PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Uma Arquitetura para Suporte à Mineração de Dados Paralela e Distribuída em Ambientes de Computação de Alto Desempenho

Élder F. F. Bernardi

Dissertação de Mestrado apresentada como requisito para obtenção do título de Mestre em Ciência da Computação pelo Programa de Pós-graduação da Faculdade de Informática. Área de concentração: Ciência da Computação.

Orientador: Prof. César A. F. De Rose

Porto Alegre, Brasil
2010

Dados Internacionais de Catalogação na Publicação (CIP)

B523a Bernardi, Élder F. F.
Uma arquitetura para suporte à mineração de dados paralela e distribuída em ambientes de computação de alto desempenho / Élder F. F. Bernardi. – Porto Alegre, 2010.
102 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. César A. F. De Rose.

1. Informática. 2. Mineração de Dados (Informática). I. De Rose, César A. F. II. Título.

CDD 005.74

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada **"Uma Arquitetura para Suporte à Mineração de Dados Paralela e Distribuída em Ambientes de Computação de Alto Desempenho"**, apresentada por Élder Francisco Fontana Bernardi, como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 11/03/10 pela Comissão Examinadora:

Prof. Dr. César Augusto Fonticielha De Rose – Orientador — PPGCC/PUCRS

Prof. Dr. Duncan Dubugras Alcoba Ruiz – PPGCC/PUCRS

Prof. Dr. Adenauer Corrêa Yamin – UCPEL

Homologada em. 24./.05../.11....., conforme Ata No. 008..... pela Comissão Coordenadora.

Prof. Dr. Fernando Gehm Moraes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 – P32– sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Para Clarissa e meus pais, Valerio e Eledi.

AGRADECIMENTOS

A conclusão deste trabalho que culmina neste documento teve a participação de diversas pessoas e entidades, as quais permito-me publicamente agradecer e dividir os méritos alcançados.

Primeiramente agradeço ao Universo e sua lógica perfeita que me permitiu estar na hora certa e no lugar certo para me tornar o que sou, graças aos desafios e experiências que tive a oportunidade de viver. Muito Obrigado!

Agradeço à PUCRS por ter sido um espaço acima de tudo de crescimento e amizades e pela contribuição que sempre exercerá para minha formação como ser humano. À CAPES, por ter me agraciado com uma bolsa que permitiu que esse trabalho fosse realizado com a devida segurança financeira, que sempre é importante.

Agradeço especialmente ao meu orientador, o professor De Rose, pelo apoio, orientação e, principalmente, pela confiança em mim depositada ao longo deste trabalho. Confiança a qual muito me honra e orgulha. Agradeço ao professor Duncan pelas "broncas"e conselhos que sempre me motivaram e ensinaram a encontrar o caminho correto. Também deixo expresso o meu agradecimento ao professor Tiago Ferreto, um amigo que sempre esteve disposto a ajudar no que fosse preciso.

São tantos a agradecer – amigos, os colegas de mestrado, de LAD, Paleo, aos papos nos bar do 32 e nos botecos da vida...– que sinto que provavelmente esquecerei de mencionar alguém que mereceria uma menção especial.

No entanto, permitam-me terminar com agradecimentos honrosos e especiais aos meus pais e à minha namorada Clarissa. Pai e mãe obrigado por terem sido o base forte de meu caráter. Tudo que sou e conquisto tenham a certeza de que passa por vocês! Obrigado pelo apoio e incentivo sempre vindos de todas as formas e sempre incondicionais. Amo vocês!

Clarissa! Fofa, se não fosse por ti não haveria nem sequer essa dissertação, literalmente (piada interna). Obrigado pela amizade, amor e companheirismo que me dedicas com tanta paixão a tanto tempo (obrigado por me aturar!). Jamais esquecerei das noites em claro que passaste para permitir que um dia eu fosse mestre. Saibas que serei eternamente grato e te amarei da melhor maneira que minha alma, ainda limitada, puder fazê-lo. Muito obrigado por tudo!

Finalmente, agradeço a você leitor, pelo prestígio em escolher entre tantas outras, esta dissertação para leitura. Obrigado.

Uma Arquitetura para Suporte à Mineração de Dados Paralela e Distribuída em Ambientes de Computação de Alto Desempenho

RESUMO

Este trabalho apresenta uma arquitetura para suporte à execução de tarefas de mineração de dados em ambientes de computação de alto desempenho, tais como: clusters, máquinas SMP e grades. Esta arquitetura automatiza o processo de dimensionamento da aplicação paralela, criando ferramentas para a construção automática de tarefas, mapeamento, gerência e execução dessas aplicações nos recursos computacionais disponíveis. Os mecanismos criados para a execução de aplicações de mineração possibilitam a combinação do paralelismo do fluxo de dados e de instruções. Como contribuição do trabalho, destaca-se a organização da arquitetura proposta e a criação de um algoritmo para mapeamento de aplicações de mineração paralelas em ambientes computacionais heterogêneos. Enfatiza-se o suporte ao aproveitamento de recursos com múltiplos núcleos de processamento (multi-cores). Além disso, apresenta-se a paralelização de um algoritmo de mineração de dados para regressão.

