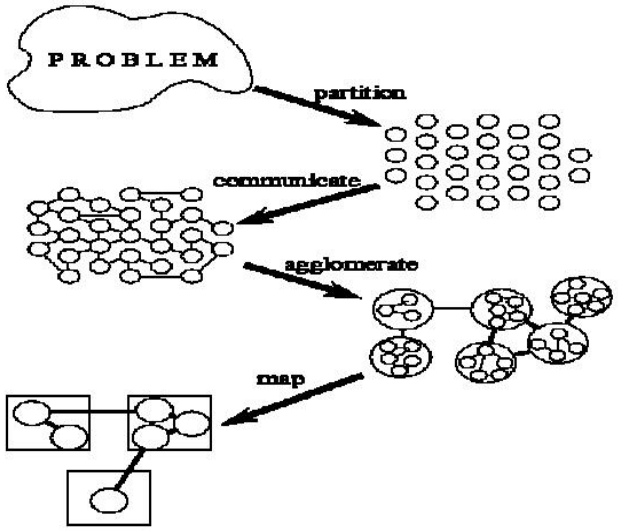

Figura 7 - Metodologia PACM para elaboração de algoritmos paralelos.

Nesta metodologia as duas primeiras etapas se preocupam com aspectos de concorrência e escalabilidade do algoritmo, enquanto que as duas etapas finais se direcionam a aspectos relativos a localização e desempenho.

A seguir são descritas as etapas da metodologia:

Particionamento: O objetivo principal desta etapa é encontrar oportunidades de paralelização. Deve ser realizada a decomposição do problema pela definição das várias tarefas que o compõe. Há uma decomposição das atividades computacionais e dos dados que serão processados.

O ideal nesta etapa é que seja encontrado um ponto de equilíbrio entre a utilização do paralelismo e do trabalho de gerenciamento das tarefas paralelas. O trabalho realizado de maneira paralela deve ser significantemente maior do que o esforço para gerenciar os diversos processamentos. Ou seja, as tarefas de comunicação e sincronização não podem ser mais complexas que as tarefas de computação. Se isto ocorrer, a execução em paralelo certamente será mais ineficiente do que a seqüencial.

São duas as técnicas de particionamento empregadas:

- Decomposição de Domínio (ou Dados): nesta técnica são particionados os dados associados ao problema e depois é feita a divisão do processamento, associando aos dados (divididos de forma disjunta) e o

conjunto de operações pertinentes. Algumas operações poderão envolver dados de mais de uma tarefa, sendo que, neste caso, será necessário realizar comunicação entre as tarefas, a fim de estabelecer algum grau de sincronismo entre as mesmas;

- Decomposição Funcional: neste caso, a ênfase é dada na divisão da computação envolvida no problema. Primeiro, procura-se particionar o processamento em tarefas disjuntas, depois é feita a análise dos dados necessários a cada tarefa. Caso os dados possam ser divididos em conjuntos disjuntos, a decomposição está completa. Caso contrário, pode-se tentar outro particionamento do processamento, replicação dos dados ou compartilhamento dos mesmos. Caso a sobreposição de dados seja grande, pode ser mais indicado o emprego da decomposição de domínio.

Comunicação: O particionamento do problema é feito com a intenção de executar as tarefas de forma concorrente e, se possível, de forma independente. Contudo, muitas vezes a computação associada a uma tarefa envolve dados associados a uma ou várias outras tarefas, o que requer o estabelecimento da comunicação entre as mesmas. Esta fase tem por objetivo analisar o fluxo de informações e de coordenação entre as tarefas, definindo uma estrutura de comunicação. A natureza do problema e o método de decomposição determinam o padrão de comunicação entre as tarefas cooperativas de um programa paralelo.

Foster divide os modelos de comunicação em quatro categorias (ortogonais):

- Local/Global: na comunicação local cada tarefa se comunica com um pequeno grupo de outras tarefas (suas "vizinhas"), enquanto que no modelo global as tarefas podem comunicar-se arbitrariamente;

- Estruturado/Não Estruturado: no modelo estruturado as tarefas formam uma estrutura regular (e.g. árvore), já no modelo não estruturado elas formam grafos arbitrários;

- Estático/Dinâmico: no modelo estático a comunicação se dá sempre entre as mesmas tarefas, enquanto que no modelo dinâmico a comunicação não possui parceiros definidos, dependendo dos valores calculados em tempo de execução;

- Síncrono/Assíncrono: há, no modelo síncrono, uma coordenação entre as tarefas comunicantes, enquanto no modelo assíncrono não existe coordenação na comunicação.

Aglomeração: Nesta etapa, são aglomeradas as tarefas que podem ser executadas em seqüência por cada processador. As tarefas e a estrutura de comunicação definida nos passos anteriores são avaliadas em termos de requisitos de desempenho e custos de implementação. Se houver necessidade, tarefas podem ser combinadas ou aglomeradas em tarefas maiores a fim de reduzir o custo de implementação ou aumentar o desempenho. Pelo mesmo motivo pode-se efetuar a replicação de dados e / ou processamento.

Mapeamento: Nesta etapa é realizada a atribuição de cada tarefa a um processador, de forma que seja maximizada a utilização dos recursos do sistema, e sejam minimizados os custos de comunicação, de forma a minimizar o tempo de execução.

O mapeamento pode ser feito estaticamente (em tempo de compilação / antes da execução do programa) ou dinamicamente durante o processamento por métodos de balanceamento de carga.

3.3 Paradigmas de programação paralela

Face ao tipo de paralelismo inerente ao problema e aos recursos computacionais disponíveis, os algoritmos paralelos acabam por apresentar semelhanças em suas estruturas de controle. Com base nestas semelhanças surgiram os paradigmas (ou modelos) de programação paralela que são classes de algoritmos paralelos que possuem a mesma estrutura de controle [37].

É amplamente aceita a idéia de que aplicações paralelas podem ser classificadas em paradigmas de programação bem definidos e na prática há um número relativamente pequeno de paradigmas suportando a maioria dos programas paralelos [37].

A escolha do paradigma é determinada pelos recursos de paralelismo disponíveis e pelo tipo de paralelismo inerente ao problema. Os recursos de computação podem

definir o nível de granularidade que pode ser eficientemente suportado pelo sistema. O tipo de paralelismo reflete a estrutura da aplicação ou dos dados e ambos os tipos podem existir em partes diferentes da mesma aplicação.

O paralelismo que surge da estrutura da aplicação é chamado paralelismo funcional. Neste caso, partes diferentes do programa realizam tarefas diferentes de forma concorrente e cooperativa. Mas o paralelismo também pode ser encontrado na estrutura dos dados. Esse tipo de paralelismo permite a execução de processos paralelos com operações idênticas, mas em diferentes partes dos dados.

Na literatura, há várias propostas diferentes de classificação dos paradigmas. Buyya e Silva [38] analisam várias delas e apresentam o que consideram ser um super conjunto de paradigmas, quais sejam:

- Master/Slave (Task-Farming): é o modelo das aplicações trivialmente paralelas que implementa a execução independente de partes de uma mesma aplicação;

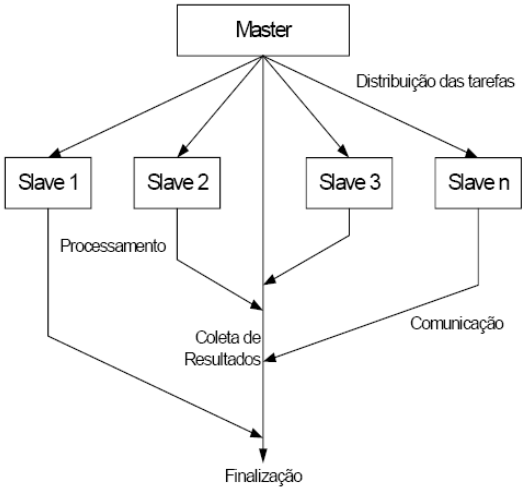

Figura 8 – Esquema representativo do paradigma de programação paralela Mestre/Escravo.

- Single-Program Multiple-Data (SPMD): segundo este paradigma, todos os processos executam o mesmo código, mas em partes diferentes dos dados. Os processos necessitam comunicar-se a fim de manter o sincronismo da execução;

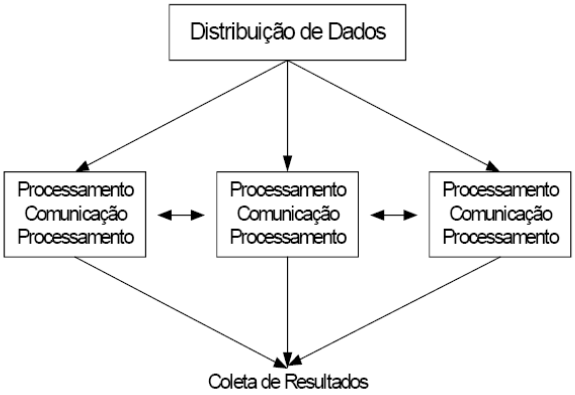

Figura 9 - Esquema representativo do paradigma de programação SPMD.

- Pipelining de Dados: neste paradigma o problema é dividido em uma seqüência de passos. Cada passo vira um processo e é executado por um EP diferente;

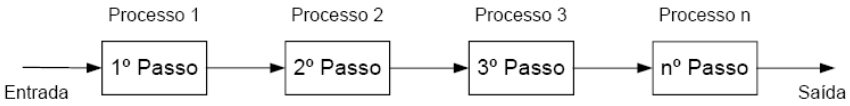

Figura 10 - Esquema representativo do paradigma de programação paralela Pipelining de Dados.

- Dividir e Conquistar: em termos gerais, neste modelo o problema é dividido em subproblemas, os quais são resolvidos independentemente e cujos resultados são então combinados para formar o resultado final. Enquanto que no paradigma Master/Slave a divisão do problema é feita somente no Master, neste paradigma o subproblema, uma vez atribuído, pode continuar a se dividir;

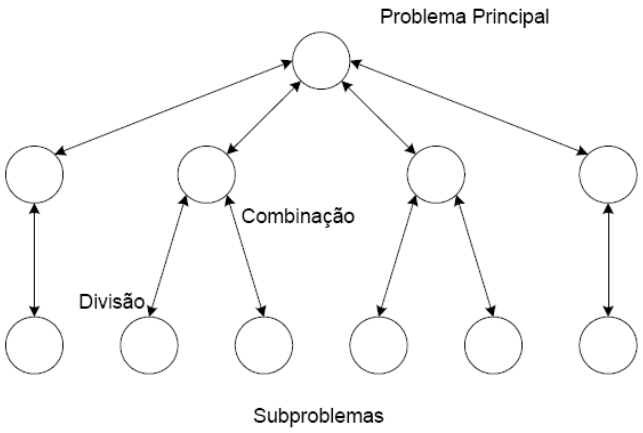

Figura 11 - Esquema representativo do paradigma de programação Dividir e Conquistar.

- Paralelismo Especulativo: empregado quando existem complexas dependências entre os dados, o que dificulta a utilização de outros paradigmas. Nestes casos o problema é dividido em partes pequenas e é utilizada especulação ou execução otimista a fim de facilitar o paralelismo.

- Modelos Híbridos: As fronteiras entre paradigmas podem ser tênues em alguns casos e, em algumas aplicações, pode haver a necessidade de misturar elementos de diferentes paradigmas. Métodos híbridos que incluem mais de um paradigma básico são usualmente observados em algumas aplicações paralelas de larga escala. Essas são situações onde faz sentido misturar paralelismo de tarefas e dados simultaneamente ou em partes diferentes do programa.

3.4 Aplicações Paralelas SPMD

Em uma aplicação SPMD o mesmo código é executado sobre diferentes conjuntos de dados nos diversos processadores. Uma aplicação SPMD é, portanto, caracterizada

por processar um conjunto de dados utilizando um código comum a todos os processamentos.

De acordo com [39], a classe de aplicações paralelas SPMD é definida por aplicações projetadas segundo o modelo de paralelismo de dados para serem executadas em arquiteturas MIMD (Multiple Instruction stream, Multiple Data stream).

Platino [42] apresenta uma caracterização das aplicações SPMD segundo o tipo de comunicação existente entre os processadores ao longo de sua execução: aplicações formadas por tarefas sem dependência e que não se comunicam, aplicações que se comunicam em pontos de sincronismo bem definidos e aquelas que não apresentam regularidade de comunicação. São elas:

3.4.1 Aplicações SPMD sem Dependência

Nesta categoria, os processadores não se comunicam ao longo da execução das tarefas que compõem a aplicação. A execução de uma tarefa não depende de dados resultantes da execução das demais. Portanto, as tarefas são completamente independentes umas das outras e podem ser executadas em qualquer processador e em qualquer ordem. Em Foster [36], os problemas que podem ser tratados por esse tipo de estratégia são denominados embaraçosamente paralelos. Esse tipo de estratégia é muito comum devido à sua facilidade de codificação e à sua aplicabilidade em vários problemas [43]. Sempre que um problema pode ser resolvido através de um loop de iterações independentes, uma aplicação SPMD sem dependência pode ser adotada.

A execução de uma aplicação SPMD desta classe se divide em três fases. Na primeira fase, executa-se a distribuição inicial das tarefas entre os processadores. Na segunda fase são executadas as tarefas que compõem a aplicação. A última fase representa a coleta e análise dos resultados de cada processador.

Tanto a distribuição das tarefas quanto a coleta dos resultados são fases comuns à maioria das aplicações SPMD e são realizadas por um dos processadores alocados para a execução da aplicação.

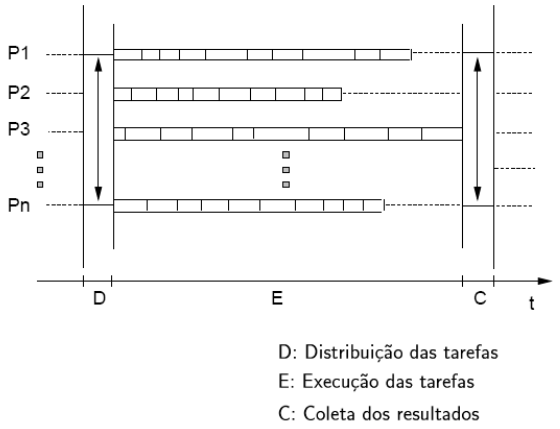

Figura 12 – Execução de uma aplicação SPMD sem dependência [42].

As aplicações SPMD desta classe possuem características que facilitam a utilização de estratégias de balanceamento de carga. Devido à independência entre as tarefas destas aplicações, as tarefas não realizadas podem ser, a qualquer momento, convenientemente transferidas de um processador a outro com o objetivo de equilibrar a carga da aplicação. Por se tratar de aplicações com paralelismo de dados, a transferência de uma tarefa é realizada transferindo-se o conjunto de dados que a define. Não há, portanto, transferência de processos, ativos ou não. Cabe estabelecer que, neste trabalho, a carga de uma aplicação SPMD sem dependência, em um determinado processador, em um instante qualquer, é considerada como o conjunto de tarefas que ainda devem ser executadas nesse processador.

3.4.2 Aplicações SPMD com Comunicação Síncrona

Nesta categoria, a comunicação entre os processadores é realizada em pontos de sincronismo bem definidos. Nestes pontos, todos os processadores se voltam para a troca de informações. Aplicações desta categoria são caracterizadas pela necessidade de troca periódica de informações. A execução das tarefas entre dois pontos de sincronismo corresponde, tipicamente, à execução de uma iteração da aplicação. A execução de uma iteração depende dos resultados da anterior. As aplicações SPMFD com comunicação síncrona podem ser também denominadas aplicações paralelas iterativas.

Em uma classificação apresentada em [43], uma aplicação SPMD com comunicação síncrona seria classificada como síncrona (*synchronous*) ou fracamente síncrona (*loosely synchronous*) dependendo, respectivamente, se os elementos de dados que definem as tarefas são equivalentes (isto, é, se a demanda computacional das diferentes tarefas é a mesma) ou não.

Na Figura 13 é ilustrada, genericamente, a execução das fases que compõem uma aplicação SPMD com comunicação síncrona. Em sua fase inicial, ocorre a distribuição das tarefas entre processadores. Em seguida, é executada uma seqüência de iterações. Cada iteração corresponde à execução das tarefas entre dois pontos de sincronismo, nos quais estas tarefas trocam as informações necessárias à continuidade do seu processamento. Na última fase, os resultados gerados por cada processador são coletados e analisados.

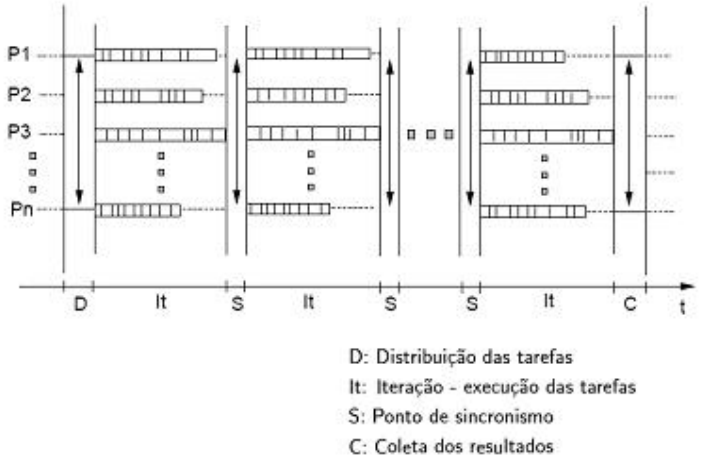

Figura 13 - Execução de uma aplicação SPMD com comunicação síncrona [42].

3.4.3 Aplicações SPMD com Comunicação Assíncrona

Nas aplicações desta categoria, a comunicação existente entre os processadores não ocorre em pontos de sincronismo bem definidos. O padrão de comunicação é dependente da aplicação. Muitas são caracterizadas pela comunicação irregular e imprevisível. Segundo [43], a utilização de aplicações SPMD com estas características é pouco comum. Isto ocorreria não por falta de problemas que poderiam se beneficiar de

estratégias SPMD com comunicação assíncrona, mas pela maior complexidade da construção deste tipo de aplicação.

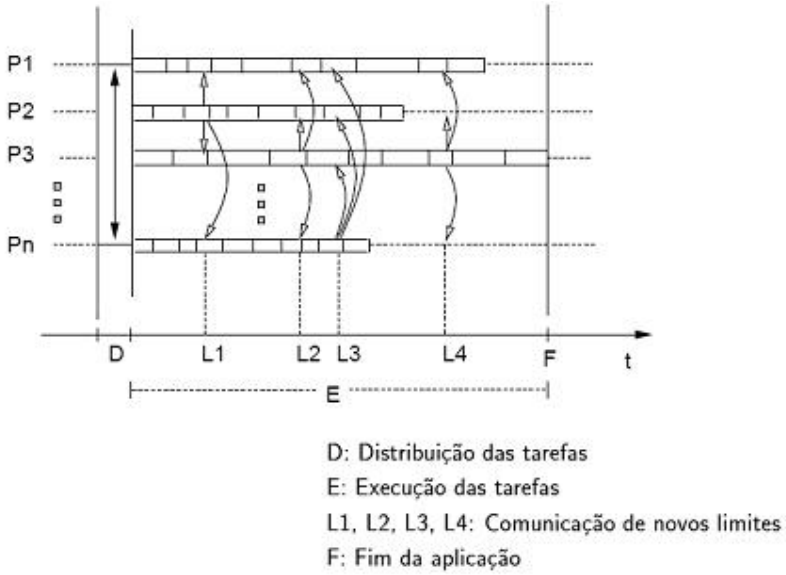

Figura 14 - Execução de uma aplicação SPMD com comunicação assíncrona [42].

A inexistência de um padrão de comunicação para esta categoria dificulta a identificação de uma forma genérica de utilização de estratégias de balanceamento de carga. A adoção de um algoritmo de balanceamento de carga deve ser estudada caso a caso.

3.5 Speedup e Eficiência

Uma característica fundamental da computação paralela trata-se do aumento de velocidade de processamento através da utilização do paralelismo. Duas medidas de desempenho muito utilizadas para a verificação da qualidade de algoritmos paralelos são o *speedup* e a eficiência.

O speedup de um algoritmo ou programa paralelo pode ser definido como o aumento de velocidade ou de desempenho observado quando se executa um determinado processo em *p* processadores em comparação à execução deste mesmo processo em um único processador.

$$Speedup = \frac{T_1}{T_p},$$

onde: T_1 = tempo de execução em um único processador e

T_p = tempo de execução em p processadores

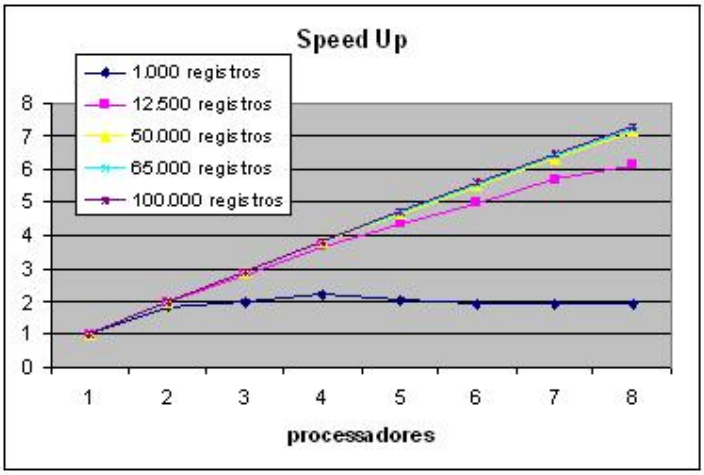

Figura 15 - Exemplo de gráfico de *speedup.*

Idealmente, o ganho de *speedup* deveria tender a *p*, que seria o seu valor ideal. Porém, três fatores podem ser citados que influenciam essa relação, gerando sobrecargas que diminuem o valor de *speedup* ideal: sobrecarga da comunicação entre os processadores, partes do código executável estritamente seqüenciais (*Amdahl's Law [41]*) e o nível de paralelismo utilizado.

A Figura 15 apresenta curvas de *speedup* obtidas na execução do algoritmo FCM paralelo por diferentes números de processadores processando diferentes arquivos.

Outra medida importante no processamento paralelo é a eficiência, que trata da relação entre o *speedup* e o número de processadores.

$$eficiência = \frac{Speedup}{p}$$

Na situação ideal onde o *speedup* fosse igual a = *p*, a eficiência seria máxima e teria valor igual a 1 (100%).

Lei de Amdahl [41]

Causado pelas sobrecargas nos programas paralelos, devido a fatores como sincronismo, comunicação e ativação de processos. Essa sobrecarga tende a diminuir com o aumento das tarefas (tarefas de maior complexidade).

3.6 Balanceamento de Carga

O balanceamento de carga é a política de distribuição do processamento de um algoritmo paralelo entre os processadores com o objetivo de alcançar a mais eficiente utilização do poder computacional existente. Essa distribuição pretende garantir a divisão igualitária da carga pelos processadores e o sincronismo de processamento entre eles. O ideal a alcançar é que todos os processadores processem a máxima carga possível trabalhando simultaneamente o máximo de tempo possível (Figura 16).

Todo projeto de um algoritmo paralelo deve levar em consideração a questão da distribuição da carga de trabalho pelos processadores. O balanceamento de carga é um fator crítico para se alcançar o desempenho ideal em aplicações com processamento paralelo.

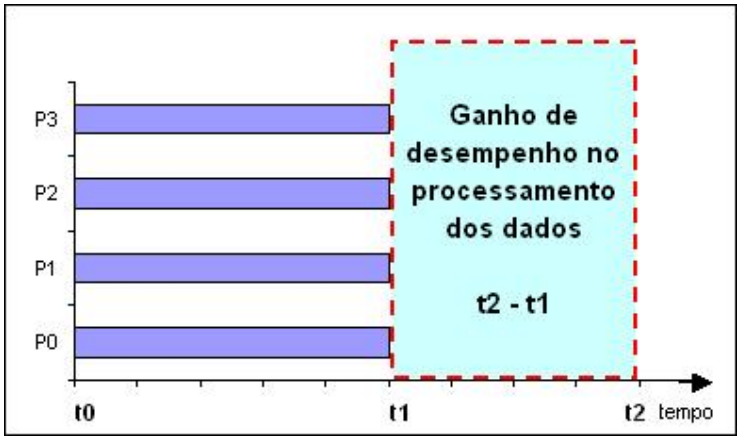

Figura 16 - Cenário ideal de balanceamento de dados.

Uma distribuição de carga inadequada pode acarretar os seguintes problemas:

- Mau desempenho do sistema;

- Má utilização dos recursos disponíveis (alguns processadores ficam ociosos enquanto outros são sobrecarregados);
- Prejuízo de outros requisitos não funcionais, tais como:
  - ✓ Confiabilidade;
  - ✓ Disponibilidade;
  - ✓ Escalabilidade e
  - ✓ Eficiência.

A questão do balanceamento de carga está intrinsecamente relacionada ao projeto do algoritmo paralelo. Uma forma adequada de desenvolver um algoritmo paralelo deve garantir uma boa distribuição da carga para o processamento. O balanceamento de carga é a parte da lógica do algoritmo responsável pela organização e distribuição do processamento entre os processadores. O planejamento adequado dessa distribuição garantirá ao processamento paralelo um balanceamento de carga eficiente gerando um algoritmo que escala adequadamente para grande número de processadores.

Em 1988, Casavant e Kuhl [44] propuseram a taxonomia para balanceamento de carga em sistemas distribuídos apresentada na Figura 17 que reflete os diversos tipos de algoritmos praticados nesta área.

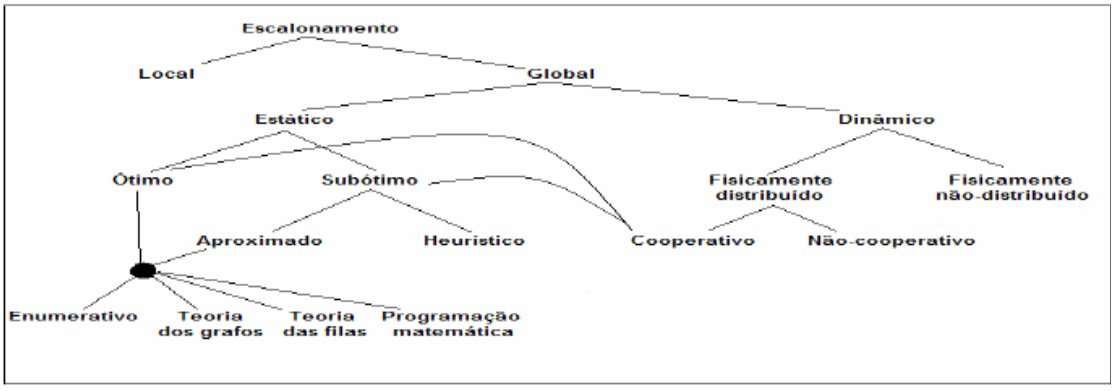

Figura 17 - Taxonomia de Casavant e Kuhl para balanceamento de carga em sistemas distribuídos (1988).

Deve-se observar que o balanceamento de carga para máquinas paralelas está dirigido para um ambiente de processamento homogêneo e mais estável do que aquele

existente para processamento distribuído. O fato do ambiente de execução nas máquinas paralelas ser homogêneo torna o processo de execução mais previsível.

Uma parte da taxonomia de [44], que, no entanto, pode ser aplicada adequadamente nesse contexto, é o dos algoritmos estáticos e dinâmicos. Em uma máquina paralela, podem-se classificar como estáticos os algoritmos que distribuem as tarefas única vez no início do processamento, e como dinâmicos, os algoritmos que distribuem as tarefas conforme a disponibilidade de processamento dos diversos nós durante a execução. O balanceamento dinâmico é tanto mais útil quanto mais heterogêneo e imprevisível for o ambiente de processamento, pois prevê o remanejamento do processamento entre os nós durante a execução de acordo com a situação de cada nó.

No entanto, em máquinas paralelas, algoritmos de balanceamento estático podem render bons resultados devido a previsibilidade desse ambiente.

Nos algoritmos paralelos de análise de agrupamentos FCM deste trabalho, a estratégia de balanceamento de carga estática foi utilizada. Esta estratégia foi considerada adequada nesse contexto devido a dois fatores: ao hardware paralelo disponível para o desenvolvimento dos algoritmos consistir em máquinas paralelas de processadores com idênticas capacidades computacionais, e, além disso, a própria natureza do algoritmo de agrupamentos FCM que permite a utilização de paralelismo segundo o paradigma SPMD. Nessa situação, o mesmo grupo de tarefas pode ser executado por diferentes processadores e a carga em cada processador pode ser calculada por uma estratégia de repartição dos dados pelos processadores no início do processamento.

A estratégia estática de balanceamento de carga adotada no projeto do algoritmo paralelo de análise de agrupamentos FCM foi modelada pelo problema das múltiplas mochilas, cujo embasamento teórico e aplicabilidade no contexto do balanceamento de carga em processamento paralelo são apresentados no próximo capítulo.

4 Balanceamento de Carga modelado pelo problema do Empacotamento

A questão do balanceamento de carga no processamento de tarefas em máquinas paralelas tem sido exaustivamente estudada ao longo da existência da ciência da computação.

O problema pode ser compreendido como a questão de determinar a alocação de n tarefas independentes, cada uma com um tempo de execução inteiro e positivo P_j para serem alocadas em m processadores paralelos idênticos, de forma que após a distribuição das tarefas aos processadores, a soma dos tempos das tarefas pertencentes ao processador com a maior carga entre todas (makespan) deve ser a mínima possível.

Outra maneira de formular o problema é dizer que seu objetivo é minimizar o tempo de finalização global das n tarefas alocadas aos m processadores supondo $n > m > 2$.

O problema da distribuição de tarefas a processadores paralelos é análogo a diversos problemas práticos existentes, tais como: a organização de linhas de produção e montagem de fábricas e indústrias, a distribuição de passageiros e clientes em filas de espera, o problema do empacotamento de cargas, a organização das rotas pelas transportadoras, a distribuição de energia elétrica, projeto de circuitos integrados, dentro inúmeros outros que surgem em diversas situações práticas.

Todos esses problemas são da área de otimização combinatória que estuda métodos para minimizar ou maximizar uma função objetiva relacionada às variáveis de decisão para um dado problema a partir de suas restrições associadas.

A solução dos problemas de otimização combinatória pode ser alcançada por enumeração exaustiva de todas as opções com a seleção posterior da melhor solução, mas na prática essa estratégia só é aplicável para coleções de itens muito pequenas. Muitos métodos são estudados com o objetivo de resolver estes problemas. Em situações práticas onde soluções ótimas têm um alto custo que as inviabiliza, muitas vezes, soluções aproximadas geram soluções aceitáveis que evitam ter que examinar todas as possíveis combinações e por conseqüência, conseguem tratar o problema de forma eficiente.

No caso específico deste trabalho, o problema de alocação de tarefas a processadores paralelos foi modelado pelo problema das múltiplas mochilas, mais especificamente por um de seus casos especiais, o problema do empacotamento, pois pode-se compreender os processadores paralelos como caixas (mochilas) de mesma capacidade computacional onde devem ser alocadas as n tarefas do processamento paralelo.

Nos tópicos seguintes são conceituados o problema da mochila e seus subproblemas e os algoritmos e métodos comumente utilizados para sua resolução no contexto do balanceamento de carga em máquinas paralelas.

4.1 Problema da Mochila

Podemos entender o *Problema da Mochila* como o desafio de encher uma mochila sem ultrapassar um determinado limite de peso, otimizando o valor do produto carregado. Reportado pela primeira vez na literatura por Dantzig [23], a formulação do problema é extremamente simples, porém sua solução é complexa.

O problema da Mochila é um problema popular muito estudado em otimização combinatória possuindo muitas aplicações práticas, tais como: investimento de capital, corte e empacotamento, carregamento de veículos, orçamento, dentre outros. Por esta razão, muitos casos especiais e generalizações têm sido examinados [25][26].

Comum a todas as versões é um conjunto de n itens, onde cada item toma valores no intervalo $1 \le j \le n$, e tem um valor associado v_j e peso p_j. O problema refere-se a escolha de alguns itens que resultem no valor máximo possível sem ultrapassar o peso máximo suportado pela mochila P. O intervalo $1 \le j \le n$ geralmente supõe que esses coeficientes são inteiros e quase sempre positivos.

O problema da mochila em sua forma básica pode ser formalizado como:

$$Maximize \sum_{j=1}^{n} v_j x_j \qquad (11)$$

$$s.t. \sum_{j=1}^{n} p_j x_j \le W,\ x_j \in \{0,1\}, 1 \le j \le n$$

Em situações reais, existem vários tipos de problemas da mochila, cada qual com restrições específicas que caracterizam problemas distintos com soluções próprias.

Dentre as generalizações diretas do problema da mochila, é de especial interesse para esse trabalho o problema das múltiplas mochilas e o problema do subconjunto. Ambos foram usados para modelar aspectos da distribuição de carga para o algoritmo paralelo de análise de agrupamentos FCM.

4.2 O Problema do Subconjunto

O problema possui as seguintes propriedades:

- É um problema de decisão;
- É um problema 0/1;
- Para cada item, o custo é igual ao valor: i.e. $p_j = v_j$.

A definição desse problema pode ser formalizada como a seguir:

$$Maximize \sum_{j=1}^{n} v_j x_j \qquad (12)$$

$$s.t. \sum_{j=1}^{n} v_j x_j \leq P,\ x_j \in \{0,1\}, 1 \leq j \leq n$$

O problema ocorre em situações onde um valor quantitativo deve ser alcançado, mas onde desvios negativos (perdas) devem ser minimizados e desvios positivos não são permitidos [25].

Uma forma de explicitar o problema pode ser a seguinte: dado um conjunto de inteiros e um inteiro *S*, existe algum subconjunto não vazio que some exatamente *S*? Valores e significados diferentes de *S* definem casos especiais do problema.

Um desses casos é denominado o problema da partição, onde deve-se decidir se existe alguma forma de dividir S em dois subconjuntos S_1 e S_2 de forma que a soma dos números em ambos os conjuntos sejam iguais. Os subconjuntos S_1 e S_2 devem formar uma divisão no sentido de serem disjuntos e cobrirem S.

O próprio problema da partição tem suas especializações, como por exemplo, o problema da terceira partição, no qual o conjunto S deve ser particionado em $|S|/3$ triplas, cada qual com mesma soma.

4.3 Problema das Múltiplas Mochilas

O problema da mochila se torna de interesse para o balanceamento de carga no processamento paralelo quando um conjunto de m mochilas de igual capacidade é considerado para distribuição do processamento.

Quando as capacidades de todas as m mochilas são iguais de forma que cada mochila tem capacidade $P_i = P/m$, onde P é a capacidade global do conjunto de mochilas e deseja-se preencher cada mochila sem ultrapassar P.

A solução para o problema é representada pela variável binária $x_{ij} \in \{0,1\}$ que atribui um item j a uma mochila i.

O problema das múltiplas mochilas pode ser formalizado como a seguir:

$$Maximize \sum_{i=1}^{m} \sum_{j=1}^{n} p_j x_{ij} \qquad (13)$$

$$s.t. \sum_{j=1}^{n} p_j x_{ij} \leq P_i, 1 \leq i \leq m$$

4.4 Problema da Soma dos Múltiplos Subconjuntos

O Problema da Soma dos Múltiplos Subconjuntos, Multiple Subset Sum Problem (MSSP) em inglês, é uma generalização do clássico problema do subconjunto, onde m mochilas devem ser preenchidas, ao invés de uma, e ao mesmo tempo, é um caso especial do problema das múltiplas mochilas, onde os valores dos itens são iguais aos custos e todas as mochilas têm a mesma capacidade [45].

No Problema da Soma dos Subconjuntos os itens de um conjunto são selecionados e empacotados em m mochilas (pacotes) idênticas, de tal forma que a soma dos pesos

dos itens em cada mochila não exceda a capacidade da mochila e a soma total dos itens empacotados seja a maior possível.

O Problema das Múltiplas Mochilas de mesmas capacidades pode ser definido da seguinte forma: dado um conjunto de *n* itens com custos e valores inteiros e positivos, um subconjunto deve ser selecionado de forma que itens neste subconjunto possam ser empacotados em *m* mochilas de mesma capacidade e de forma que o valor total de todos os itens nas mochilas seja máximo. Quando *m = 1* (MKP) o problema é reduzido ao clássico 0-1 problema da mochila [45].

$$Maximize \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij}$$

$$s.t.\ \sum_{j=1}^{n} p_j x_{ij} \le W_i, 1 \le i \le m$$

$$\sum_{i=1}^{m} x_{ij} \le 1,$$

$$1 \le j \le n,$$

$$1 \le x_{ij} \le n,$$

$$1 \le i \le n$$

Um caso especial desse problema é o problema do empacotamento, que consiste em alocar itens em mochilas utilizando o menor número possível de mochilas possíveis, onde as mochilas possuem a mesma capacidade e os itens tamanhos diferentes. No caso das mochilas terem apenas uma dimensão, temos o problema do empacotamento unidimensional exemplificado na Figura 18.

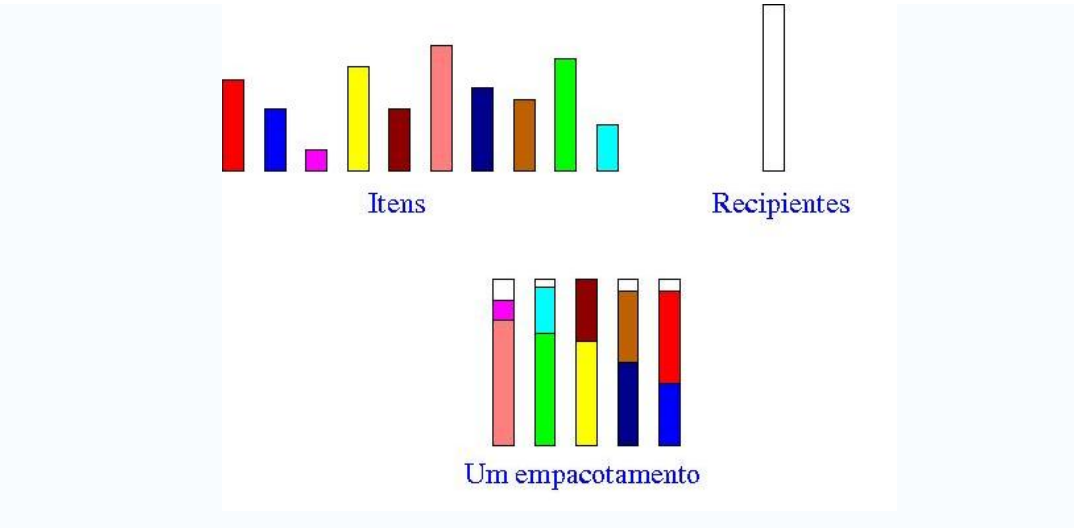

Figura 18 – Ilustração sobre empacotamento unidimensional de itens em recipientes [50].

4.5 Complexidade do Problema das Múltiplas Mochilas e solução adotada

Os problemas da mochila e subproblemas de corte e empacotamento têm sido exaustivamente tratados na literatura. Há uma grande diversidade de casos em que esses problemas podem aparecer na prática, mas em geral, os problemas têm uma complexidade estrutural associada e são na maioria das vezes NP-difíceis, ou seja, problemas cuja solução ótima não pode ser encontrada em tempo polinomial.

Devido aos problemas NP-difíceis exigirem a avaliação de um imenso número de alternativas para determinar sua solução exata, consistindo em uma classe de problemas intratáveis computacionalmente, a grande maioria dos trabalhos encontrados na literatura apresenta abordagens aproximadas para sua resolução.

As técnicas mais comumente empregadas na resolução computacional da maior parte dos diversos problemas de otimização são: programação linear e inteira, programação por restrições, algoritmos híbridos, heurísticas e algoritmos aproximativos. Em muitas situações práticas, estas técnicas fornecem soluções adequadas em um tempo computacional razoável.

Heurísticas são métodos para resolução de problemas que embora não rigorosos, aplicam o conhecimento humano a fim de obter uma solução satisfatória em

determinado contexto. São métodos aproximados (Figura 19), pois geralmente não garantem a solução ótima para o problema. Garantem, no entanto, encontrar uma solução adequada em um tempo computacionalmente razoável, através de privilegiar a eficiência do processo de busca, em detrimento da exploração de todas as alternativas (completude).

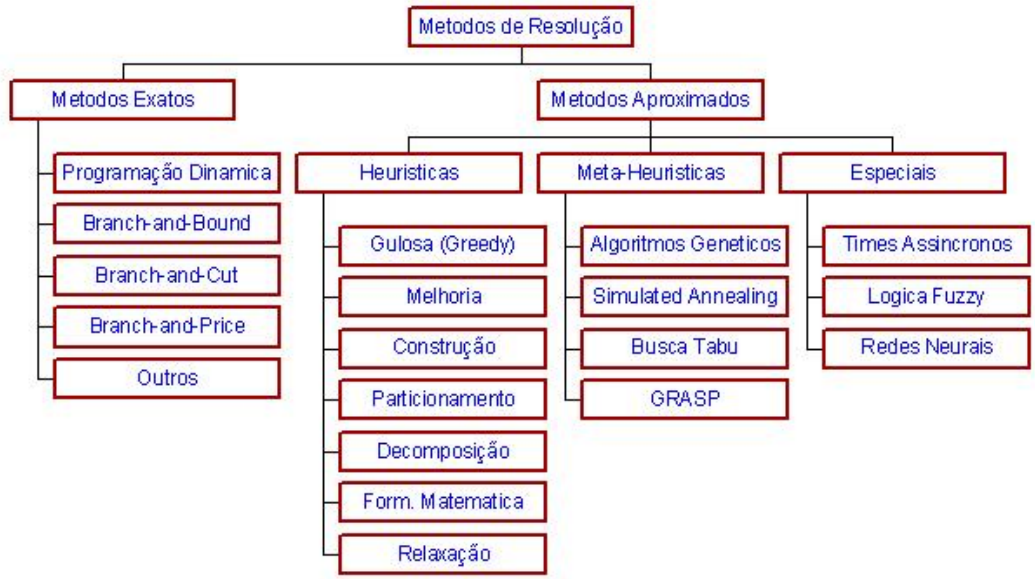

Figura 19 - Classificação das técnicas de resolução de problemas de otimização

Nos Métodos Heurísticos, não há garantia alguma a respeito da solução encontrada. Isto é, não há como saber se a solução obtida está "perto" ou "longe" da melhor solução possível em termos de qualidade. Contudo, há ocasiões em que essa noção de proximidade faz-se necessária. Por exemplo, pode ser necessário em uma solução que não precisa ser a melhor, mas deve ser, no máximo, 10% pior que a melhor solução possível. Nesses casos, uma solução que pode ser empregada a determinados problemas é a dos algoritmos aproximados.

Para o caso da distribuição de carga no processamento paralelo da análise de agrupamentos FCM apresentado neste trabalho a heurística proposta tem seu núcleo baseada em um algoritmo muito conhecido e muito difundido na solução do problema do empacotamento, o algoritmo o *primeiro que encaixa em ordem decrescente* (*First-Fit Decreasing order* – FFD em inglês).

O funcionamento do algoritmo FFD é muito simples e consiste em ordenar decrescentemente os itens a serem inseridos nas mochilas por tamanho, e então inserir cada item na primeira mochila da lista com espaço restante suficiente.

Os passos do algoritmo podem ser descritos como a seguir:

1.Para cada elemento do conjunto de itens

1.1 Encontrar alguma mochila com espaço suficiente e adicionar o elemento a ela

1.2. Se nenhuma mochila tiver espaço suficiente, usar uma nova para colocar o elemento

2. Retorne o número de mochilas usadas

Em 1973, D. Johnson [46] demonstrou que a estratégia adotada neste algoritmo nunca é sub ótima em mais que 22%. Também foi demonstrado que nenhum algoritmo para o problema do empacotamento pode garantir mais do que 22% de aproximação da solução ótima [47].

O algoritmo FFD é uma das mais simples heurísticas para solucionar o problema do empacotamento. Foi demonstrado que esse algoritmo não gasta mais do que 11/9 OPT + 1 mochilas (onde OPT é o número de mochilas dado pela solução ótima) [48]. Na estratégia FFD a etapa de ordenação pode ser relativamente custosa dependendo do conjunto de objetos que se vai ordenar, mas sem ela apenas seria alcançado o limite inferior de 17/10 OPT + 2 mochilas. Uma versão mais eficiente do FFD não usa mais do que 71/60 OPT + 1 mochilas [49][51].

Embora, essas estratégias simples de aproximação sejam, freqüentemente, bastante boas na resolução de muitos problemas práticos, algoritmos de aproximação eficientes têm sido demonstrados para resolver o problema do empacotamento dentro de qualquer percentagem fixada em relação à solução ótima para inputs grandes o suficiente (esquema de aproximação em tempo polinomial assintótico). Essa é característica específica do problema do empacotamento, pois muitos outros problemas NP-difíceis não permitem ser aproximados por nenhum fator.