Palavras-chave: Mineração de dados paralela; Escalonamento; Cluster; Multi-core.

An Architecture for Supporting Distributed Parallel Data Mining Applications on High Performance Computing Enviroments

ABSTRACT

In this paper, we present an architecture to support the execution of data mining applications on high performance computing environments such as clusters, SMP and grids. This architecture automates the process of parallel applications sizing, presenting tools for automatic construction of parallel tasks, automatic scheduling, managing and execution of these applications on high performance computing environments. The mechanisms created for executing mining applications make it possible to explore both data and instruction parallelism. The main contributions of this work are the organization of the proposed architecture and the creation of an algorithm for mapping parallel data mining applications on heterogeneous computational environments. The support of multi-core resources is taken on account. Furthermore, we present the parallelization of a data mining algorithm for regression.

Keywords: Parallel Data Mining; Scheduling; Cluster; multi-core.

Lista de Figuras

Lista de Tabelas

SUMÁRIO

1. Introdução

A sociedade contemporânea vive na chamada era da informação, onde cada vez mais se produz dados dos mais variados tipos sobre os mais diversos assuntos. A Web por exemplo, concentra dados que vão desde resultados de pesquisas científicas até diários pessoais. Assim, torna-se cada vez mais importante a existência de ferramentas que possibilitem extrair conhecimento útil dessa panaceia de dados que se acumulam diariamente, pois de nada adianta se ter uma quantidade imensurável de dados e não conseguir extrair desses algum conhecimento útil que possa ser aplicado para o melhoramento dos negócios de uma organização e para tomadas de decisões que possam guiar estratégias futuras. Por isso, os processos de mineração de dados estão cada vez mais requisitados na realidade de muitas organizações, tanto corporativas, quanto científicas.

Com esta realidade, o impacto que soluções de descoberta de conhecimento podem trazer para determinados negócios é cada vez maior e mais importante. A descoberta de informações até então desconhecidas pode ser um fator chave e diferencial para uma organização.

Embora o crescimento da quantidade de dados armazenados possa ampliar as oportunidades de aplicar mineração de dados de forma eficaz, por outro lado demanda a criação de técnicas que permitam que os processos de mineração possam efetivamente manipular de forma adequada grandes bases de dados [36], gerando resultados num tempo adequado para o problema em que se está trabalhando. Processos de mineração geralmente são processos de relativa complexidade [35] que dependendo da técnica aplicada, do problema a ser resolvido e do tamanho da base a ser analisada podem levar mais tempo para gerar resultados do que o aceitável. Por exemplo, para se analisar o comportamento de uma bolsa de valores com o objetivo de guiar investimentos do dia seguinte, o tempo de análise para geração de resultados não pode exceder 24 horas.

Entre as abordagens que visam diminuir o tempo de mineração sem que se precise diminuir a quantidade de dados a ser minerada ou a acurácia dos resultados se encontra a realização do processo de mineração de dados de forma paralela, que identifica fluxos independentes no processo de mineração e os executa simultaneamente [24, 38]. Assim sendo, criou-se inumeráveis soluções de mineração que utilizam essa abordagem, implementadas com as mais diversas tecnologias. No entanto, a aplicação dessas abordagens, em termos computacionais, demanda necessariamente a existência de recursos que permitam que os diferentes fluxos do processo possam ser efetivamente executados paralelamente e que forneçam capacidade computacional capaz de resolver o problema no tempo requisitado. De acordo com Hwang [23], ambientes especialmente designados para tal são denominados Ambientes de Computação de Alto Desempenho.

Inicialmente ambientes de computação de alto desempenho eram relativamente caros e restritos a grandes centros de computação. No entanto, com a popularização dos recursos computacionais e o avanço tecnológico, hoje se tem acesso a recursos computacionais com capacidade razoavelmente elevada a custos acessíveis até mesmo para organizações de pequeno porte. Recursos multiproces-

sados, que antes só eram encontrados em máquinas caríssimas e que demandavam um alto custo de manutenção, estão presentes em todos os níveis de uma organização. Além disso, o custo para se acessar ambientes especializados para computação de alto desempenho está cada vez mais baixo. A combinação desses fatores faz com que as possibilidades de acesso a uma variedade de opções de tipos de recursos de alto desempenho sejam cada vez maiores e mais baratas.

O cenário descrito, no qual existe uma diversidade de aplicações para realizar mineração de dados de forma paralela e distribuída aliada a alta oferta e diversidade de recursos computacionais de alto desempenho, motiva a criação de ferramentas que facilitem e apoiem adequadamente a execução dessas aplicações nesses recursos, levando em conta as peculiaridades de cada aplicação e como essas se adequam aos diferentes tipos de ambientes de computação de alto desempenho disponíveis aos usuários de mineração de dados.