4.6 Métodos para Máquinas Paralelas Idênticas

Máquinas paralelas são classificadas em: máquinas paralelas idênticas, máquinas paralelas uniformes e máquinas paralelas não-relacionadas. Para cada tipo de máquina paralela existem métodos estudados [52][53] para realização do seqüenciamento das tarefas. As máquinas paralelas são idênticas quando existe um conjunto único contendo os tempos de execução (ou finalização) das tarefas e estes tempos de execução permanecem constantes (idênticos) não importando para qual máquina uma tarefa é atribuída. Neste trabalho estamos lidando com arquiteturas paralelas onde os processadores possuem idênticas capacidades computacionais, configurando o caso das máquinas paralelas idênticas.

Imaginando a possibilidade da divisão de uma tarefa entre duas máquinas, tem-se a chamada preempção. Em uma situação de não-preempção, uma tarefa uma vez alocada a uma máquina deve permanecer nela até o final de sua execução, sem interrupções. Os métodos para solução do problema de seqüenciamento em máquinas paralelas, apresentados a seguir, referem-se exclusivamente à situação de não-preempção. Como função objetiva vai-se considerar a minimização do tempo máximo de finalização de todas as tarefas (*makespan*). Segundo classificação de três campos introduzida em [54], os problemas de seqüenciamento em máquinas paralelas idênticas, sem preempção, com o objetivo de minimizar o *makespan* são denominados $P||C_{max}$.

Um dos resultados iniciais no estudo de alocação das tarefas em máquinas idênticas foi a definição de um limitante para ser utilizado como referência, proposto em [55]. O valor do limitante pode ser calculado somando-se o tempo de processamento das *n* tarefas e dividindo-o por *m*, o número de máquinas. Trata-se de uma média aritmética, representando a média de carga para cada máquina, possuindo as vantagens e desvantagens desta medida estatística.

Graham [56] propôs algoritmos heurísticos com complexidade de tempo polinomial e teoremas demonstrando resultados nesta área. Seus algoritmos heurísticos são fáceis de implementar e muito rápidos na execução. Variações de seus algoritmos são encontradas com certa freqüência em adaptações para os problemas em máquinas paralelas uniformes e máquinas paralelas não-relacionadas.

O primeiro desses algoritmos foi chamado de *List Sheduling* (*LS*). Trata-se de uma heurística construtiva, cuja representação algorítmica é tão pequena que por vezes alguns autores referem-se ao algoritmo como regra heurística. O enunciado de *LS*

determina que as *n* tarefas sejam organizadas em uma lista, de ordem aleatória, sendo então alocadas, uma a uma, à máquina menos carregada. Cada vez que ocorrer empate na questão "máquina menos carregada", decide-se, arbitrariamente, para qual máquina atribuir a tarefa. Após a aplicação da regra heurística *LS*, pode-se esperar uma solução onde a carga com maior tempo de finalização em uma máquina seja, no máximo, duas vezes maior que a solução exata (ou ótima).

Posteriormente, Graham melhorou o desempenho do pior caso ao introduzir o algoritmo *LPT* (*Longest Processing Time first*). A regra de distribuição foi alterada permitindo que as tarefas com maior tempo de execução sejam alocadas antes. Condições de impasse (empates) também são resolvidas arbitrariamente.

A regra heurística (*LPT*) pode ser representada através do algoritmo:

Passo 1. Faça a ordenação decrescente das *n* tarefas.

Passo 2. Aplique a regra heurística *LS*.

As heurísticas de [56] aplicam ao problema do seqüenciamento de tarefas em processadores paralelos as mesmas etapas de resolução do algoritmo FFD para o problema do empacotamento.

Isso também acontece na heurística construtiva *MultiFit* (MF) [57] , que demonstra que é possível adaptar um método para resolução de um problema de otimização existente, o problema do empacotamento (FFD) para o problema de seqüenciamento em máquinas paralelas.

O algoritmo MultiFit [57] proposto para solucionar o problema *P||Cmax*, transforma o problema de sequenciamento de tarefas em um problema de empacotamento *(bin packing problem).* Essa abordagem considera cada processador como uma caixa a ser preenchida com as tarefas do processamento paralelo que correspondem aos itens a serem armazenados. O algoritmo de empacotamento utilizado para o MultiFit é o FFD *(First Fit Decreasing)* que faz uma pré-ordenação dos itens a serem alocados em ordem decrescente de seu tamanho. No momento da alocação, o item é designado à primeira caixa na qual existe espaço para alocação ou, então, uma nova caixa é criada. No algoritmo MultiFit necessita-se de um limitante superior e um

inferior para o tamanho das caixas utilizadas no algoritmo FFD. Definidos esse parâmetros, faz-se um processo iterativo, atualizando os limitantes inferiores e superiores da solução, considerando as seguintes situações: no caso de falta de espaço para a alocação das tarefas, aumenta-se o tamanho das caixas, caso contrário, diminui-se o tamanho dessas.

A implementação do balanceamento de carga na análise de agrupamentos FCM desenvolvida nesse trabalho foi implementada segundo uma heurística aproximada do algoritmo MultiFit. O limitante superior da distribuição é definido pelo maior valor entre os seguintes dois: pelo valor da maior carga ou pelo valor da carga média, onde esta última é calculada pela divisão do somatório das cargas pelo número de processadores disponíveis no momento do processamento. O limite inferior é o valor da carga média. Utilizou-se como critério de parada a igualdade entre os limitantes.

No capítulo seguinte os algoritmos paralelos de análise de agrupamentos FCM são apresentados e a respectiva heurística de balanceamento de carga utilizada.

5 Paralelismo aplicado à Análise de Agrupamentos

5.1 Algoritmo FCM Paralelo

O algoritmo de agrupamento FCM paralelo foi estudado por diversos pesquisadores a fim de processar grandes bases de dados, pois apresenta características bastante adequadas para a paralelização [12][13]. O maior custo computacional do algoritmo está no cálculo das distâncias dos pontos aos centros de agrupamentos, que é um processo iterativo. Apesar de ser proporcional ao número de registros e atributos do conjunto de dados, o custo do algoritmo deve-se principalmente ao número de padrões analisados. Em grandes volumes de informação o tempo de processamento pode ser inviável devido ao grande número de cálculos que devem ser feitos para processar todas as variáveis envolvidas no problema. Embora os computadores atuais tenham cada vez mais memória e capacidade de processamento, em alguns casos pode ser que uma base de dados não caiba inteira na memória de um único processador, ou que a análise de um problema seja extremamente demorada se for realizada por apenas um processador. O uso de paralelismo tem como objetivo reduzir o tempo total de processamento do algoritmo para viabilizar sua utilização em diversos hardwares e aplicações práticas.

5.2 Análise de Agrupamentos Paralela

O objetivo da análise de agrupamentos é determinar a melhor classificação para os dados através do reconhecimento de padrões presentes nos próprios dados. O algoritmo de agrupamentos FCM classifica um conjunto de dados dividindo-os em grupos que são representados pelas coordenadas dos seus centros. No entanto, o algoritmo FCM apenas classifica os dados em um número de grupos que lhe é informado *a priori*, mas não consegue dizer se a partição gerada representa uma boa classificação para os dados ou não.

A informação sobre a adequação da partição ao conjunto de dados transcende o objetivo do algoritmo FCM. Para analisar a qualidade das partições geradas deve ser usado um algoritmo de validação de partições como o do índice PBM [22]. O algoritmo PBM é capaz de calcular a qualidade de diferentes partições relativas a um conjunto de

dados e a partir desta avaliação pode-se selecionar qual partição representa melhor um padrão natural para o conjunto de dados.

Nesse trabalho, é proposta a combinação desses dois algoritmos para responder adequadamente ao problema da análise de agrupamentos. A solução reúne os algoritmos FCM e PBM, com o algoritmo de agrupamento FCM calculando as partições, e o algoritmo PBM identificando qual é a melhor partição para os dados.

Na prática, o algoritmo FCM processa cada partição calculando as distâncias de registros a centros de agrupamentos e para cada partição calculada pelo FCM é calculado o índice PBM.

Após o cálculo de todas as partições, a partição correspondente ao PBM máximo é escolhida como a melhor classificação para o conjunto de dados.

5.3 O algoritmo de Análise de Agrupamentos FCM Paralelo

Várias estratégias podem ser aplicadas para geração de um algoritmo paralelo. Através da utilização da metodologia PCAM [Projeto de Algoritmos Paralelos] pode-se avaliar as oportunidades de paralelismo referentes a um determinado problema.

No caso do algoritmo FCM paralelo um paradigma que surge como natural é o SPMD (*Single Program Multiple Data* - 3.4), pois o núcleo do algoritmo é o ciclo iterativo de cálculo das distâncias de registros a centros de agrupamentos. A estratégia de dividir os dados pelos diversos processadores permite que cada processador realize o ciclo iterativo de cálculo das distâncias de registros a centros de agrupamentos apenas sobre o seu subconjunto dos dados, diminuindo o tempo total do processamento. O paradigma funciona pela divisão dos dados em g partes onde $g = N / p$, sendo N o total de registros e p o número de processadores. Esta abordagem produz um algoritmo escalável para um número crescente de processadores, onde o tempo de processamento tende a diminuir na proporção em que são acrescentados novos processadores.

Na abordagem SPMD com divisão de dados aplicada à análise de agrupamentos FCM, o conjunto de partições é processado seqüencialmente pelos processadores que trabalham de forma cooperativa sincronamente. A estratégia utilizada neste caso é a da divisão do conjunto de dados igualmente por todos os processadores, para que eles, de

forma síncrona, trabalhem sobre cada subconjunto de dados. Este paradigma gera um algoritmo com carga naturalmente balanceada pela divisão em iguais proporções dos dados. As partições são processadas simultaneamente pelos processadores que calculam os centros de agrupamentos sobre o seu subconjunto de dados.

Esta abordagem pretende ganhar desempenho ao reduzir o número de registros a ser processado por cada processador.

Um problema prático que surge dessa abordagem é que deve ser realizada a correspondência entre os resultados dos processamentos de cada processador, ou seja, saber se os centros de agrupamentos referenciados por $c_1, c_2, ..., c_n$ para o subconjunto de dados **A** corresponde de fato aos centros de agrupamentos $c_1, c_2, ..., c_n$ dos outros subconjuntos processados (Figura 20).

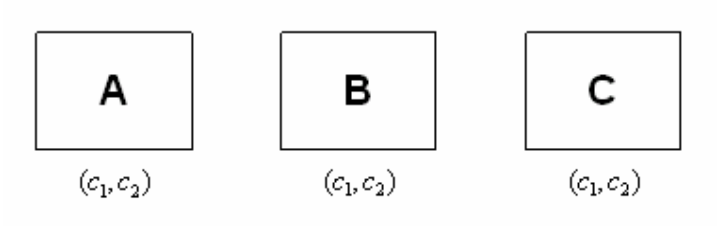

Figura 20 – Diferentes agrupamentos.

Deve ficar assegurado que o centro do agrupamento 1 do agrupamento **A** calculado pelo processador 1 aponta para a mesma área do espaço vetorial que o centro do agrupamento 1 calculado pelo processador 2, e por diante, pois caso contrário, quando os centros calculados forem consolidados, haverá uma distorção dos resultados. Esse problema é abordado por Boutsinas em [12] (Figura 21).

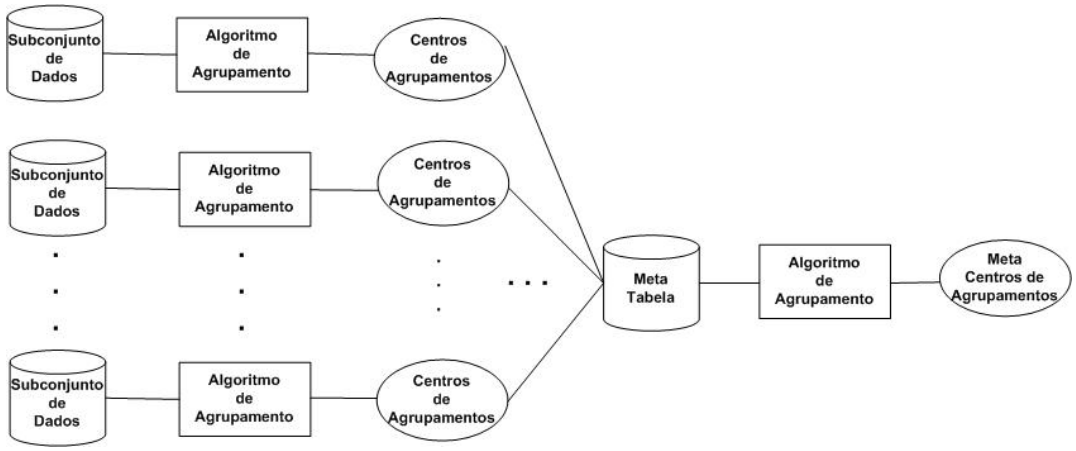

Figura 21 - Esquema do algoritmo proposto por Boutsinas [12].

Como uma forma de superar este problema, o algoritmo paralelo de análise de agrupamentos FCM proposto neste trabalho calcula as distâncias de registros a centros de agrupamentos no ciclo iterativo FCM sincronizando os centros de agrupamentos a cada iteração, garantindo dessa forma que os centros calculados estão se referindo a uma mesma região do espaço vetorial. No entanto, o sincronismo dos processadores a cada ciclo FCM, para totalizar somatórios de medidas de pertinências de registros a agrupamentos a fim de que todos os processadores estejam a cada iteração com os mesmos centros de agrupamentos, gera um alto custo de comunicação. Em contrapartida, um ponto forte dessa abordagem é que ela minimiza a necessidade de memória para processar grandes conjuntos de dados.

Conforme já dito anteriormente, esta abordagem gera um automático balanceamento de carga, pois a divisão em iguais proporções do conjunto de dados por processadores com mesma capacidade de processamento assegura que a carga imposta pelo processamento de cada subconjunto de dados é a mesma em cada processador (Figura 22).

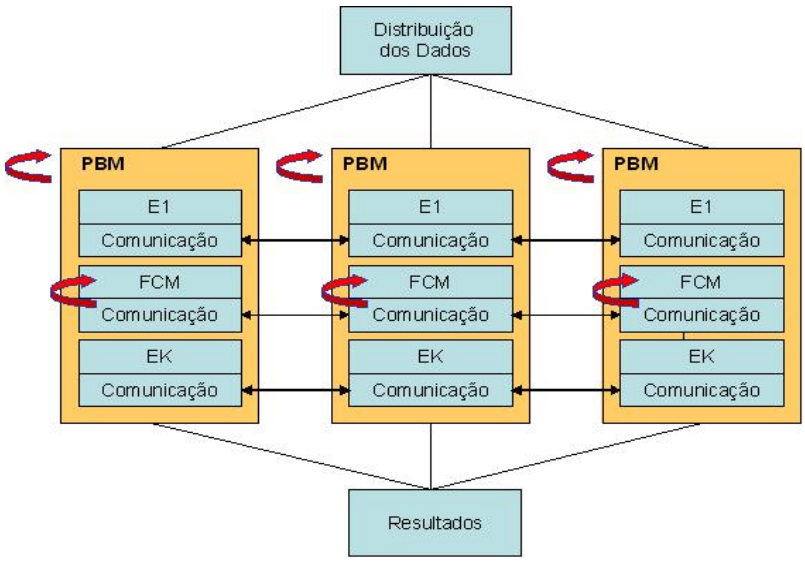

Figura 22 - Diagrama esquemático do algoritmo de análise de agrupamentos FMC por divisão de dados.

O algoritmo está esquematizado na Figura 23 e é descrito pelo procedimento a seguir:

Etapa 1.(Processador raiz): Divide os dados igualmente pelos processadores de forma que cada um receba N/p registros, onde N é o número de registros e p é o número de processadores.

Etapa 2.(Todos os processadores): Calculam o centro geométrico dos dados locais e comunicam esse centro para todos os processadores, de forma que cada processador possa calcular o centro geométrico do conjunto inteiro de dados. Calculam o fator E_1 (7) do índice PBM sobre os dados locais e mandam para o processador raiz.

Etapa 3. (Processador raiz): Calcula os centros de agrupamentos iniciais e os transmite a todos os processadores, de forma que todos tenham os mesmos valores de centros de agrupamentos no início do ciclo FCM.

Etapa 4.(Todos os processadores): Até que a convergência do algoritmo seja alcançada calculam as distâncias de cada registro do conjunto de dados local a todos os centros de agrupamentos, atualizam a matriz de partição como em (5) e calculam novos centros de agrupamentos como em (4).

Etapa 5.(Todos os processadores): Calculam o fator E_K (8) do índice PBM e o enviam ao processador raiz.

Etapa 6.(Processador raiz): Integra o índice PBM como em (6) e o armazena. Se o intervalo de partições foi coberto, o processamento está completo, caso contrário, retorna a *Etapa3*.

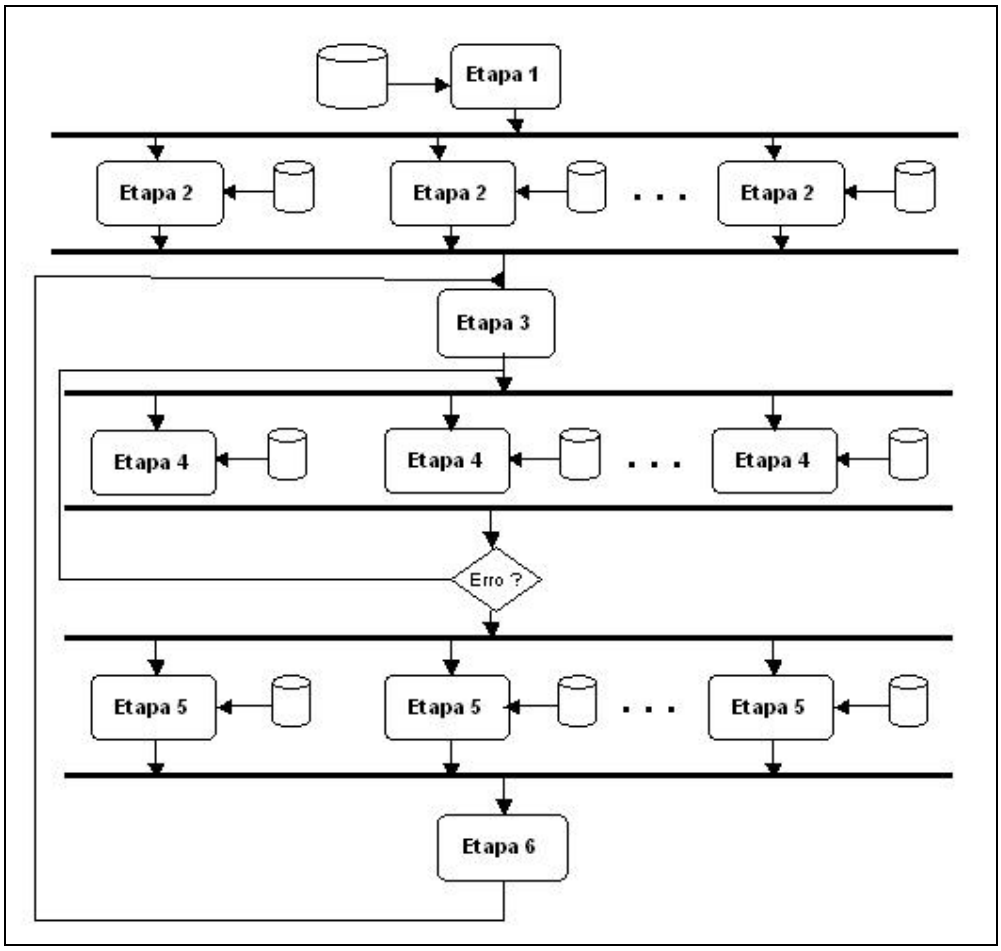

Figura 23 - Algoritmo paralelo de análise de agrupamentos FCM por divisão de dados.

O peso maior do algoritmo está no cálculo das distâncias de cada amostra $\mathbf{x}(t)$ aos centros dos agrupamentos $\mathbf{w}_i$, $i = 1..K$. Esse cálculo é realizado a cada iteração para todos os registros do conjunto de dados. Somatórios das distâncias são utilizados para estimar os novos centros de agrupamentos (4), a matriz de partição *fuzzy* (5) e os fatores para cálculo do índice PBM E_1 (7) e E_K (8). O procedimento é realizado para cada partição FCM do intervalo de partições. O algoritmo responde ao problema da análise de agrupamentos FCM selecionando como a melhor partição a que corresponde ao índice PBM máximo.

A estratégia de aplicar o paralelismo aos dados na análise de agrupamentos FCM paralela é provada válida para grandes volumes de dados e também na investigação de um grande número de partições, onde o tamanho do problema supera consideravelmente

o esforço de comunicação envolvido no sincronismo dos processadores, conforme os resultados dos testes apresentados no próximo capítulo deste trabalho.

5.4 O Algoritmo de Análise de Agrupamentos Paralela FCM balanceado pelo problema das múltiplas mochilas

No algoritmo paralelo de análise de agrupamentos FCM por divisão de dados, o que acontece é que o conjunto de dados é dividido igualmente por todos os processadores que de forma síncrona calculam cada partição. Há um custo de comunicação a cada iteração FCM, pois a cada rodada os processadores se comunicam para trocar somatórios de medidas de pertinências de registros a agrupamentos com o objetivo de que todos os processadores estejam com os mesmos centros de agrupamentos. Como o custo de comunicação é uma variável de grande impacto no desempenho do algoritmo, foi idealizada uma abordagem que otimiza o custo total do processamento paralelo pela minimização da comunicação envolvida no processamento.

Considerando que o problema da análise de agrupamentos deve processar um conjunto de partições FCM, outra forma de aplicar paralelismo a este problema é a da distribuição do conjunto de partições pelos processadores, ao invés da distribuição dos dados.

Nessa situação, a idéia central é que a partir da distribuição das partições pelos processadores, cada processador possa a calcular seu subconjunto de partições de forma independente e ao final do processamento transmitir seus resultados para um processador mestre que consolida os resultados. Sendo K o número de partições e p o número de processadores, cada processador processa $g = K / p$ partições.

Uma conseqüência prática desta abordagem é que cada processador deve ter acesso ao conjunto total dos dados. Essa pode ser considerada uma limitação da estratégia, pois a carga de grandes conjuntos de dados em memória em algumas máquinas paralelas é muitas vezes limitada. No entanto, nas máquinas paralelas onde os algoritmos foram testados isso não foi um problema, mesmo no processamento dos testes de estudo de caso com dados sísmicos, que consistem em um gigantesco volume de dados com mais de um milhão de registros. Além disso, considerando que espaço de armazenamento e

memória se tornam a cada dia mais baratos e disponíveis, essa abordagem foi considerada válida para os objetivos desse trabalho.

A estratégia da divisão das partições tem menor custo de comunicação quando comparada com a anterior, pois não exige o sincronismo dos centros de agrupamentos a cada iteração, consistindo na prática no paradigma de processamento paralelo SPMD sem dependência (Figura 24). Em contrapartida, apresenta uma necessidade de balanceamento de carga devido aos diferentes pesos das partições.

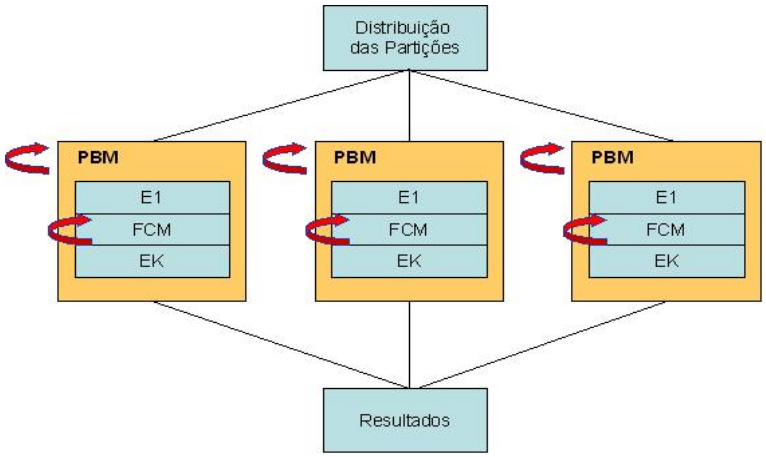

Figura 24 - Diagrama esquemático do algoritmo de análise de agrupamentos FMC por divisão das partições.

No algoritmo FCM o custo do processamento de uma determinada partição é proporcional ao número de agrupamentos que serão calculados. Quanto maior a partição a ser processada, maior é o seu custo computacional, pois quanto maior o número de agrupamentos maior a computação de cálculo das distâncias entre registros e centros de agrupamentos. Assim, partições com maior número de padrões envolvem mais esforço computacional do que partições em menores grupos. Portanto, para que essa abordagem seja efetiva, é necessária uma política de balanceamento de carga que distribua as partições por processadores considerando o custo computacional de cada partição.

O problema existente pode ser entendido através da seguinte proposição: Há uma maneira sistemática de dividir o cálculo das diversas partições FCM pelos processadores que resulte em uma carga balanceada mínima para o processamento paralelo da análise de agrupamentos?

Os dois principais pontos que emergem desse problema são:

1. Que as partições devem ser divididas entre os processadores considerando seus diferentes pesos computacionais;

2. Que a divisão das partições por processadores deve respeitar um valor limitante, calculado para cada processamento, que garanta um custo mínimo para o processamento paralelo.

Se por analogia consideramos as filas de processamento como mochilas e as partições FCM como os itens que devem ser empacotados, o problema pode ser considerado como o problema das múltiplas mochilas, onde itens devem ser empacotados em mochilas respeitando um critério, que no contexto do processamento paralelo é de minimização do custo total de processamento.

Para que o problema seja corretamente modelado devem ser incorporadas as restrições referentes ao contexto da análise de agrupamentos, são elas:

1. Cada partição é atribuída apenas uma vez, ou seja, é um problema 0/1;
2. As partições têm valor igual ao custo;
3. As filas de processamento correspondem às várias mochilas (múltiplas mochilas) e
4. O número de mochilas (processadores) é conhecido a priori e deve permitir a embalagem de todos os itens, pois no contexto da análise de agrupamentos todas as partições devem ser processadas.

As restrições relativas à análise de agrupamentos remetem a casos especiais do problema da mochila. A restrição número 1 remete ao problema da mochila 0 ou 1, a restrição número 2 ao problema do subconjunto e a restrição número 3 ao problema das múltiplas mochilas, todos problemas clássicos para os quais existe a proposição de inúmeros algoritmos e heurísticas. A restrição 4 é bastante específica ao contexto da análise de agrupamentos, pois todas as partições devem ser processadas por um determinado número de processadores (todos os itens devem ser embalados em um número disponível de mochilas) e ela implica no cálculo ou definição do tamanho ideal das mochilas de forma a garantir o menor custo possível para o processamento paralelo.

Considerando que cada processador deve processar um subconjunto do total das partições de forma que o custo total do processamento paralelo seja mínimo, pode-se

concluir que idealmente, o tamanho ideal da fila de processamento de cada processador é o da média das partições, onde, sendo K_i um conjunto de partições com $2 \leq i \leq n$ e p o número de processadores, a carga média do processamento $S = \frac{\sum_{i=2}^{n} K_i}{p}$. Então, idealmente o tamanho da cada fila de processamento deveria ser igual a S.

Conforme o intervalo de partições de cada processamento, o critério da fila de processamento ter tamanho igual ao valor médio das partições pode ter uma solução exata ou não. Existem diversos contextos onde o somatório do conjunto de partições não pode ser dividido exatamente. Em geral, contextos onde as partições têm valores muito dispersos ou em que há pequena proporcionalidade entre o número de partições e o número de processadores apresentam uma distribuição que pode ser bastante desigual dentro dessa estratégia.

Portanto, a abordagem da divisão das partições FCM pelos processadores limitando o custo do processamento paralelo pelo valor médio das partições pode não ser bem sucedida pela impossibilidade de distribuir as partições de forma a respeitar esse critério.

A fim de propor uma abordagem que supere esta limitação e que seja escalável para qualquer conjunto de partições e número de processadores, foi proposta uma heurística que trabalha dentro de uma faixa de valores: um limite inferior definido pelo valor médio das partições indicando a carga ideal para a distribuição, e um limite superior, que é a carga máxima resultante na distribuição indicando o menor custo possível para o processamento paralelo.

A heurística opera em um ciclo iterativo onde uma distribuição é gerada e avaliada. Se a distribuição não é satisfatória o limite inferior é aumentado e uma nova distribuição é gerada e avaliada. A avaliação de cada distribuição é feita pela comparação do coeficiente de variação das cargas com o coeficiente informado no início do processamento. O ciclo termina quando a distribuição gerada é satisfatória ou quando o limite inferior se iguala ao limite superior.

O limite superior da distribuição traduz o custo máximo do processamento paralelo e não pode ser ultrapassado. O objetivo do algoritmo é tentar diminuí-lo. Como ele não pode ser reduzido o algoritmo passa a agrupar as cargas menores em um número menor

de filas de processamento de forma a tentar liberar processadores para que estes possam ser, então, alocados para o processamento das maiores cargas. Essa estratégia tenta reduzir o tempo do processamento paralelo pela concentração de maior poder computacional no processamento das maiores cargas. Ou seja, quando a abordagem esbarra no limite superior do custo do processamento paralelo, que não pode ser reduzido, ela tenta aumentar o poder computacional alocado ao processamento da(s) maior(es) carga(s) para dessa forma reduzir o tempo total do processamento paralelo. Essa tentativa de agrupar as cargas em um número menor de filas de processamento caracteriza o problema do empacotamento.

Quando mais de um processador é alocado ao processamento da mesma carga eles passam a trabalhar segundo o paradigma de divisão de dados proposto no primeiro algoritmo paralelo apresentado neste trabalho.

A proposta deste segundo algoritmo tenta a privilegiar o paradigma da divisão de partições por processadores, mas quando essa estratégia esbarra em limites de escalabilidade pelas próprias características do processamento, a heurística utiliza o paradigma da divisão de dados para continuar reduzindo o custo do processamento paralelo.

O algoritmo proposto é apresentado a seguir. As etapas 1, 2, 3 e 4 que compreendem a heurística de balanceamento de carga estão representadas graficamente no diagrama da Figura 25. O algoritmo completo está representado no diagrama da Figura 26:

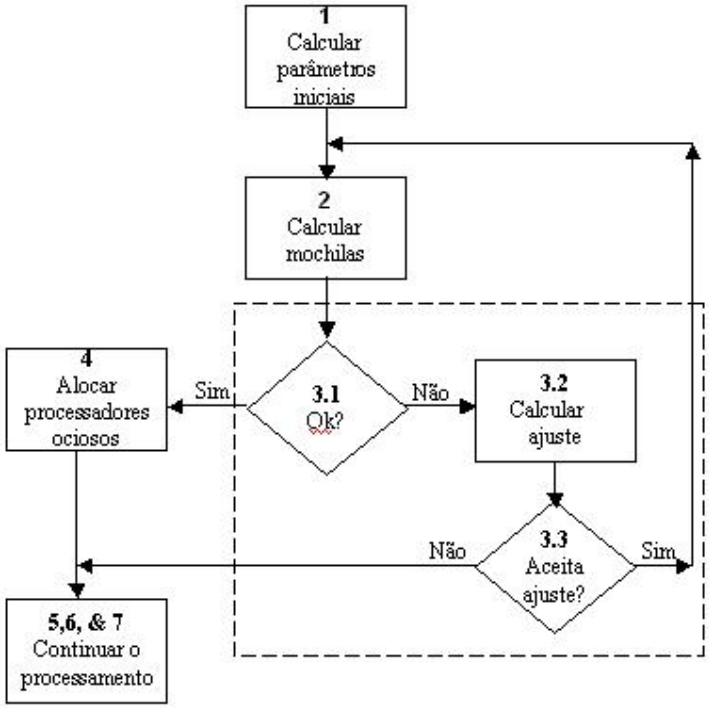

Figura 25 - Detalhamento da parte do algoritmo relativo ao balanceamento de carga modelado pelo problema da mochila.

Algoritmo Proposto

Etapa 1. **Calcular parâmetros iniciais.** *(processador mestre)*

a. Ordenar partições por custo decrescente;

b. Calcular carga média $S = \frac{\sum_{i=2}^{n} K_i}{p}$, onde K_i é o conjunto das partições com $2 \le i \le n$ e p o número de processadores;

Etapa 2. **Gerar as mochilas** *(processador mestre)*

2.1 Atribuir as partições (com custo maior que a média, se houver)

Para cada partição com custo acima ou igual à média

Adicionar partição à uma fila.

2.2 Atribuir as partições com valor abaixo da média (first fit decreasing)

Para cada partição

Verificar se há fila com espaço suficiente e em caso positivo adicionar partição à fila.

2.3 Atribuir partições que não couberam nas filas por falta de espaço

Enquanto houver partições não atribuídas

Ordenar as filas de processamento por espaço livre decrescente.

Atribuir as partições restantes em ordem decrescente, atribuindo as maiores partições às maiores filas.

Se for primeira iteração identificar carga máxima *M* do conjunto de partições.

Etapa 3. Avaliar as mochilas geradas *(processador mestre)*

3.1 Avaliar a distribuição gerada

Calcular média, desvio padrão e coeficiente de variação das cargas de cada processador.

Se coeficiente de variação > variação desejada

3.2 Calcular ajustes

$S = S + 1$

Ajustar carga máxima da iteração anulando processadores com maiores cargas com processadores sem carga.

3.3 Avaliar se a distribuição pode ser ajustada

Se S < que carga máxima ajustada retorna para Etapa 2.

Etapa 4. **Alocar processadores sem carga** *(processador mestre)*

Se houver processadores sem carga atribuir as maiores partições em ordem decrescente até terminar processadores sem carga.

Etapa 5. **Comunicar tarefas de cada processador** *(processador mestre)*

5.1 Comunicar partições que processador vai processar.

5.2 Comunicar se processador é parte de um grupo.

Etapa 6. Processamento da mochila *(Todos os processadores)*

6.1 Se pertence a grupo de processamento preparar grupo

Abrir comunicador

Dividir dados para processar as partições em conjunto

6.2 Para cada partição do seu intervalo processar parâmetros do índice PBM e ciclo FCM.

6.3 Enviar resultados das partições para processador mestre.

Etapa 7. Selecionar a partição com maior PBM. *(processador mestre)*

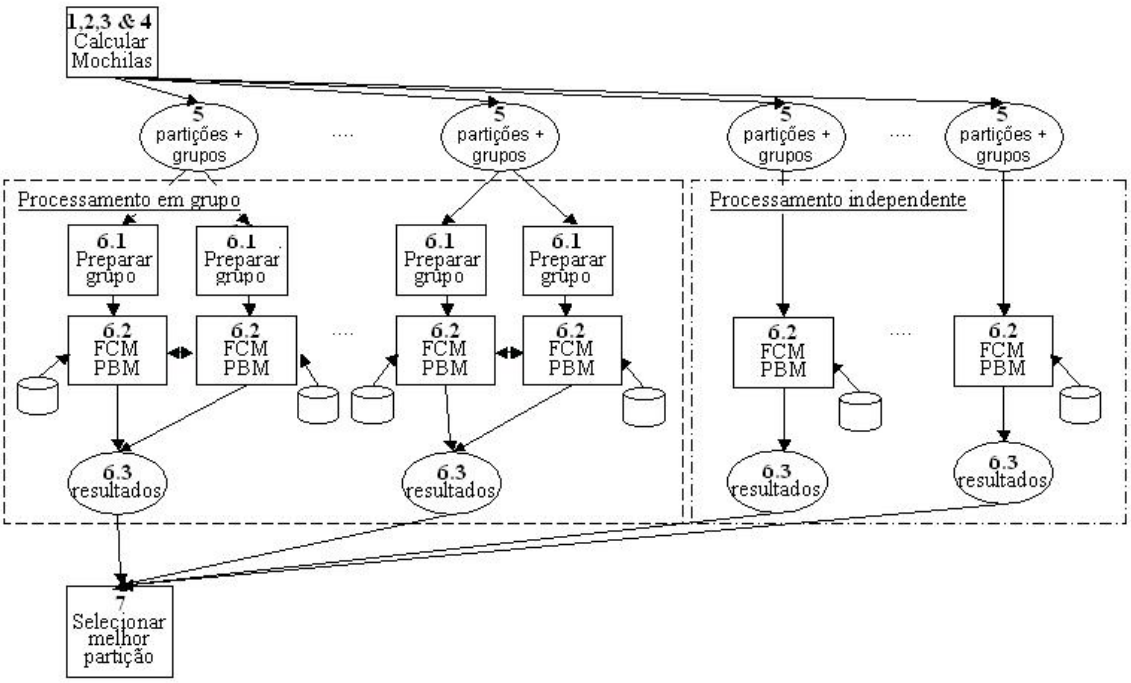

Figura 26 -Esquema de processamento do algoritmo de análise de agrupamentos paralelo FCM com balanceamento de carga modelado pelo problema das múltiplas mochilas.

Testes realizados com o algoritmo demonstram que quando o número de partições do intervalo é grande o bastante para garantir uma razoável proporção em relação ao número de processadores, todos os processadores são carregados com pequenas variações de carga e a distribuição é satisfatória logo na primeira iteração (Figura 27).

No entanto, nos cenários onde a distribuição de partições pela carga média não atinge bom resultado, como por exemplo, quando o número de partições é muito próximo do número de processadores ou quando o número de processadores é maior do que o número de partições, o algoritmo recalcula as mochilas, tentando melhorar o balanceamento de carga.

Isso é feito pelo algoritmo “empurrando” as partições para uma parte da mochila, através do reajuste o valor do tamanho da fila de processamento, mas sem igualar o valor da carga máxima calculada na iteração precedente. A nova distribuição gerada tende a ser mais agrupada utilizando menos mochilas. Esse aumento do tamanho da fila de processamento é limitado ao valor da carga máxima do início do processamento para que o custo mínimo total do processamento paralelo não seja aumentado.

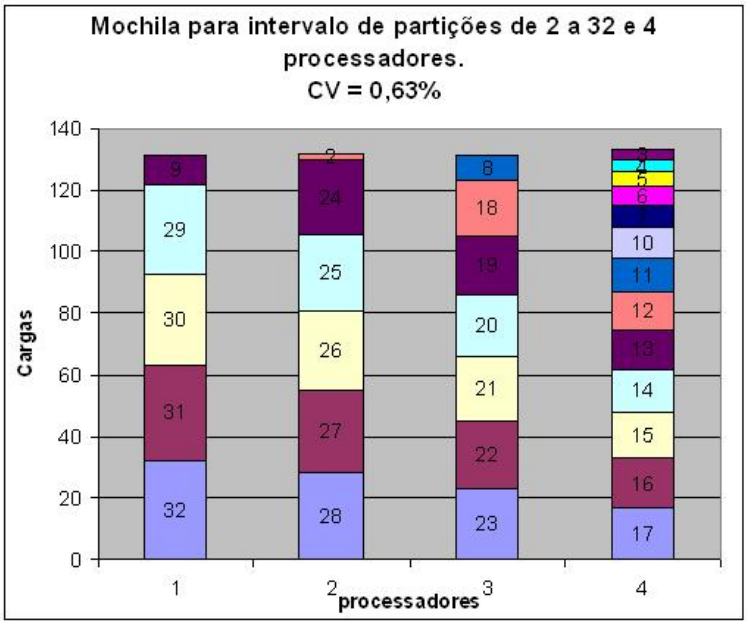

Figura 27 - Mochila gerada pelo algoritmo de balanceamento de carga na primeira iteração do loop de geração da mochila.

Na prática essa tentativa de agrupar as cargas tem o objetivo de liberar processadores que ficando ociosos são remanejados, passando a dividir o processamento das maiores cargas reduzindo o tempo total do processamento paralelo. Um exemplo desse processo é apresentado através dos gráficos das Figura 28 e Figura 29.

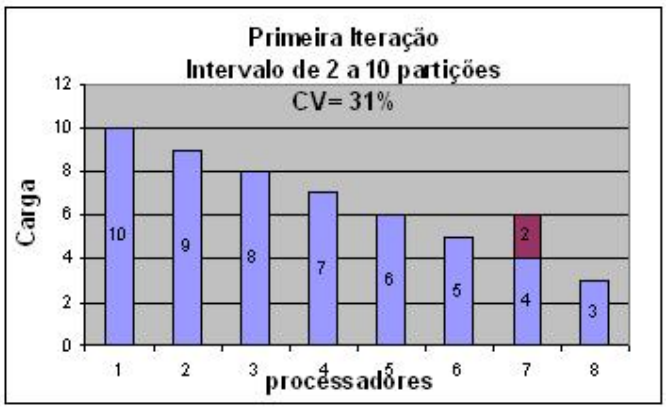

Figura 28 - Mochilas geradas na primeira iteração do algoritmo de balanceamento com coeficiente de variação de comparação de 20%.

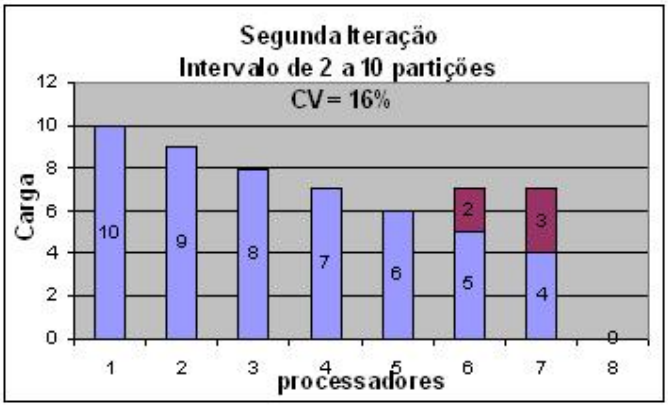

Figura 29 - Mochilas geradas na segunda iteração do algoritmo de balanceamento com coeficiente de variação de comparação de 20%.

No exemplo apresentado nas Figura 28 e Figura 29**,** após duas iterações o algoritmo atende ao coeficiente de variação estabelecido gerando uma boa distribuição. O processador 8, que ficou liberado, recebe a carga máxima, no caso composta pela partição 10, e passa a compor com o processador 1 o grupo que divide o processamento da partição 10. O tempo total do processamento paralelo não será mais o tempo de processar uma carga correspondente à partição 10, pois seu peso computacional agora está dividido entre dois processadores, o que idealmente reduz seu tempo de processamento à metade, e o maior esforço computacional corresponde ao processamento da partição 9.

O algoritmo de balanceamento de carga apresentado propõe uma solução aproximada, mas eficiente para o balanceamento de carga da análise de agrupamentos FCM paralela, conforme resultados apresentados no capítulo 6 deste trabalho. A solução se mostra especialmente eficiente para situações onde o número de partições a ser investigada é bem maior que o número de processadores disponíveis. Avaliações e resultados são apresentados a seguir.

6 Testes dos Algoritmos e Análise dos Resultados

6.1 Arquitetura utilizada

Os algoritmos paralelos gerados para investigar o problema da análise de agrupamentos FCM foram projetados para funcionar em máquinas paralelas de memória distribuída do Núcleo de Alto Desempenho da Universidade Federal do Rio de Janeiro (NACAD). Foram elas:

. **Cluster Mercury**

- Cluster de PCs com 16 dual Pentium III, 1 GHz, nós de processamento interconectados via Fast Ethernet e redes Gb com 8GB de memória RAM total.

. Máquina Saturno

- SGI Altix 350 com 14 Intel Itanium2 CPUs com 28 GB de memória RAM (compartilhada - NUMA) e 360 GB armazenamento em disco.

. Máquina Vênus

- SGI Altix 450 com 32 Intel Itanium2 CPUs (1.6 GHz) com 64 GB de memória de RAM (compartilhada - NUMA) e 360 GB de armazenamento em disco.

A comunicação entre os processadores foi feita pela passagem de pacotes de mensagens utilizando a biblioteca MPI MPT (Message Passing Toolkit). O job scheduler PBS (Portable Batch System) foi usado para execução e controle de jobs, e o sistema operacional administrativo e de processamento dos nós foi o Linux Red Hat no Cluster Mercury. Os programas foram desenvolvidos na linguagem C.

6.2 MPI – Message Passing Interface

MPI (Message Passing Interface) é uma biblioteca de funções de comunicação por passagem de mensagens projetada para ser um padrão para arquiteturas paralelas com memória distribuída.

Suas principais características são padronização, portabilidade, desempenho, funcionalidade e disponibilidade. Na sua utilização, todo o paralelismo é explícito, ou seja, o programador é responsável por identificar e implementar corretamente o código

paralelo usando as funções do MPI. É possível usar as funções MPI com as linguagens C, C++ e FORTRAN.

As seguintes funções MPI foram utilizadas na implementação do algoritmo de análise de agrupamentos paralelo:

- **MPI_Init( )** ➔ Inicializa o ambiente de execução MPI;
- **MPI_Comm_size()** ➔ Retorna o número de processadores correntemente sendo usados;
- **MPI_Comm_rank** ➔ Retorna a identificação do processo;
- **MPI_Bcast()** ➔ Envia dados do processo root (ou identificado como 0) para todos os processos do comunicador (Figura 30);
- **MPI_Allreduce(a,b,Oper)** ➔ Executa a operação Oper tendo como argumentos todos os valores colocados em a e coloca o resultado em b. Sendo a e b variáveis de todos os processos (Figura 31);
- **MPI_WTime()** ➔ Retorna o número de segundos transcorridos de uma chamada da função a algum tempo arbitrário fixado no passado. Para calcular o tempo de execução de um processo paralelo basta calcular a diferença entre duas chamadas da função.
- **MPI_Finalize()** ➔ Termina o ambiente de processamento MPI.

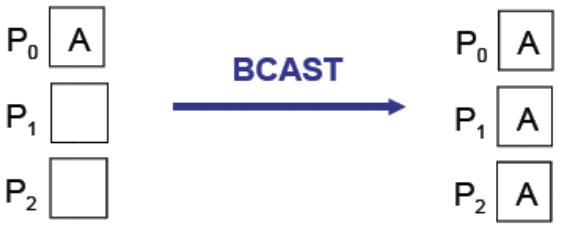

Figura 30 - Funcionamento da função BROADCAST.

Figura 31 - Funcionamento da função ALLREDUCE.

6.3 Testes, Resultados e Análises

Os algoritmos de análise de agrupamentos FCM paralelos foram submetidos a conjuntos de testes com dois propósitos distintos:

1. Testes para avaliação de desempenho e escalabilidade dos algoritmos e
2. Testes de comparação das duas abordagens.

A seguir são descritos os testes realizados e apresentadas as análises dos resultados.

6.3.1 Testes de Avaliação de Desempenho e Escalabilidade

Propósito dos testes: Avaliar o impacto do número de registros, variáveis e das partições no desempenho do algoritmo.

Máquinas utilizadas: Dois conjuntos de testes foram realizados com o algoritmo paralelo de análise de agrupamentos FCM – divisão de dados. O primeiro foi realizado no Cluster Mercury e o segundo na máquina Altix Saturno 350, ambas do Núcleo de Alto Desempenho da UFRJ – NACAD.

Inicialmente, os testes foram planejados para rodar no Cluster Mercury, pois havia o objetivo de demonstrar que o algoritmo podia rodar em máquinas de baixo custo e que, portanto, apresentava grande aplicabilidade prática, não dependendo da aquisição de máquinas paralelas de nenhum fornecedor e de nenhuma configuração específica.

Os primeiros testes foram realizados no Cluster Mercury, e conseguiram demonstrar a escalabilidade do algoritmo, mas as atividades de testes nessa máquina tiveram que ser suspensas devido à desativação do Cluster Mercury durante a realização desse estudo. Assim, os testes relativos de escalabilidade e eficiência do algoritmo foram realizados em duas máquinas distintas, inicialmente no Cluster Mercury e em seguida completados na máquina Altix 350 Saturno.

1 *Avaliar o impacto do número de registros no desempenho do algoritmo*

Hardware: Cluster Mercury

Dados de teste:

Dados sintéticos. Os dados foram gerados por programas que replicaram os registros extraídos da base de dados de benchmark [59].

Os arquivos foram gerados com 1.000, 12.500, 50.000, 65.000 e 100.000 registros (linhas) e tamanhos de 38KB, 500KB, 1.88MB, 2.5MB e 3.76MB.

Cada registro tinha dezoito atributos inteiros com valores entre 0 e 10.

Descrição do teste:

Os testes foram realizados para o cálculo de nove partições, de $K = 2$ a $K = 10$, usadas para o cálculo do índice PBM.

Todos os arquivos foram processados com 1, 2, 4, 6, e 8 processadores.

Resultados:

As curvas de *speedup* (3.5) estão apresentadas na Figura 32 e seus valores na Tabela 1.

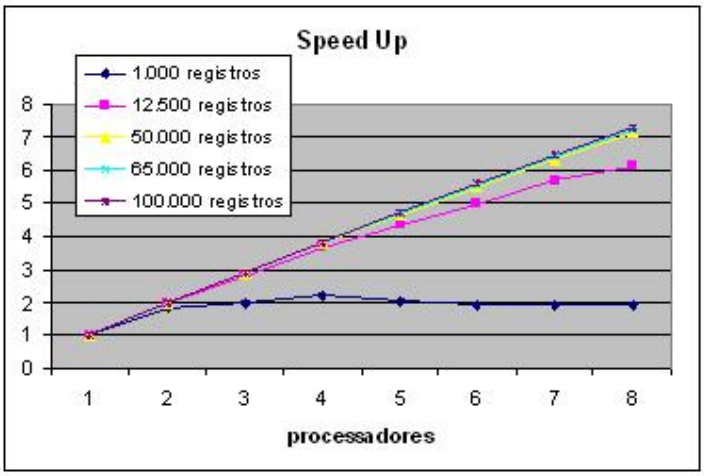

Figura 32 - Curvas de *speedup* relativas ao processamento de arquivos com diferente número de registros no Cluster Mercury.

Tabela 1 - Valores de *speedup* do teste no Cluster Mercury.

Registros	Número de Processadores							
	1	2	3	4	5	6	7	8
1.000	1	1,84	1,97	2,24	2,01	1,94	1,95	1,91
12.500	1	1,96	2,80	3,67	4,33	4,97	5,67	6,12
50.000	1	1,96	2,89	3,80	4,62	5,48	6,32	7,11
65.000	1	1,96	2,90	3,82	4,68	5,55	6,41	7,21
100.000	1	1,96	2,91	3,83	4,72	5,60	6,47	7,30

Análise:

As curvas de *speedup* melhoram consideravelmente com o aumento do número de registros.

A pior curva é relativa ao processamento do menor arquivo. Acrescentar processadores não é efetivo no processamento de arquivos pequenos. No processamento de arquivos pequenos, com pequeno número de registros, o custo da comunicação fica muito alto se comparado às vantagens da paralelização do cálculo das distâncias de registros a centros de clusters e os benefícios do paralelismo não acontecem.

No entanto, no processamento dos arquivos maiores que 1.88MB, valores de *speedup* bastante satisfatórios são alcançados, demonstrando que o algoritmo apresenta bom desempenho e é escalável no processamento de grandes volumes de dados.

***2** Avaliar o impacto do número de variáveis e do intervalo de partições no desempenho do algoritmo*

Hardware: Cluster Mercury

Dados de teste:

Dados sintéticos. Os dados foram gerados por programas que replicaram os registros extraídos da base de dados de benchmark [59].

Arquivos usados: 50.000 linhas com 10 atributos, tamanho de 1.07MB e arquivo de 50.000 linhas com 40 atributos, tamanho de 4.12MB.

Todos os atributos eram inteiros com valores entre 0 e 10.

Descrição do teste:

Os testes foram realizados com diferentes intervalos de partições. Os dois arquivos foram processados com uma partição de 2 agrupamentos, partições de 2, 3 e 4 agrupamentos, partições de 2 a 8 agrupamentos, partições de 2 a 16 agrupamentos e partições de 2 a 32 agrupamentos.

Todos os intervalos foram processados com 1, 2, 4, 6, e 8 processadores.

Resultados:

As curvas de *speedup* são apresentadas na Figura 33 e na Figura 34 e seus valores na Tabela 2 e na Tabela 3.

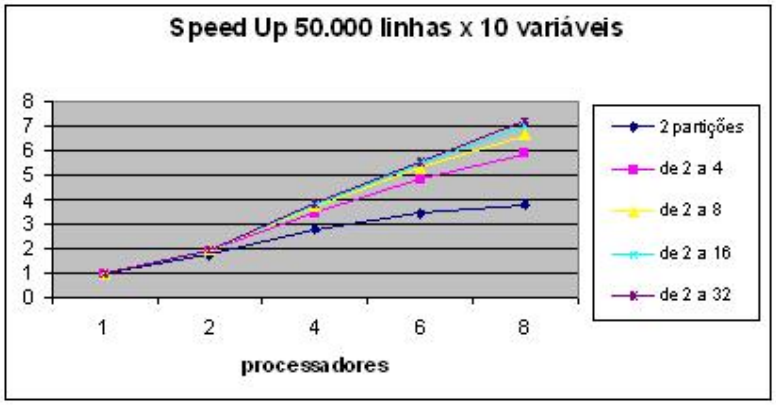

Figura 33 - Curvas de *speedup* relativas ao processamento de diferentes intervalos de partições no Cluster Mercury – arquivo de 50.000 linhas x 10 variáveis.

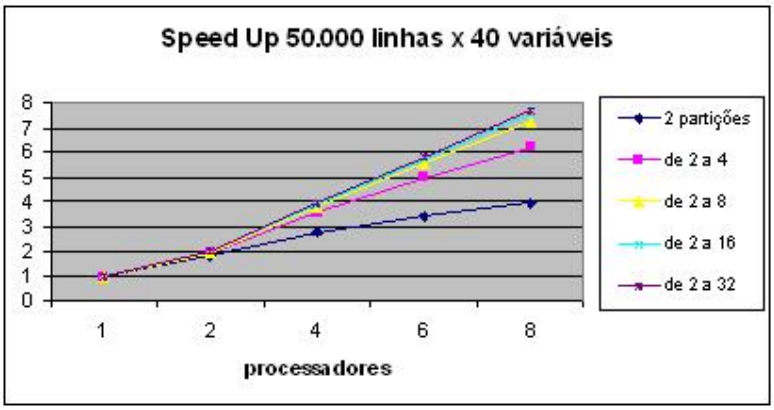

Figura 34 -Curvas de *speedup* relativas ao processamento de diferentes intervalos de partições no Cluster Mercury arquivo de 50.000 linhas x 40 variáveis.

Tabela 2 - *Speedup* do processamento com arquivo de 50.000 linhas x 10 variáveis.

Intervalo de Partições	Numero de Processadores				
	1	2	4	6	8
2 partições	1	1,77	2,82	3,44	3,79
de 2 a 4	1	1,91	3,47	4,78	5,84
de 2 a 8	1	1,94	3,71	5,26	6,66
de 2 a 16	1	1,95	3,78	5,46	7,02
de 2 a 32	1	1,96	3,82	5,52	7,19

Tabela 3 - *Speedup* do processamento com arquivo de 50.000 linhas x 40 variáveis.

Intervalo de Partições	Numero de Processadores				
	1	2	4	6	8
2 partições	1	1,78	2,80	3,45	3,93
de 2 a 4	1	1,92	3,56	4,94	6,22
de 2 a 8	1	1,96	3,82	5,54	7,21
de 2 a 16	1	1,98	3,89	5,73	7,53
de 2 a 32	1	1,98	3,92	5,81	7,67

Análise:

Os melhores valores de *speedup* foram obtidos no processamento do arquivo com maior número de variáveis para qualquer intervalo de partições calculado. Pode-se, portanto concluir que o algoritmo apresenta melhor desempenho no processamento de maior número de variáveis. Uma explicação para este fato é a de que no processamento

dos menores volumes de dados os custos de comunicação pesam mais do que os benefícios do paralelismo.

O número do intervalo de partições também apresenta impacto direto no desempenho do algoritmo. Os benefícios do paralelismo aparecem na mesma proporção em que o número de partições cresce. Isso pode ser explicado pelo fato de que quanto maior é o número de partições que serão calculadas, mais computação está envolvida no processamento, devido ao cálculo das distâncias dos registros aos centros de agrupamentos. Conforme a quantidade de cálculos aumenta, os benefícios da utilização do paralelismo aparecem mais claramente.

Os melhores valores de *speedup* são obtidos no processamento do arquivo com maior número de variáveis e do maior intervalo de partições, demonstrando que o algoritmo tem boa aplicabilidade para o processamento de grandes volumes de dados e que é escalável nesse tipo de hardware.

3 *Avaliar o impacto do número de variáveis, registros e do intervalo de partições no desempenho do algoritmo*

Hardware: Altix 350 Saturno

Dados de teste:

Foram usados doze arquivos com dados sintéticos gerados por programas que replicaram os registros extraídos da base de dados de benchmark [59].

Todos os atributos eram inteiros com valores entre 0 e 10.

O esquema com as dimensões dos arquivos usados nos testes são apresentados na Tabela 4. As dimensões e tamanho de cada arquivo de teste foram apresentados na Tabela 5.

Tabela 4 - Dimensões dos arquivos de teste.

Registros	Variáveis			
50.000	50	100	150	200
100.000	50	100	150	200
150.000	50	-	-	-
200.000	50	100	150	-

Descrição do teste:

Foram realizados duzentos e oitenta e oito testes de desempenho usando os arquivos apresentados na Tabela 5.

Os testes foram realizados com uma partição de 2 agrupamentos e com intervalos de partições de 2, 3 e 4 agrupamentos, de 2 a 8 agrupamentos e de 2 a 16 agrupamentos.

Todos os intervalos foram processados com 1, 2, 3, 4, 5 e 6 processadores.

Os resultados e análises dos testes são apresentados separadamente em relação a cada aspecto observado nos testes, ou seja, separados quanto a impacto do aumento do número de registros, quanto ao aumento do número de variáveis e quanto ao aumento do número de partições. Os resultados mais representativos de cada caso são apresentados e comentados.

Tabela 5 – Tamanho dos arquivos de teste.

Identificação do Arquivo	Registros (milhares) x Variáveis	Tamanho (MB)
1	50 x 50	5.138
2	50 x 100	10.227
3	50 x 150	15.317
4	50 x 200	20.406
5	100 x 50	10.347
6	100 x 100	20.552
7	100 x 150	30.730
8	100 x 200	40.909
9	150 x 50	15.561
10	200 x 50	20.747
11	200 x 100	41.104
12	200 x 150	61.469

Resultado 1: Impacto do número de registros no desempenho do algoritmo

Tabela 6 - Tempos de processamento variando o número de registros com número fixo de 50 variáveis.

Intervalo de Partições	Número de Registros	Processadores					
		1	2	3	4	5	6
2	50.000	6,204519	4,459167	3,961105	3,330845	3,180190	3,050159
	100.000	15,716073	8,949648	7,511082	6,708606	6,634592	5,881509
	200.000	24,819229	16,778518	17,194461	13,306745	11,932726	11,820070
4	50.000	37,884057	17,895794	16,806470	12,836225	13,327468	10,209263
	100.000	96,640253	59,659188	38,235850	25,944614	22,808330	21,724819
	200.000	223,773259	107,172222	68,771022	57,915676	56,051443	46,170797
8	50.000	200,679868	114,920623	88,120902	53,656157	46,600746	41,010480
	100.000	475,771353	239,606612	161,465136	115,435392	81,553531	88,424591
	200.000	876,743919	519,489051	334,618063	215,900302	169,074074	158,855838
16	50.000	1.800,829288	872,104768	559,898155	436,472400	337,112004	294,556893
	100.000	3.460,099178	1.416,643067	1.218,321425	869,201020	679,319966	579,739406
	200.000	7.078,542063	3.574,867393	2.392,248032	1.721,448129	1.369,351690	1.087,757584

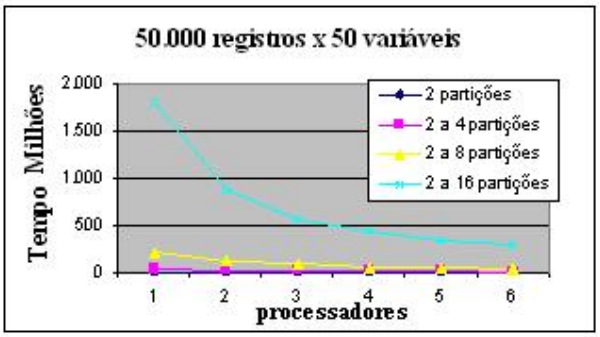

Figura 35 – Tempo de processamento de arquivo com 50.000 registros e 50 variáveis para diferentes intervalos de partições.

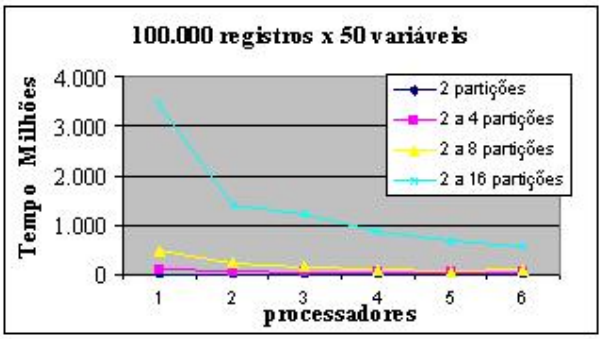

Figura 36 - Tempo de processamento de arquivo com 100.000 registros e 50 variáveis para diferentes intervalos de partições.

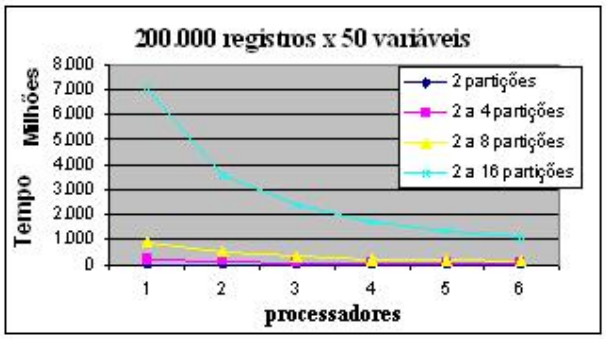

Figura 37- Tempo de processamento de arquivo com 200.000 registros e 50 variáveis para diferentes intervalos de partições.

Análise:

Foi observado que o aumento do tempo de processamento do algoritmo é diretamente proporcional ao aumento do número de registros. O tempo de processamento aumenta na mesma proporção em que aumentam os registros seja qual for o intervalo de partições (Figura 35, Figura 36 e Figura 37).

Resultados 2: Impacto do número de variáveis no desempenho do algoritmo.

Tabela 7 - - Tempos de processamento variando o número de variáveis com número fixo de 100.000 registros.

Intervalo de Partições	Número de Variáveis	Processadores					
		1	2	3	4	5	6
2	50	15,716073	8,949648	75,110820	6,708606	6,634592	5,881509
	100	26,267237	16,241356	15,071309	12,871869	11,470805	11,725922
	200	51,839783	40,345933	28,095541	27,339369	24,337734	22,229937
4	50	96,640253	59,659188	38,235850	25,944614	22,808330	21,724819
	100	182,686097	100,272878	92,842828	60,854545	52,107163	53,570418
	200	530,426093	247,538593	155,079657	147,374870	146,036286	71,911678
8	50	475,771353	239,606612	161,465136	115,435392	81,553531	88,424591
	100	1.189,248070	537,205977	309,487086	263,846291	201,908637	166,511040
	200	2.918,384553	1.682,775058	847,732534	689,238319	430,521036	454,664237
16	50	3.460,099178	1.416,643067	1.218,321425	869,201020	679,319966	579,739406
	100	7.158,952608	3.809,566389	2.431,179152	1.620,442398	1.481,858118	1.274,003089
	200	17.614,318679	8.152,742352	5.314,040268	3.984,544104	3.401,449762	258,188141

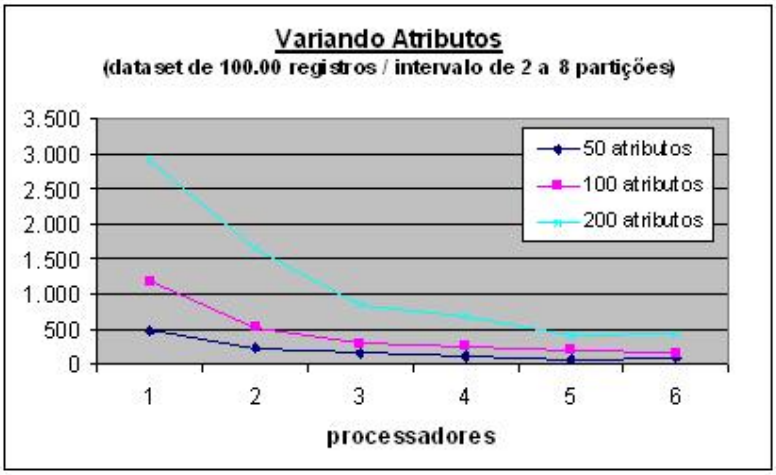

Figura 38 - Tempo de processamento de arquivos com diferente número de atributos para intervalo de 2 a 8 partições.

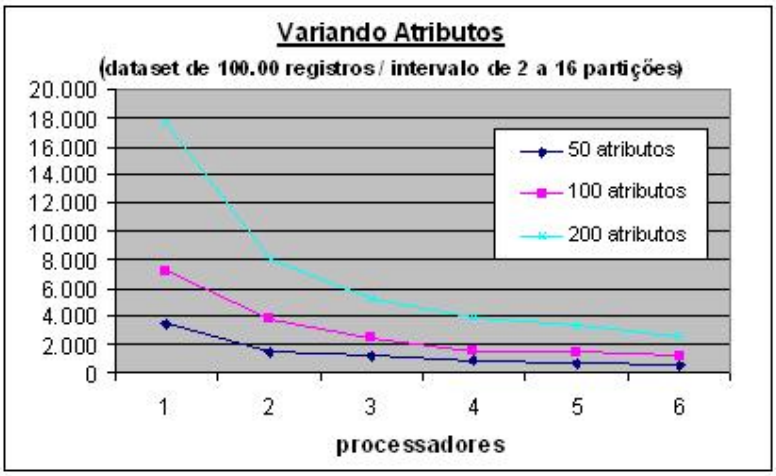

Figura 39 - Tempo de processamento de arquivos com diferente número de atributos para intervalo de 2 a 16 partições.

Análise:

O tempo de processamento do algoritmo aumentou proporcionalmente ao aumento do número de atributos. O tempo de processamento aumenta na mesma taxa em que aumenta o número de atributos dos arquivos (Figura 38 e Figura 39).

Análise dos resultados do aumento do número de registros e de atributos ao mesmo tempo

Os gráficos as seguir têm o objetivo de apresentar de forma sintetizada o comportamento do algoritmo quando as duas condições, o aumento do número de registros e aumento do número de atributos, ocorrem ao mesmo tempo.

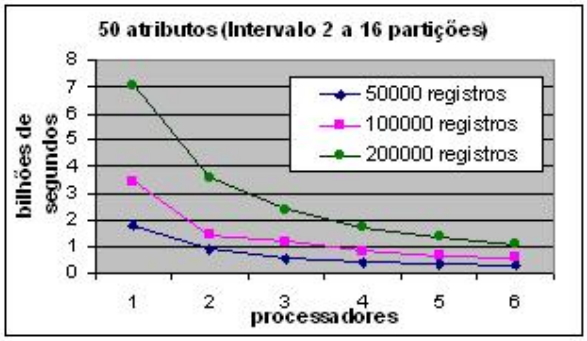

Figura 40 - Tempos de processamento aumentando o número de registros (50 atributos).

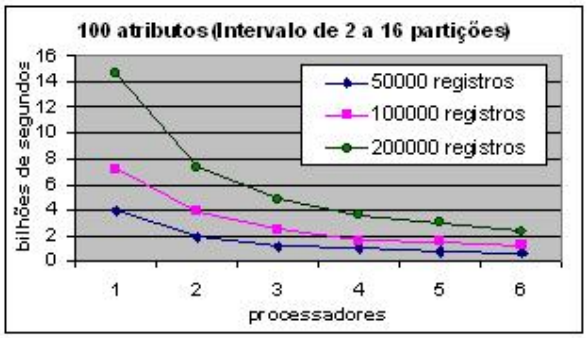

Figura 41 - Tempos de processamento aumentando o número de registros (100 atributos).

Conforme os resultados apresentados na Figura 40 e na Figura 41, o tempo de processamento do algoritmo aumenta na proporção em que aumentam o número de registros e de atributos dos arquivos de teste.

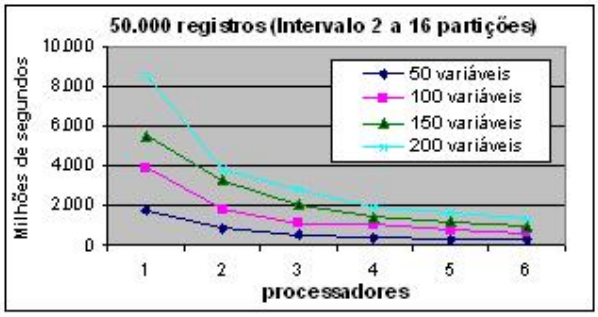

Figura 42 - Tempos de processamento aumentando o número de atributos (50.000 registros).

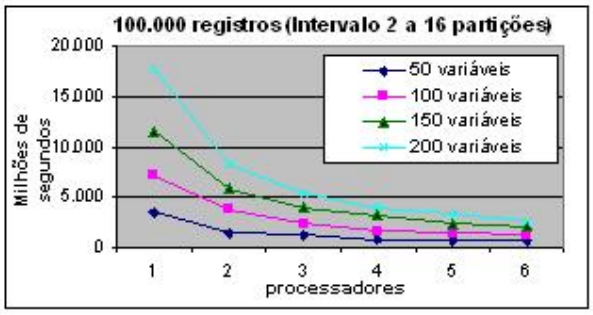

Figura 43 - Tempos de processamento aumentando o número de atributos (100.000 registros).

Analisando os resultados de forma inversa, conforme gráficos apresentados na Figura 42 e na Figura 43, o processamento do algoritmo aumenta na proporção em que aumentam os atributos e registros dos arquivos de teste.

Constata-se, portanto, que o tempo de processamento do algoritmo para análise de agrupamentos paralela FCM é linearmente proporcional ao aumento do tamanho do problema, ou seja, linearmente proporcional ao aumento do número de registros e variáveis dos conjuntos de dados.

Em todos os testes realizados o algoritmo paralelo reduz o tempo de processamento dos arquivos quando são acrescentados novos processadores demonstrando sua escalabilidade, mas essa redução é mais significativa no processamento dos grandes conjuntos de dados onde o processamento de distâncias de registros a centros de agrupamentos é significantemente maior do que nos arquivos menores.

Resultados 3: Avaliar o impacto do número de partições do intervalo de agrupamentos no desempenho do algoritmo.

Para essa avaliação são apresentados resultados obtidos no processamento de diferentes intervalos de partições com arquivo de 200.000 registros e 100 atributos.

As curvas de tempo de processamento do algoritmo são apresentadas em dois gráficos (Figura 44 e Figura 45) devido à diferença de grandeza dos tempos de processamento dos intervalos.

Tabela 8 – Tempos de processamento de diferentes intervalos de partições para 1, 2, 3, 4, 5 e 6 processadores.

Partições	Processamento de arquivo com 200.000 registros e 100 atributos					
2	40,668712	25,809221	12,695962	10,713711	8,758235	6,750953
2 a 4	407,435793	202,615907	147,897918	122,894546	106,624920	62,017368
2 a 8	1.848,145489	959,516333	701,180642	526,498358	359,034782	322,394114
2 a 16	14.640,105900	7.378,311848	4.806,076349	3.549,043560	2.944,447473	2.354,987200

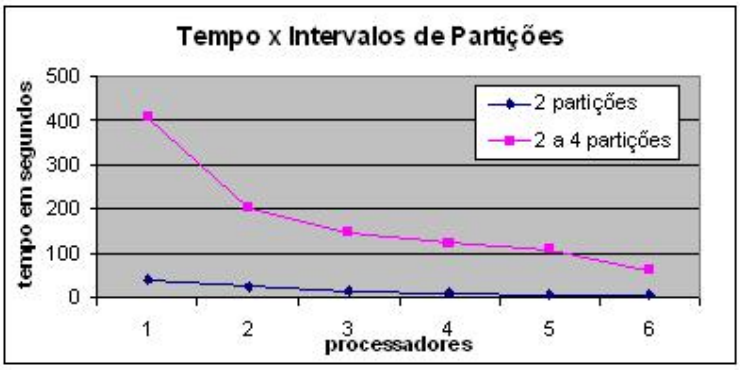

Figura 44 - Tempos de processamento dos intervalos de 2 partições e de 2 a 4 partições no processamento de arquivo de 200.000 registros e 100 atributos.

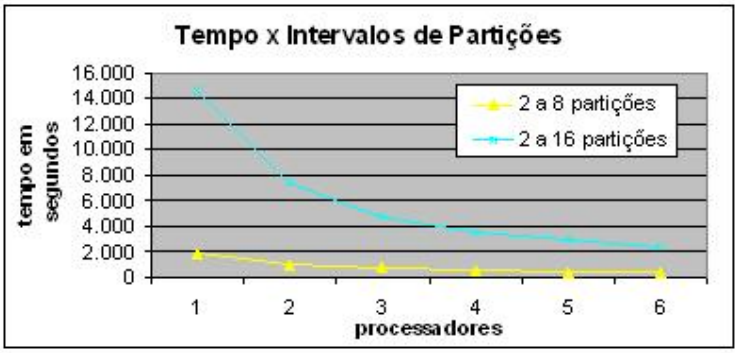

Figura 45 - Tempos de processamento dos intervalos de 2 a 8 partições e de 2 a 16 partições no processamento de arquivo de 200.000 registros e 100 atributos.

Quanto maior o número de partições FCM do intervalo a ser processado pelo algoritmo de análise de agrupamentos paralelo FCM maior são os tempos de processamento.

Pode-se observar que sempre existe uma correlação linear entre o aumento do número de amostras (registros) do conjunto de dados e o aumento do tempo de processamento (Figura 46 e Figura 47).

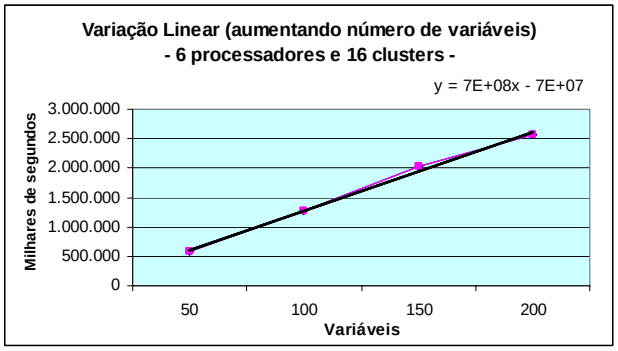

Figura 46 – Tempos de processamento de intervalo de 2 a 16 partições para diferentes números de atributos utilizando 6 processadores.

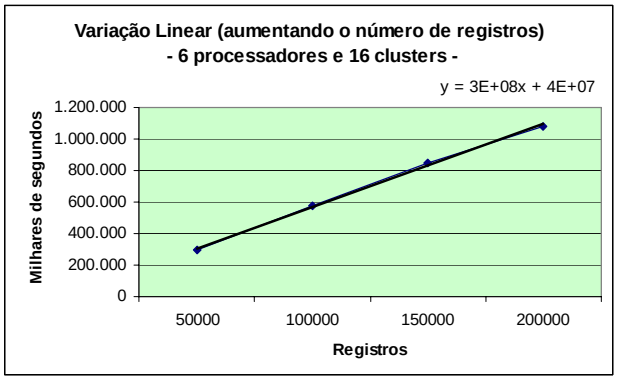

Figura 47 - Tempos de processamento de intervalo de 2 a 16 partições para diferentes números de registros utilizando 6 processadores.

O gráfico apresentado na Figura 48 apresenta os valores dos tempos de processamento do algoritmo no processamento de diferentes intervalos de partições usando o mesmo arquivo e o mesmo número de processadores. Os tempos são ajustados por uma curva exponencial, indicando que o aumento do número de partições causa o crescimento exponencial do tempo de processamento.

Isso na prática é devido ao fato de que o aumento do número de agrupamentos implica no aumento de tanto do cálculo das operações entre centros de agrupamentos e atributos como no aumento das operações entre centros de agrupamentos e registros do conjunto de dados. Além disso, há o aumento do número de partições a ser calculado, sendo o processo de cálculo dos centros de agrupamentos tendo que ser realizado para cada partição do intervalo.

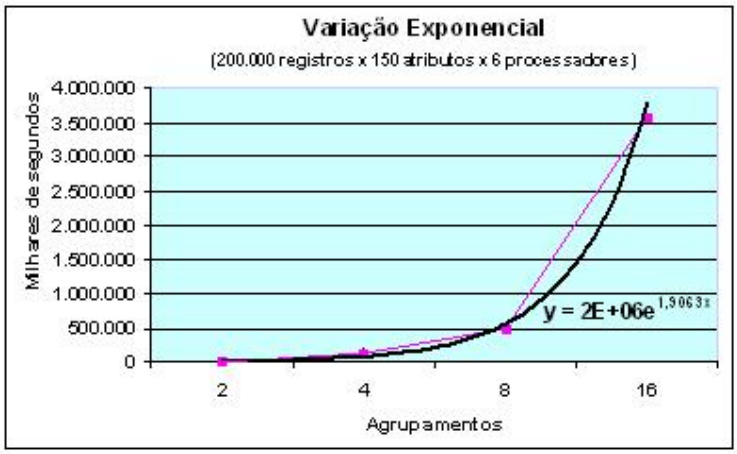

Figura 48 - Tempos de processamento de intervalo de 2 a 16 partições com número de registros variando de 50.000 a 200.000.

Portanto, no algoritmo paralelo de análise de agrupamentos FCM o número de partições é a variável de maior peso, devido a natureza iterativa do processo.

Análise de *Speedup* e de Eficiência:

A seguir, na Tabela 9 são apresentados os valores de *speedup* do algoritmo nos testes relativos ao processamento com arquivos de 200.00 registros com 50, 100 e 150 atributos na máquina Altix 350 Saturno. Também são apresentados na Tabela 10 os valores de eficiência relativos aos mesmos testes.

Tabela 9 - *Speedups* relativos ao processamento de arquivos de 200.000 registros com 50, 100 e 150 atributos.

Arquivo de 200.000 registros							
Atributos	Partições	Speed Up Altix Saturno					
		1	2	3	4	5	6
50	2	1	1,989444258	1,865994076	3,220476908	4,32200747	4,442105684
	2 a 4	1	2,181639522	3,562625609	4,334595791	4,490390569	5,652140422
	2 a 8	1	1,69928421	2,662413871	4,181264763	5,393459736	5,759129368
	2 a 16	1	1,982531059	2,965741056	4,126756117	5,193754816	6,548344664
100	2	1	1,575743491	3,203279279	3,795950068	4,64348262	6,024143851
	2 a 4	1	2,010877621	2,754844683	3,315328518	3,821206084	6,569704683
	2 a 8	1	1,926121969	2,635762282	3,51025879	5,14753885	5,732565853
	2 a 16	1	1,984208068	3,046165903	4,133399558	4,972106324	6,216639267
150	2	1	1,982115637	2,436784311	3,84995907	4,722684202	6,161966671
	2 a 4	1	1,810723982	3,518088546	3,895552087	5,854076429	6,220999805
	2 a 8	1	2,233411513	3,505465855	4,893565693	5,42545839	8,099014977
	2 a 16	1	2,07776623	2,935097462	4,246621175	4,847821197	6,396164057

Tabela 10 – Valores de eficiência relativos ao processamento de arquivos de 200.000 registros com 50, 100 e 150 atributos.

Arquivo de 200.000 registros							
Atributos	Partições	Eficiência Altix Saturno					
		1	2	3	4	5	6
50	2	1	0,994722129	0,621998025	0,805119227	0,864401494	0,740350947
	2 a 4	1	1,090819761	1,18754187	1,083648948	0,898078114	0,942023404
	2 a 8	1	0,849642105	0,88747129	1,045316191	1,078691947	0,959854895
	2 a 16	1	0,991265529	0,988580352	1,031689029	1,038750963	1,091390777
100	2	1	0,787871746	1,06775976	0,948987517	0,928696524	1,004023975
	2 a 4	1	1,00543881	0,918281561	0,828832129	0,764241217	1,094950781
	2 a 8	1	0,963060985	0,878587427	0,877564697	1,02950777	0,955427642
	2 a 16	1	0,992104034	1,015388634	1,03334989	0,994421265	1,036106544
150	2	1	0,991057818	0,812261437	0,962489767	0,94453684	1,026994445
	2 a 4	1	0,905361991	1,172696182	0,973888022	1,170815286	1,036833301
	2 a 8	1	1,116705756	1,168488618	1,223391423	1,085091678	1,349835829
	2 a 16	1	1,038883115	0,978365821	1,061655294	0,969564239	1,066027343

Os testes na máquina Saturno apresentam em algumas situações valores de eficiência superlineares como mostrado na Figura 49 e na Figura 50. Esse comportamento pode ser explicado pelo fato de que a distribuição de um arquivo grande por vários processadores produz arquivos menores para cada processador.

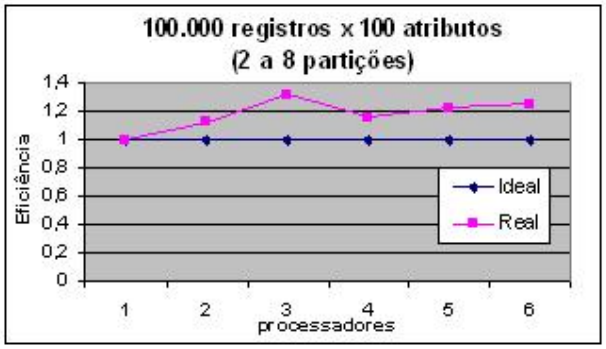

Figura 49 - Valores de eficiência superlineares.

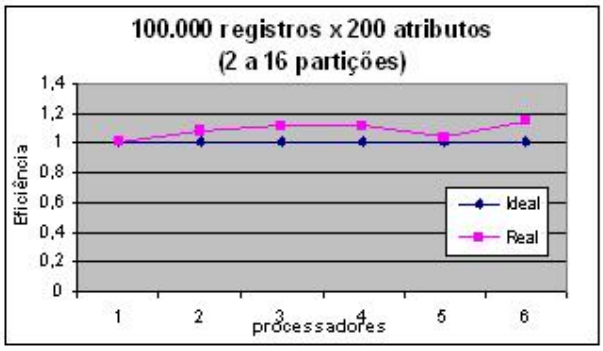

Figura 50 - Valores de eficiência superlineares.

A arquitetura das máquinas Altix 350 provê uma melhoria de desempenho para aplicações com essa característica. Se o arquivo processado couber todo na memória cache, o tempo computacional decresce e a curva de eficiência se torna superlinear.

Outro ponto a destacar nos resultados dos testes na máquina Altix Saturno, foi relativo ao processamento do algoritmo para pequeno número de partições. Esses testes, em geral, não apresentaram bons valores de seedup. Os resultados foram piores quanto menor os tamanhos dos arquivos processados (Figura 51 e Figura 52).

Esse fato pode ser compreendido ao considerar-se que o esforço computacional para processar pequeno número de partições pode ser menor do que o custo de comunicação envolvido no processamento do algoritmo.

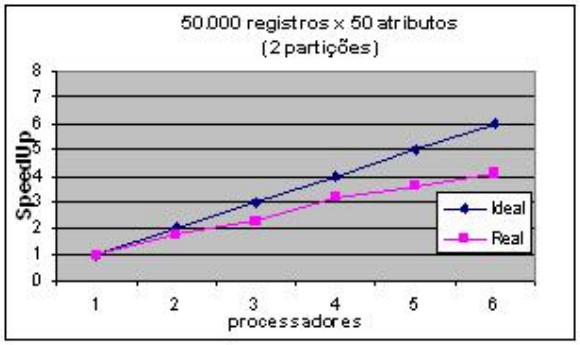

Figura 51 - Curva de *speedup* relativa ao processamento de partição de 2 agrupamentos.

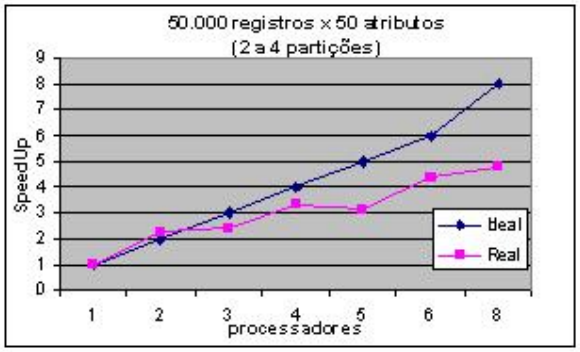

Figura 52 - Curva de *speedup* relativa ao processamento do intervalo de partição de 2 a 4 partições.

Tamanho do problema versus benefícios do paralelismo

Para avaliar de forma completa se é eficiente usar o algoritmo paralelo proposto, todos os duzentos e oitenta e oito testes realizados na máquina Altix foram usados como

registros de entrada para uma árvore de decisão de um algoritmo de indução de classificação.

As variáveis de entrada para o algoritmo de árvore de decisão foram o número de registros, o número de variáveis e o número de partições. A saída do algoritmo foi classificada pela eficiência do processo paralelo, onde eficiência é definida como a razão entre o valor de *speedup* e o número de processadores.

Para cada registro de teste se a eficiência era maior do que 0,8 a saída do teste era **SIM**, o que significa que o processamento paralelo foi eficiente, e caso contrário a classificação era **NÃO**.

Para esta análise, foi usado o algoritmo de árvore de decisão por indução J48, do Weka Open Source Data Mining Workbench [60]. O algoritmo J48 é uma implementação Java do clássico algoritmo C4.5, desenvolvido por Quinlan [61].

A árvore de decisão resultante é apresentada na Figura 53, onde fica muito claro que a aplicação do algoritmo paralelo de análise de agrupamentos FCM não é eficiente para 2 partições e que nos intervalos de 2 a 4 partições a eficiência do processamento depende do número de registros e/ou do número de variáveis envolvidos.

No entanto, o algoritmo é eficiente no processamento dos maiores intervalos de partições, onde os custos computacionais superam os custos de comunicação do algoritmo.

Embora, não seja uma resposta definitiva para a otimização da carga da análise de agrupamentos FCM, a árvore de decisão mostra que o número de partições é o parâmetro que mais afeta a eficiência do processamento paralelo.

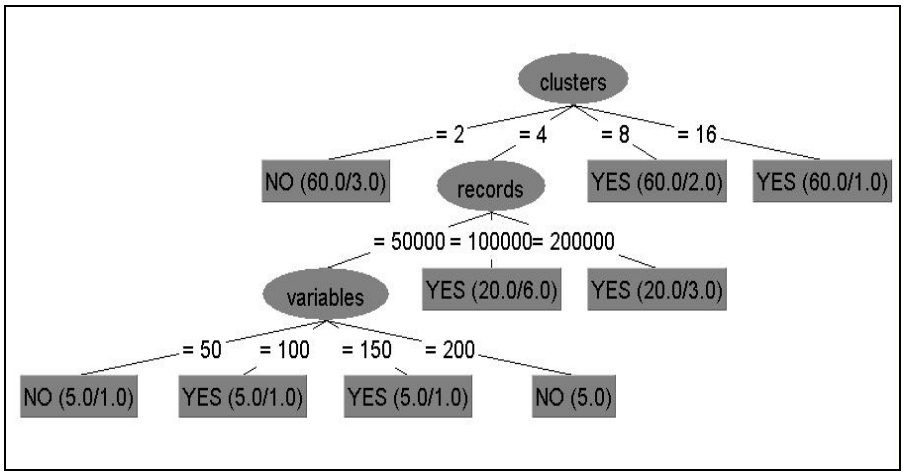

Figura 53 – Árvore de decisão J48 do Weka.

Embora, o algoritmo não apresente bons valores de eficiência para o processamento de pequenos intervalos de partições ou no processamento de arquivos pequenos, a utilização de processamento paralelo na análise de agrupamentos está justamente preocupada com a viabilidade do processamento de grandes bases de dados e com a investigação de grande número de padrões.

O algoritmo proposto deve prioritariamente atingir a esse objetivo, pois bases menores ou pequenos intervalos de padrões não encontram dificuldade considerável em serem investigados pelos algoritmos seqüenciais existentes.

Dentro dessa abordagem, podemos dizer que a etapa de testes realizada na máquina Altix Saturno do NACAD valida a utilização do algoritmo paralelo de análise de agrupamentos FCM para o processamento de grandes intervalos de partições e para o processamento de grandes volumes de dados.

6.3.2 Comparação das abordagens com e sem balanceamento de Carga

Propósito dos testes: Comparar as duas abordagens paralelas de análise de agrupamentos FCM, com e sem balanceamento de carga.

Máquina utilizada: Máquina Altix 450 Vênus do Núcleo de Alto Desempenho da UFRJ – NACAD.

Dados de teste:

Sintéticos

Foram gerados quatro arquivos para os testes. Os dados de cada arquivo foram obtidos a partir do arquivo de estudo de caso referente aos levantamentos sísmicos realizados no campo de óleo e gás de Marlim pela Petrobras.

Todos os arquivos sintéticos foram gerados com sete atributos e, com respectivamente, mil, dez mil, cem mil e um milhão de registros (Tabela 11). Todos os atributos foram normalizados no intervalo -1 a 1 com 6 casas decimais.

Tabela 11 - Tamanho dos arquivos usados nos testes de comparação.

Arquivos de Teste	
Registros	**Tamanho dos arquivos (Kb)**
1.000.000	**77.853**
100.000	**7.785**
10.000	**778**
1.000	**77**

Reais

O arquivo de estudo de caso foi gerado a partir de sete atributos obtidos de medições sísmicas realizadas pela Petrobras no campo de óleo e gás de Marlim, localizado na parte noroeste da Bacia de Campos, a cerca de 110km offshore do Rio de Janeiro, Brasil, em águas cuja profundidade varia de 650m a 1.050 metros. O arquivo referente ao levantamento sísmico do campo de Marlim foi gerado com dois milhões, setecentas e sessenta e seis mil e oitocentas e vinte cinco registros.

Os atributos sísmicos utilizados foram: amplitude, fase, cosseno da fase, energia, envelope, transformada de Hilbert e similaridade.

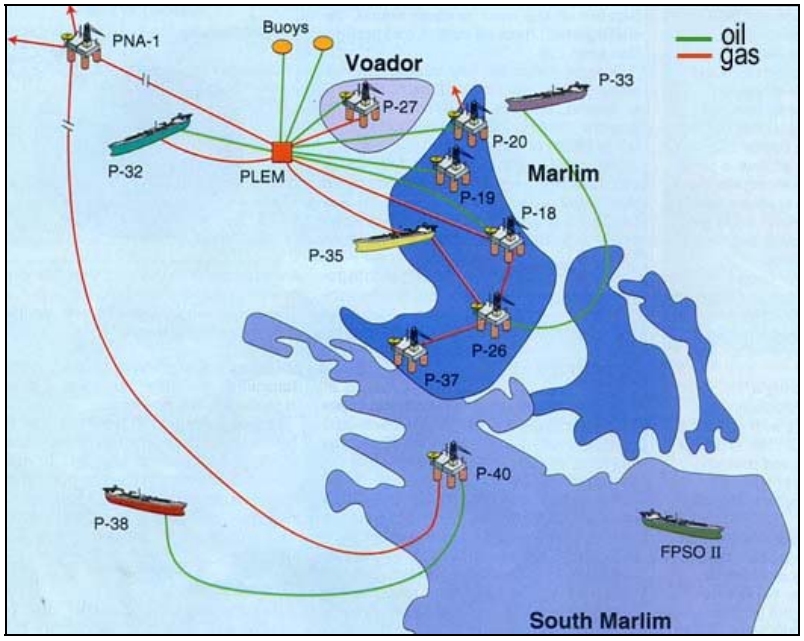

Figura 54 - Estágio final de desenvolvimento do campo de Marlim [62].

Descrição dos testes:

Cada arquivo de teste foi processado para os intervalos de 2 a 4 partições, 2 a 8 partições, 2 a 16 partições e 2 a 32 partições. Os coeficientes de variação utilizados para avaliar a qualidade das mochilas foram de20%, 10% e 3%.

Resultados:

O algoritmo de análise de agrupamentos com balanceamento de carga apresentou tempos consideravelmente melhores dos apresentados pelo algoritmo sem balanceamento de carga para os intervalos de partições de 2 a 8, 2 a 16 e 2 a 32 partições.

Os tempos obtidos pelos dois algoritmos no processamento do arquivo com um milhão de registros são apresentados na Tabela 12.

Tabela 12 - Tempos de processamento do arquivo de um milhão de linhas para diferentes intervalos de partições.

Processadores	Intervalo das Partições / Tempo (segundos)			
Algoritmo Balanceado	*2 a 4*	*2 a 8*	*2 a 16*	*2 a 32*
1	62,576763	238,519065	957,624869	4.042,916829
4	27,659887	65,121103	246,093375	1.026,675303
8	13,440102	46,530077	132,095710	517,987720
12	9,103440	34,832853	91,230101	352,376751
Algoritmo não Balanceado				
1	119,448050	475,618360	1.794,868058	7.350,881991
4	30,383583	114,260539	451,347466	1.837,258185
8	15,636881	57,642829	224,130619	909,555979
12	10,358894	38,253603	148,851859	616,004967

O algoritmo balanceado reduziu o tempo de processamento em todos os testes. Os tempos obtidos pelos dois algoritmos no processamento de todos os arquivos de teste para o intervalo de partições de 2 a 32 são apresentados a seguir na Tabela 13:

Tabela 13 - Tempos de processamento dos arquivos de teste para o intervalo de 2 a 32 partições.

Tempos de processamento (segundos)			
Registros	Procs	Balanceado	Sem Balanceamento
2.766.825	1	23.156,928192	132.370,617975
	4	6.112,632906	32.749,534511
	8	3.048,806565	16.349,538118
	12	2.068,782174	10.775,512474
1.000.000	1	4.042,916829	7.315,806734
	4	1.026,675303	1.862,821484
	8	517,987720	915,709679
	12	352,376751	616,004967
100.000	1	399,641707	730,246971
	4	106,576624	185,351616
	8	51,887652	90,021243
	12	33,439068	60,441903
10.000	1	38,809928	72,326372
	4	10,144735	18,067765
	8	5,153679	9,061798
	12	3,494762	6,065369
1.000	1	20,837336	39,300354
	4	5,385984	8,589276
	8	2,863517	5,248012
	12	0,735913	3,823965

O algoritmo de análise de agrupamentos FCM com balanceamento de carga apresenta uma redução do tempo de processamento considerável na maioria dos testes realizados. No processamento do arquivo de um milhão de registros a redução do tempo de processamento chega a 50% em alguns casos (Tabela 14) quando comparado ao tempo que leva o processamento do mesmo arquivo com o algoritmo sem balanceamento. No processamento do arquivo do campo de óleo e gás de Marlim foram obtidos os maiores índices de redução do tempo de processamento conforme apresentado na Tabela 16, com a redução do tempo de processamento em mais de 80% no processamento do intervalo de 2 a 32 partições.

Os melhores índices de redução dos tempos de processamento do algoritmo balanceado em relação ao algoritmo não balanceado acontecem quando a proporção entre o número de partições e o número de processadores alcança os maiores valores. Por outro lado, quando essa proporção diminuiu, a taxa de redução do tempo de processamento também diminui (Tabela 15). Este fato demonstra que a comunicação é o maior obstáculo para a obtenção do bom desempenho do algoritmo.

Podemos, então, concluir que a vantagem no desempenho do algoritmo com balanceamento de carga varia de acordo com os valores de proporcionalidade entre o número de partições e número de processadores. Razões maiores que 1 produzem ganhos a partir de 40%, na maioria dos processamentos. Conforme a razão vai diminuindo, vai decrescendo também o ganho da abordagem balanceada em comparação com a sem balanceamento. Para valores de razão menores que 1 o algoritmo balanceado passa a apresentar pequena vantagem em relação a abordagem de divisão de dados.

Tabela 14 - Percentuais de redução do tempo de processamento obtidos pelo algoritmo com balanceamento quando comparado ao algoritmo sem balanceamento no processamento do arquivo de um milhão de registros.

Percentual de redução do tempo de processamento				
Procs	Intervalo de Partições			
	4	8	16	32
1	47,61%	49,85%	46,65%	45,00%
4	8,96%	43,11%	45,48%	44,12%
8	14,05%	19,28%	41,11%	43,05%
12	12,12%	16,78%	39,54%	42,80%

Tabela 15 – Proporção entre o número de partições e o número de processadores.

Razão entre no de Partições e no de Processadores				
No procs	Número de partições			
	3	7	15	31
1	3,00	7,00	15,00	31,00
4	0,75	1,75	3,75	7,75
8	0,38	0,88	1,88	3,88
12	0,25	0,58	1,25	2,58

Essa mesma tendência se confirmou no processamento de todos os arquivos de teste independente do tamanho de cada arquivo.

Tabela 16 - Tempos de processamento do arquivo do campo de Marlim para diferentes intervalos de partições.

Processadores	Intervalo das Partições / Tempo (segundos)			
Algoritmo Balanceado	*2 a 4*	*2 a 8*	*2 a 16*	*2 a 32*
1	258,049520	1.269,963524	5.436,060132	23.156,928192
4	98,190133	380,945057	1.409,189242	6.112,632906
8	55,090709	217,804037	760,719544	3.048,806565
12	37,117348	166,185060	531,053735	2.068,782174
Algoritmo não Balanceado				
1	798,056355	3.026,873653	25.339,413827	132.370,617975
4	184,605210	975,010745	6.242,373216	32.749,534511
8	73,415045	453,216679	3.137,309552	16.349,538118
12	40,655708	334,850554	2.298,532210	10.775,512474

Tabela 17 - Percentuais de redução do tempo de processamento obtidos pelo algoritmo com balanceamento quando comparado ao algoritmo sem balanceamento no processamento do arquivo com dados do campo de óleo e gás de Marlim.

Percentual de redução do tempo de processamento				
Procs	Intervalo de Partições			
	4	8	16	32
1	67,67%	58,04%	78,55%	82,51%
4	46,81%	60,93%	77,43%	81,34%
8	24,96%	51,94%	75,75%	81,35%
12	8,70%	50,37%	76,90%	80,80%

O fato do arquivo com os dados do campo de Marlim ter 2.766.825 (dois milhões, setecentos e sessenta e seis mil, e oitocentos e vinte e cinco) registros permite avaliar o comportamento dos algoritmos em situações práticas de processamento de grandes volumes de dados.

De uma forma geral, as curvas resultantes dos tempos obtidos pelos dois algoritmos no processamento dos dados do campo de óleo e gás de Marlin foram proporcionalmente iguais às curvas dos tempos obtidos no processamento dos arquivos sintéticos com menor número de registros. Observa-se que os tempos obtidos pelo algoritmo de análise de agrupamentos FCM balanceado são significativamente menores do que os tempos obtidos pelo algoritmo sem balanceamento nos testes onde a proporção entre o número de partições e processadores é maior do que 1.

Análise de *Speedup* e Eficiência do Algoritmo Balanceado:

Os valores de *speedup* e eficiência do processamento do arquivo de um milhão de registros demonstram que o algoritmo balanceado é escalável para um número crescente de processadores apresentando bons valores de *speedup* e eficiência para diferentes intervalos de partições (Figura 55 e Figura 56).

Os melhores valores de *speedup* e eficiência são encontrados no processamento do maior intervalo de partições, de 2 a 32 partições. Isso demonstra que o algoritmo apresenta uma redução significativa do tempo de processamento quando está sendo investigado um grande número de partições.

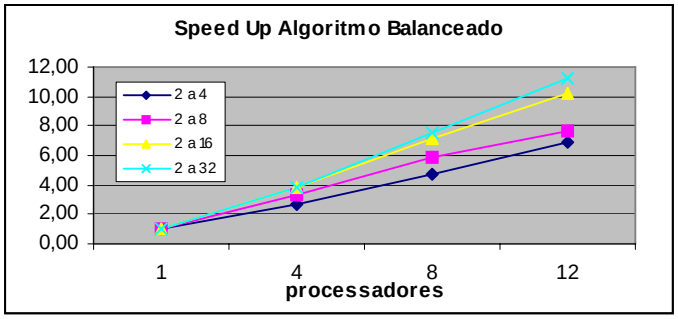

Figura 55 - Curvas de *speedup* do processamento do arquivo de um milhão de linhas pelo algoritmo balanceado.

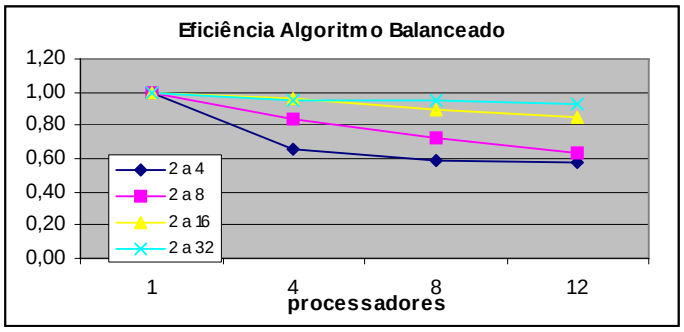

Figura 56 - Curvas de eficiência do processamento do arquivo de um milhão de linhas pelo algoritmo balanceado.

Análise de *Speedup* e Eficiência do Algoritmo não Balanceado:

Embora esta análise já houvesse sido feita na etapa de teste anterior para o algoritmo, ela é apresentada aqui para confirmação do comportamento do algoritmo na máquina diferente e com outros dados. Conforme apresentado abaixo o algoritmo se comporta da mesma forma apresentada na etapa anterior de testes. A seguir são descritas as situações observadas.

As curvas de *speedup* do algoritmo sem balanceamento apresentam valores semelhantes independente do tamanho do arquivo processado.

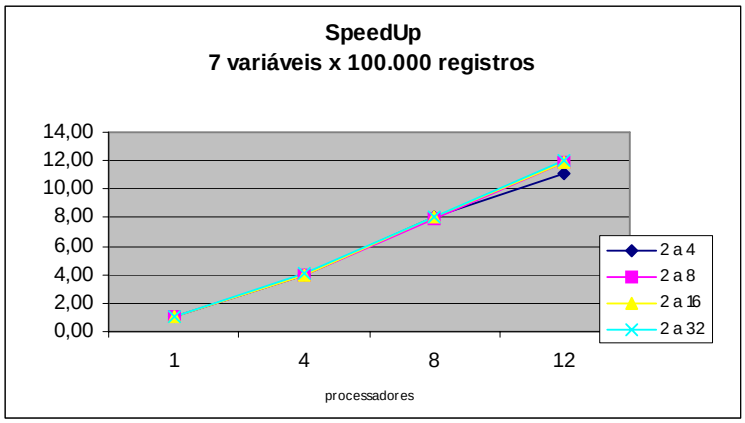

Figura 57 - Curvas de *speedup* do processamento do arquivo de cem mil linhas pelo algoritmo sem balanceamento.

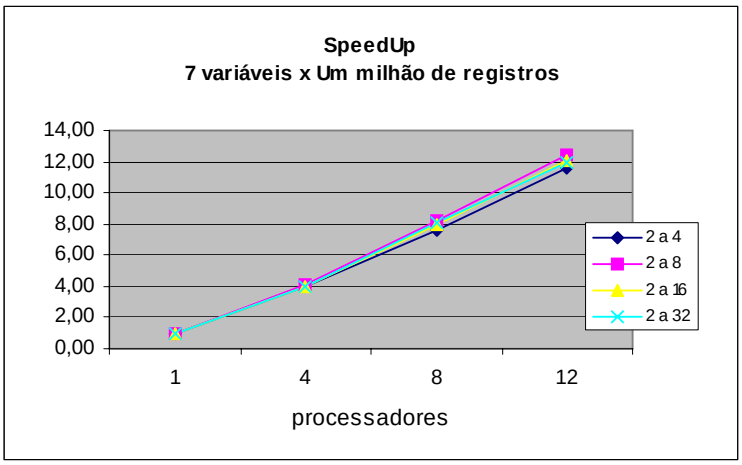

Figura 58 – Curvas de *speedup* do processamento do arquivo de um milhão de linhas pelo algoritmo sem balanceamento.

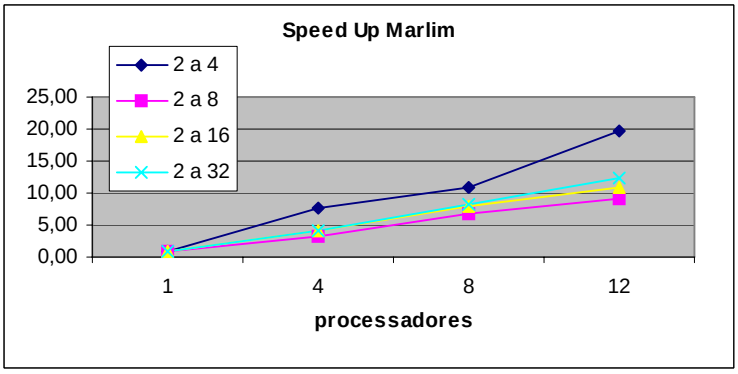

Figura 59 - Curvas de *speedup* do processamento do arquivo de Marlim pelo algoritmo sem balanceamento.

As curvas de eficiência apresentadas pelo algoritmo nos processamentos de diferentes arquivos de testes tendem a apresentar valores superlineares, conforme já havia acontecido nos testes realizados no Altix Saturno 350 na etapa de teste anterior e apresentado na Figura 60 e Figura 61 e para o processamento de diferentes arquivos de testes.

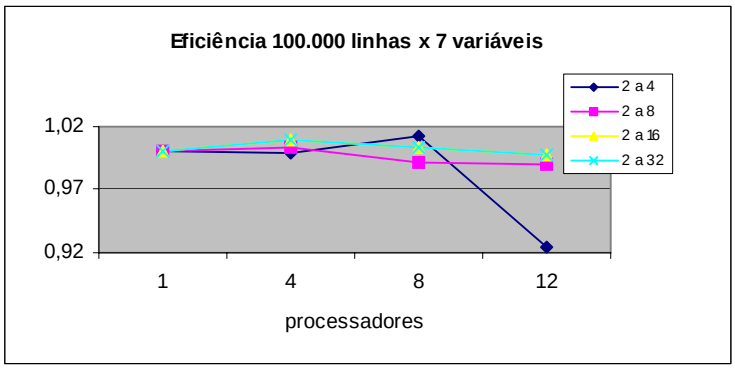

Figura 60 – Curvas de eficiência do processamento do arquivo de cem mil linhas pelo algoritmo sem balanceamento. Valores Superlineares.

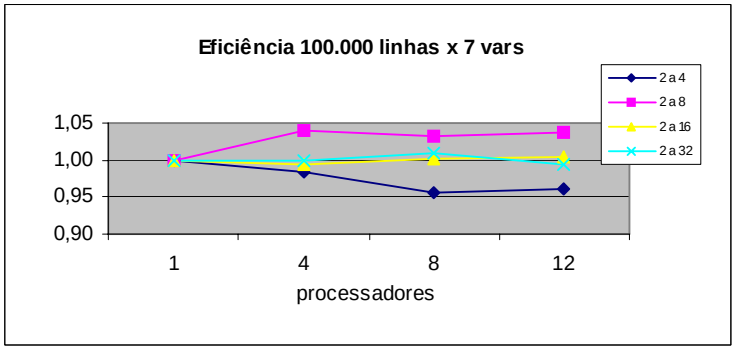

Figura 61 – Curvas de eficiência do processamento do arquivo de um milhão de linhas pelo algoritmo sem balanceamento. Valores Superlineares.

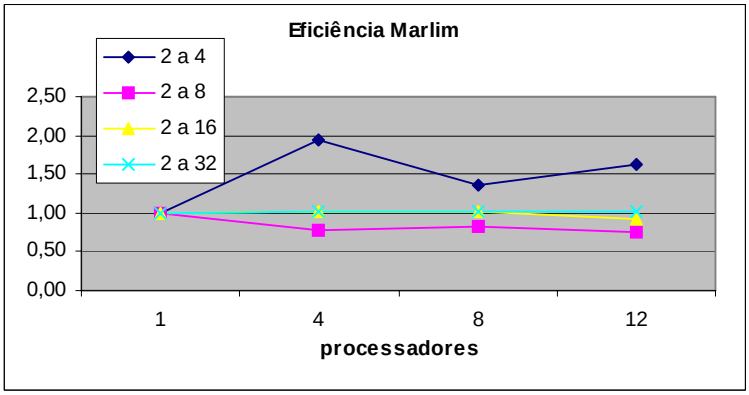

Figura 62 - Curvas de eficiência do processamento do arquivo de um milhão de linhas pelo algoritmo sem balanceamento. Valores Superlineares.

Valores de eficiência superlineares podem acontecer em casos onde o arquivo de entrada seja dividido por vários processadores e passe a caber nas memórias locais dos processadores causando uma melhoria de desempenho em relação ao processamento do arquivo inteiro por um único processador. Mas podemos perceber que no caso do

processamento do arquivo de estudo de caso de Marlim, isso não procede, visto o arquivo ter uma quantidade de quase três milhões de registros e totalizam mais de 400MB.

Outra razão pela qual podem acontecer valores superlineares pode ser a ineficiência do algoritmo seqüencial utilizado para comparação no cálculo da eficiência.

Comparando-se os tempos de processamento dos dois algoritmos com apenas um processador, pode-se observar que o tempo apresentado pelo algoritmo balanceado é consideravelmente melhor, o que significa que o paralelismo envolvido no algoritmo sem balanceamento (divisão de dados) acrescenta um peso considerável sobre seu processamento por um único processador.

A fim de permitir uma comparação melhor entre as duas abordagens, os valores de que eficiência absolutos no processamento dos dados do campo de óleo e gás de Marlim pelos dois algoritmos foram plotados e são apresentadas nas Figura 63 e Figura 64.

Comparação de *Speedup* e Eficiência absolutos dos algoritmos:

Os valores de *speedup* dos dois algoritmos foram obtidos a partir do menor tempo de processamento seqüencial obtido, ou seja, o menor tempo de processamento seqüencial conhecido, que foi o do algoritmo com balanceamento de carga. Portanto, as curvas de *speedup* do algoritmo não balanceado foram calculadas a partir do tempo de processamento sequencial do algoritmo balanceado, pois este apresentava um tempo de processamento seqüencial menor.

As curvas de *speedup* e eficiência do algoritmo balanceado apresentam comportamento oposto aos das curvas do algoritmo sem balanceamento, ou seja, as curvas dos dois algoritmos demonstram seus melhores valores em condições contrárias, tanto nos gráficos de *speedup* quanto de eficiência.

O algoritmo balanceado apresenta os melhores valores de *speedup* quando processa os maiores intervalos de partições revelando que a estratégia de minimizar o custo de comunicação é uma forma bastante efetiva de reduzir o tempo do processamento paralelo. O algoritmo sem balanceamento de carga apresenta melhores valores de *speedup* para menores intervalos de partições, demonstrando o peso do custo de comunicação embutido nesta solução.

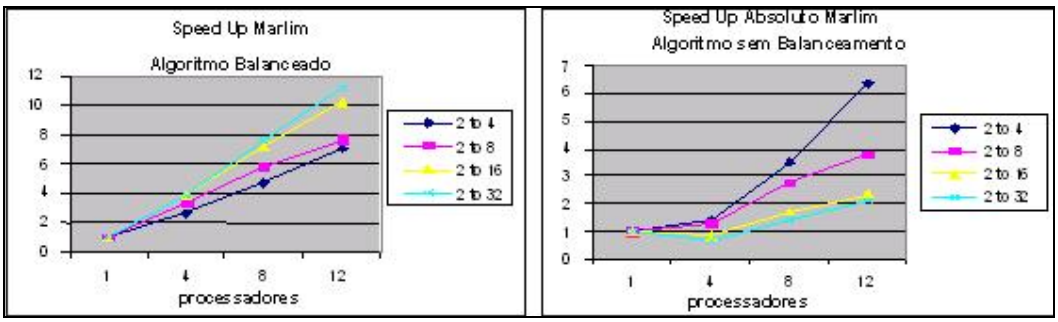

Figura 63 - Curvas de *speedup* para o processamento dos dados do campo de óleo e gás de Marlim para diferentes intervalos de partições pelos dois algoritmos.

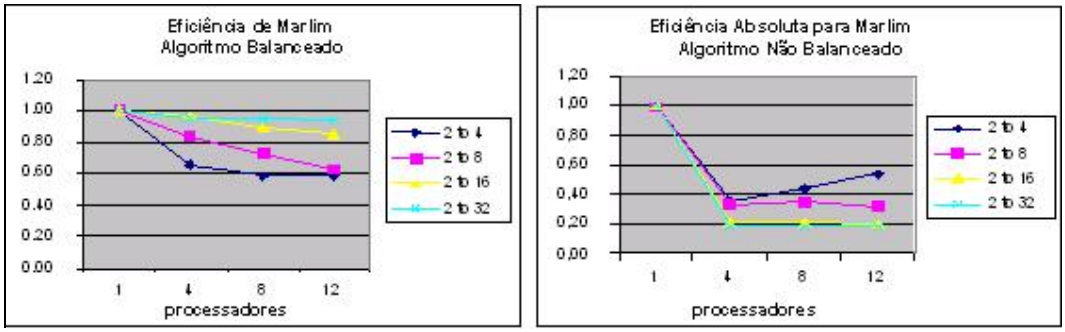

Figura 64 - Curvas de eficiência para o processamento dos dados do campo de óleo e gás de Marlim para diferentes intervalos de partições pelos dois algoritmos.

Os resultados mostram que a eficiência do algoritmo balanceado cai de acordo com o declínio da proporção entre partições e processadores e que quando essa proporcionalidade diminui a diferença entre os valores apresentados pelas duas abordagens também é reduzida.

Analisando os resultados dos testes pela proporcionalidade entre número de partições e número de processadores, pode-se constatar que a estratégia do algoritmo balanceado apresenta as melhores curvas de *speedup* quando a relação entre o número de partições e o número de processadores está acima de 1, pois nesse caso ocorrem os melhores valores de *speedup* e eficiência. Quando essa proporção fica abaixo desse valor, embora o algoritmo ainda apresente uma redução do tempo de processamento dos arquivos de teste, os valores de *speedup* e eficiência são menores.

O ganho da abordagem balanceada sobre a abordagem não balanceada pode ser compreendido pelo fato do primeiro algoritmo ter um custo de comunicação muito menor.

A abordagem balanceada usa o poder computacional disponível para minimizar ao máximo os custos de comunicação entre processadores, mas quando não há como melhorar os tempos de execução do processamento paralelo dessa forma, ela lança mão da abordagem de divisão de dados para continuar diminuindo o tempo total do processamento paralelo. Nesse último caso, as duas abordagens passam a ter tempos de execução muito mais próximos.

Quanto a análise da qualidade física dos resultados, seria necessário contar com o apoio de um especialista em geofísica para sua interpretação, pois a análise do significado das partições realizadas pelo algoritmo no processamento dos atributos sísmicos depende de conhecimento especializado que permita avaliar as partições geradas e seu significado no contexto da exploração de bacias sedimentares em busca de reservatórios através de medições sísmicas.

7 Conclusão

Esse trabalho apresentou uma estratégia paralela para a análise de agrupamentos FCM onde a determinação dos centros dos agrupamentos e do número de padrões estão integrados.

A principal contribuição deste trabalho é a da integração do índice de validação de partições PBM no processo de análise de agrupamentos FCM permitindo a otimização geral do processo paralelo.

Questões inerentes à otimização do processamento paralelo motivaram a proposição de uma variação da abordagem original, onde uma estratégia complementar de paralelismo é acrescida ao algoritmo original e implementada segundo uma política de balanceamento de carga modelada pelo problema do empacotamento.

As estratégias propostas visam o processamento de grandes volumes de dados onde o tempo de processamento dos algoritmos seqüenciais existentes pode ser inviável, e que consiste em desafios enfrentados por diversos segmentos industriais, tais como a indústria de óleo e gás, dentre outras.

Para avaliar a aplicabilidade dos algoritmos propostos para o processamento de grandes volumes de dados foram realizados testes processando arquivos sintéticos de diferentes dimensões e um estudo de caso com dados de medições sísmicas feitas pela Petrobras no campo de óleo e gás de Marlim.

A primeira estratégia paralela foi implementada pela divisão do conjunto de dados pelos processadores paralelizando o código de agrupamento FCM. A segunda estratégia otimiza a primeira pelo acréscimo de mais um nível de paralelismo ao algoritmo original onde as partições são distribuídas por uma política de balanceamento de carga modelada pelo problema do empacotamento e são preferencialmente processadas de forma independente por diferentes processadores, mas em determinadas situações para manter a escalabilidade do algoritmo podem ser processadas por um grupo de processadores quando há suficiente poder computacional disponível.

As principais conclusões encontradas a partir dos resultados dos testes dos algoritmos são as seguintes:

1. O algoritmo paralelo de análise de agrupamentos FCM por divisão de dados reduz os tempos de processamento dos conjuntos de dados testados em relação a um mesmo processamento seqüencial, mas apresenta um alto custo de comunicação. A estratégia de sincronizar processadores a cada iteração FCM é responsável por esse custo. Seria interessante comparar esta solução com outros algoritmos particionais paralelos que adotaram paralelismo por divisão de dados, como a de consolidar os centros de agrupamentos apenas ao final do processamento de cada partição [12], ou a da divisão dos atributos do conjunto de dados ao invés da divisão do número de registros [7], dentre outras. Essas estratégias, embora se refiram apenas à implementação paralela de um algoritmo de mineração de dados particional, poderiam ser integradas ao índice PBM de avaliação de partições para avaliação dentro do contexto da análise de agrupamentos.

2. Em relação à estratégia de balanceamento da carga do processamento paralelo da análise de agrupamentos FCM, a divisão das partições por processadores, modelada pelo problema do empacotamento, se apresenta como uma alternativa bem eficiente para acrescentar eficiência ao processamento paralelo. O algoritmo projetado segundo esta estratégia apresenta resultados bem melhores do que os do algoritmo da estratégia da divisão de dados. Resultados bastante significativos acontecem especialmente quando o intervalo de partições apresenta boa proporcionalidade em relação ao número de processadores. O algoritmo é realmente eficiente nessas situações apresentando excelente desempenho. Nos casos em que o número de partições não apresenta uma proporcionalidade razoável em relação ao número de processadores essa vantagem não se apresenta da mesma forma, pois o acréscimo de mais processadores não consegue escalar adequadamente para usufruir o poder computacional disponível.

O que se conclui é que no processamento paralelo o custo de comunicação é sempre um ponto a ser minimizado devido ao seu alto impacto sobre qualquer algoritmo.

A política de balanceamento de carga utilizada nesse trabalho é um tipo de solução que pode ser aplicada a diferentes situações onde a investigação de um problema deve ser realizada para diferentes cenários. A estratégia funciona por minimizar os custos de comunicação e se houver poder computacional disponível ela se associada com sucesso a outro tipo de paralelismo para continuar a aumentar a eficiência do processamento paralelo e superar seu limite natural de escalabilidade. Neste trabalho, o outro nível de

paralelismo implementado foi o da estratégia de divisão do conjunto de dados por processadores, a qual cabem as considerações feitas no item anterior.

Alguns pontos sobre o contexto onde a estratégia de balanceamento de carga foi utilizada devem ser, no entanto, observados:

Intervalos de partições contínuas variando de uma unidade

Nos testes dos algoritmos foram sempre usados intervalos contínuos de partições, ou seja, as partições investigadas variavam de 2 a *N* agrupamentos com a diferença de uma unidade entre uma partição e sua sucessora. Para que o algoritmo possa avaliar um número não contínuo de partições, ou possa avaliar intervalos onde haja partições que variem em mais de uma unidade ou que variem de forma não uniforme, o algoritmo deve ser adaptado e testes específicos para estas situações devem ser realizados.

Validação do algoritmo para pequenos intervalos de partições

O fato de serem sempre avaliados intervalos contínuos de partições onde os menores intervalos eram compostos por partições com um número pequeno de agrupamentos, como, por exemplo, o intervalo de 2 a 4 partições, não permitiu avaliar com muita precisão o comportamento do algoritmo balanceado quando as duas estratégias (divisão de partições e divisão de dados) estivessem coexistindo, pois vários processadores começaram a ser alocados no processamento de partições com um número muito pequeno de agrupamentos, menos do que 4, situação em que o algoritmo de divisão de dados não apresenta ganhos significativos. Esse fato na prática aproxima os valores obtidos pelas duas estratégias, mas não permite quantificar corretamente quanto se ganha em cada situação.

Para validar melhor esse ponto seria necessário realizar testes em intervalos com poucas partições, mas com maior número de agrupamentos. Por exemplo, processar com 6 processadores um intervalo de partições de 30 a 32 partições pelos dois algoritmos.

Considerar o valor das cargas na distribuição do poder computacional disponível

Outra situação que seria interessante testar seria a de alocar processadores ociosos de forma proporcional ao valor das cargas. Assim, ao invés de distribuir os processadores na razão de um processador para cada carga por ordem decrescente de valor das cargas, o número de processadores que cada carga receberia seria proporcional

ao seu peso na distribuição. Esse tipo de distribuição garantiria uma melhor distribuição do poder computacional e balancearia melhor o processamento paralelo.

Trabalhos futuros

A análise de diferentes estratégias de paralelismo permite perceber a existência de premissas básicas que norteiam o projeto de algoritmos paralelos, independente do tipo de problema que esteja sendo resolvido. Um estudo futuro que poderia acrescentar valor a esse tipo de pesquisa seria o da proposição de um paralelizador de algoritmos particionais de mineração de dados. A idéia central desse trabalho seria considerar as variáveis envolvidas no paralelismo de um problema, tais como, atributos, registros, cenários de investigação e o código do algoritmo seqüencial A partir de um grupo básico de premissas, tais como minimização do custo de comunicação do processamento paralelo, garantia da escalabilidade do algoritmo paralelo e utilização eficiente dos recursos computacionais, dentre outros, seria proposto um algoritmo paralelo equivalente ao seqüencial com seus pontos de sincronismo definidos.

Considerações Finais

De uma forma geral, os algoritmos de análise de agrupamentos FCM propostos são um avanço no estudo do emprego de técnicas de processamento paralelo para mineração de grandes volumes de dados. A partir do tratamento integrado dos dois desafios da análise de agrupamentos eles respondem à questão da análise de agrupamentos FCM identificando o número de agrupamentos mais natural para um conjunto de dados tratando esse problema em uma única etapa de automação.

As questões de otimização que surgem na investigação da forma de paralelismo mais eficiente para o tratamento do problema da análise de agrupamentos FCM são também uma reflexão importante para a utilização de processamento paralelo no processamento de grandes volumes de informação. A partir de estudos realizados com diferentes estratégias de paralelismo para diferentes algoritmos de mineração de dados podem emergir diretrizes para a utilização desse tipo de solução na resolução de problemas de aquisição do conhecimento.

Os algoritmos apresentados, especialmente o algoritmo otimizado, demonstram boa escalabilidade para a maioria dos cenários investigados e agregam desempenho ao processamento de grandes volumes de dados com redução significativa do tempo de processamento nas diferentes máquinas utilizadas demonstrando que podem ser usados como uma ferramenta valiosa e eficiente para a descoberta de conhecimento em grandes bases de dados.

8 Referências

[1] Sousa, M. S. R., Mattoso, M., Ebecken, N. F.F., 1999, "Mining a large database with a parallel database server". I*ntelligent Data Analysis* 3, pp. 437-451.

[2] Coppola, M., Vanneschi, M., 2002, "High-performance data mining with skeleton-based structured parallel programming". *Parallel Computing*, 28, pp. 783-813.

[3] Jin, R.,Yang, G., Agrawal G., 2005, "Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance". *IEEE Transaction on Knowledge and Data Engineering*, vol. 17, no. 1, pp. 71-89.

[4] Cannataro, M., Congiusta, A., Pugliese, A., et al., 2004, "Distributed data mining on grids: services, tools, and applications". *IEEE Transactions on Systems, Man and Cybernetics, Part B*, vol. 34, no. 6, pp. 2451 – 2465.

[5] Kubota, K., Nakase, A., Sakai, H., et al, 2000, "Parallelization of decision tree algorithm and its performance evaluation". *Proceedings of the Fourth International Conference on High Performance Computing in the Asia-Pacific Region*, vol. 2, pp. 574 – 579.

[6] Kim, M. W., Lee, J. G., Min, C., 1999, "Efficient fuzzy rule generation based on fuzzy decision tree for data mining". *Proceedings of the IEEE International Fuzzy Systems Conference - FUZZ-IEEE '99*. pp. 1223 – 1228.

[7] Evsukoff, A., Costa, M.C. A., Ebecken, N.F. F., 2004, "Parallel Implementation of Fuzzy Rule Based Classifier". *Proceedings of the VECPAR'2004*, vol. 2, pp. 443-452.

[8] Phua, P. K. H., Ming, D., 2003, "Parallel nonlinear optimization techniques for training neural networks". *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1460 - 1468.

[9] Costa, M. C. A., Ebecken, N. F. F., 2001, "A Neural Network Implementation for Data Mining High Performance Computing". *Proceedings of the V Brazilian Conference on Neural Networks*, pp. 139-142.

[10] Agrawal, R., Shafer, J. C., 1996, "Parallel mining of association rules". *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 6, pp. 962 - 969.

[11] Shen, L., Shen, H., Cheng, L., 1999, "New algorithms for efficient mining of association rules". *Information Sciences* 118, pp. 251 – 268.

[12] Boutsinas, B., Gnardellis, T., 2002, "On distributing the clustering process". *Pattern Recognition Letters* 23, pp. 999–1008.

[13] Rahimi, S., Zargham, M., Thakre, A., et al, 2004, "A parallel Fuzzy C-Mean algorithm for image segmentation". *Proceedings of the IEEE Annual Meeting of the Fuzzy Information NAFIPS '04*, v. 1, pp. 234 – 237.

[14] MacQueen, J. B., 1967, "Some Methods for classification and Analysis of Multivariate Observations". *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, Berkeley, University of California Press, v. 1, pp. 281-297.

[15] Jain, A. K., Dubes, R. C., 1988, *Algorithms for Clustering Data*, Prentice Hall.

[16] Jain, A. K., Murty, M.N., Flynn, P.J., 1999, "Data clustering: a review". *ACM Computing Surveys*, v. 31, no. 3. pp. 264-323.

[17] Anderberg, M., 1973, *Cluster Analysis for Applications*. NewYork, Academic Press.

[18] Ester, M. et al, 1996, "A density-based algorithm for discovering clusters in large spatial databases with noise". *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Minig*, p. 226-231.

[19] Bezdek, J.C., 1981, *Pattern Recognition with Fuzzy Objective Function Algorithm*. New York, Plenum.

[20] Xie, X. L., Beni, G. A., 1991, "Validity measure for fuzzy clustering". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 3 no. 8, pp. 841-846.

[21] Bezdek, J., Pal, N.R., 1998, "Some new indexes of cluster validity". *IEEE Trans. Systems Man and Cybernetics B*, vol. 28, pp. 301–315.

[22] Pakhira, M. K., Bandyopadhyay, S., Maulik, U., 2004. "Validity index for crisp and fuzzy clusters". *Pattern Recognition*, vol. 37, pp. 487-501.

[23] Dantzig, G.B., 1957, "Discrete Variable Extremum Problems", *Operations Research*, v 5, 266-277.

[24] Mitten, L.G., 1970, "Branch-And-Bound Methods: General Formulation and Properties". *Operations Research*, v. 18, no 1, pp. 24-34.

[25] Martello, S., Toth, P., 1990, *Knapsack Problems, Algorithms and Computer Implementations*. John Wiley & Sons.

[26] Pissinger, D., Toth, P., 1998, "Knapsak problems". *Handbook of Combinatorial Optimization*, Kluwer Academic Publishers, Dordrecht.

[27] Coffman, Jr, E. G., Garey, M. R., Johnson, D. S., 1997, "Approximation algorithms for NP-hard problems", In: *Approximation Algorithms for Bin Packing: A Survey*, PWS Publishing Co, Boston, p. 46-93.

[28] Everitt, B.S., 1974, *Cluster Analysis*, John Wiley & Sons, Inc., New York.

[29] Joshi, M. N., 2003, "Parallel K-Means Algorithm on Distributed Memory Multiprocessors", Computer Science Department, University of Minnesota, Twin Cities.

[30] Arthur, D. e Vassilvitskii, S., 2006, "How Slow is the k-means Method?". *Proceedings of the 2006 Symposium on Computational Geometry (SoCG)*.

[31] Selim, Shokri, Z, Ismail, M.A., 1984, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality". *IEEE Trans. Pattern Anal. Mach. Intell.*, v. 6, 81-87.

[32] Bradley, P.S., Fayyad, U.M., 1983, "Refining initial points for k-means clustering". *Proceedings of the IJCAI-93*, San Mateo, CA, p. 1058-1063.

[33] Bradley, P.S., Fayyad, U.M., Reina, C., 1998, "Scaling clustering algorithms to large databases". *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, p. 9-15.

[34] Provost, F., Kolluri, V.,1997, "Scaling up inductive algorithms: an overview". *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, p.239-242.

[35] Buyya, R., 1999, *High Performance - Cluster Computing*, Prentice Hall.

[36] Foster, I., 1995, *Designing and Building Parallel Programs*, Addison-Wesley, on-line book - http://www-unix.mcs.anl.gov/dbpp/text/book.html - acesso em 29 de abril de 2008.

[37] Hansen, P. B., 1993, "Model Programs for Computational Science: A Programming Methodology for Multicomputers". *Concurrency: Practice and Experience*, vol. 5 (5), p. 407-423.

[38] Silva, L. M., Buyya R., 1999, "Parallel Programming Models and Paradigms". *High Performance Cluster Computing: Programming and Applications*, Rajkumar Buyya (editor), Prentice Hall PTR, NJ, USA.

[39] Quinn, M. J., 1994, *Parallel Computing: Theory and Practice*, McGraw-Hill.

[40] Tanenbaum, A. S., 2001, *Organização Estruturada de Computadores*. Rio de Janeiro, RJ: Livros Técnicos e Científicos Editora.

[41] Paterson, D., Hennessy, J., 1996, *Computer Architecture: a Quantitative Approach,* 2ª edição, Morgan Kaufmann Publishers.

[42] Carvalho, Alexandre Plastino, 2000, *Balanceamento de Carga de Aplicações Paralelas SPMD*, Tese de D.Sc., PUC-Rio, Rio de Janeiro, RJ, Brasil.

[43] Mattson, T.G., 1996, “Scientific Computation”, *In: Parallel and Distributed Computing Handbook*, McGraw-Hill, p. 981-1002.

[44] Casavant, T. L. e Kuhl, J. G., 1988, “A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems”, *IEEE Transactions on Software Engineering*, p. 141-154.

[45] Caprara, A., Kellerer, H. e Pferschy, U., 1998, “The Multiple Subset Sum Problem”. *SIAM Journal on Optimization*, v. 11, Issue 2, p. 308-319.

[46] Johnson, D., 1973, *Near-Optimal Bin Packing Algorithms*. Doctoral Thesis, MIT, Cambridge.

[47] Hoffman, P., 1998, *The Man Who Loved Only Numbers: The Story of Paul Erdos and The Search for Mathematical Truth*. Fourth Estate, London, England.

[48] Yue, M., 1991, “A simple proof of the inequality FFD(L) ≤ (11/9)OPT(L) + 1, for all L, for the FFD bin-packing algorithm". *Acta Mathematicae Applicatae Sinica,* v. 7, p. 321-331.

[49] Garey, M.R., Johnson, D.S., 1985, “A 71/60 theorem for bin packing". *Journal of Complexity*, v.1, p.65-106.

[50] Myiazawa, F.K., “Problemas de Corte e Empacotamento” – site acessado em 17 de fevereiro de 2008, http://www.dcc.unicamp.br/~fkm/empacotamento.html.

[51] Minyi, Y., Lei, Z., 1995, "A simple proof of the inequality MFFD(L) ≤ 71/60 OPT(L) + 1, L for the MFFD bin-packing algorithm", *Acta Mathematicae Applicatae Sinica*, v. 11, p.318-330.

[52] Muller, F. M., 1993, *Algoritmos Heurísticos e Exatos para Resolução do Problema de Sequenciamento em Processadores Paralelos*. Tese D. SC., Universidade Estadual de Campinas. Campinas, SP.

[53] Muller, F. M., Dias, et al., 2002, “Algoritmo para o Problema de Seqüenciamento em Máquinas Paralelas não-relacionadas”, SciELO, Prod. v. 12, no2, São Paulo.

[54] Lawler, E.L., Lenstra, J. K., Rinnooy Kan, et al, 1989, “D.B. *Sequencing and scheduling: algorithms and complexity”*, Report BS-R8909, Center for Mathematics and Computer Science, Amsterdan.

[55] McNaughton, R., 1959, “Scheduling with deadlines and loss functions”. *Management Science*, v. 6(1), p. 1-12.

[56] R.L. Graham, 1966, "Bounds on multiprocessor timing anomalies" *SIAM J. Appl. Math.* , v. 17, p. 416–429.

[57] Coffman Jr., E.G., Garey, M.R., Johnson, D.S., 1978, "An application of bin-packing to multiprocessor scheduling". *SIAM Journal on Computing*, v. 7, p. 1-17.

[58] Alvim, A.C.F., Ribeiro, C. C., 2004, "A Hybrid Bin-Packing Heuristic to Multiprocessor Scheduling", *WEA 2004: International worshop on experimental and efficient algorithms N°3*, Angra dos Reis, BRASIL.

[59] Wisconsin Breast Cancer Database, 8 de Janeiro 1991, University of Wisconsin Hospitals,Madison, Wisconsin, USA. Doador: Olvi Mangasarian. (mangasarian@cs.wisc.edu).

[60] Witten, I. H., Frank, E., 2005, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd Edition, Morgan Kaufmann, San Francisco.

[61] Quinlan, R., 1993, *Programs for Machine Learning*. Morgan Kaufmann, San Francisco.

[62] Site com informações sobre o campo de óleo e gás de Marlin. Acesso em 25 de setembro de 2007. http://www.offshore-technology.com/projects/marlimpetro/