Com essa motivação, este trabalho apresenta um conjunto de ferramentas organizadas por meio de uma arquitetura que visam apoiar e facilitar a execução de aplicações de mineração de dados paralelas e/ou distribuídas em conjuntos heterogêneos de recursos computacionais de alto desempenho. A arquitetura proposta visa permitir que usuários de mineração de dados possam executar suas aplicações nos recursos computacionais que têm disponíveis sem que para isso precisem ter conhecimento técnico avançado sobre o funcionamento desses recursos e possam utilizá-los de forma eficaz e eficiente, de acordo com o tipo de aplicação que possuem, sem que para isso tenham que dominar técnicas de dimensionamento e escalonamento de aplicações paralelas. Dá-se ênfase especial ao suporte de arquiteturas de computação multiprocessadas, que estão cada vez mais presentes e acessíveis e, devido a sua recente popularização, existem relativamente poucas soluções que abordam maneiras de utilizar corretamente esse tipo de recurso.

Para se chegar à definição da arquitetura apresentada neste trabalho, realizou-se um estudo de aplicações de mineração de dados paralela e/ou distribuída e de ambientes que visam apoiar a execução dessas aplicações em ambientes de computação de alto desempenho. No primeiro caso, buscou-se levantar requisitos demandados pelas aplicações estudadas de forma que se pudesse entender o funcionamento do processo de mineração e se pudesse suportar esses requisitos de maneira adequada através das ferramentas propostas na arquitetura. No estudo dos ambientes já existentes, buscou-se identificar as abordagens utilizadas e, com base nisso, verificar quais eram as lacunas que essas ferramentas possuíam, de modo que se pudesse supri-las através do trabalho proposto.

Também se realizou a paralelização de uma aplicação de mineração de dados para a tarefa de regressão. Considerou-se importante essa paralelização pois além de constituir uma contribuição para o Estado da Arte possibilitou uma maior compreensão de um processo de mineração, o que certamente foi útil ao processo de construção da solução apresentada nessa dissertação. Além disso, possibilitou a existência de uma implementação consistente a ponto de ser usada para fins de avaliação do trabalho desenvolvido.

Apresentadas a introdução e motivação do trabalho desenvolvido, segue a descrição dos objetivos que se buscou atingir durante o desenvolvimento do trabalho.

1.1 Objetivo Geral

O objetivo geral do trabalho é apresentar uma solução que apoie todos os processos de execução de uma aplicação de mineração de dados paralela e/ou distribuída em recursos computacionais heterogêneos sem que isso demande conhecimento técnico avançado do usuário da ferramenta.

1.2 Objetivos Específicos

Os objetivos específicos do trabalho foram:

- Propor ferramentas que apoiem a manipulação de grandes bases de dados;
- Construir um algoritmo de dimensionamento e mapeamento de aplicações de mineração em recursos heterogêneos capaz de tomar decisões sem ter conhecimento de métricas de desempenho e do comportamento da aplicação nos recursos e de realizar mapeamentos e dimensionamentos em virtude das peculiaridades da aplicação e dos recursos disponíveis;
- Criar uma ferramenta para prospecção dos recursos computacionais disponibilizados;
- Paralelizar uma aplicação de mineração ainda não paralelizada.
- Disponibilizar um mecanismo para apoio a execução remota de processos em recursos computacionais heterogêneos.
- Criar uma arquitetura que gerencie a execução completa de uma aplicação de mineração agregando todos as ferramentas propostas.

1.3 Organização do Documento

O restante do documento está organizado da seguinte maneira: O Capítulo 2 apresenta uma introdução aos conceitos de computação paralela e distribuída e de mineração de dados. No Capítulo 3 está exposto o estudo das aplicações de mineração de dados paralela e/ou distribuída utilizados como levantamento de requisitos à arquitetura. Neste capítulo também se apresenta a descrição da paralelização realizada para a técnica de regressão. No Capítulo 5 são apresentados a arquitetura e o algoritmo de mapeamento desenvolvidos. O Capítulo 6 descreve os experimentos realizados a fim de avaliar a solução proposta e faz uma discussão sobre esses resultados. Por fim, as considerações finais são apresentadas no Capítulo 7.

2. Fundamentação Teórica

Este capítulo apresenta uma revisão de termos e conceitos que embasaram o trabalho desenvolvido e apresentado neste documento. Esta fundamentação apresenta os conceitos básicos de computação paralela e distribuída e de mineração de dados.

2.1 Computação Paralela e Distribuída

Esta seção apresenta os conceitos básicos sobre computação paralela e distribuída com o objetivo de auxiliar a compreensão do trabalho proposto. Segue a descrição desses conceitos.

2.1.1 Conceitos Básicos de Processamento Paralelo e Distribuído

Segundo De Rose [12] e, Hwang [23], define-se processamento paralelo como sendo várias unidades de processamento ativas utilizadas para a resolução de um mesmo problema. Entende-se por processamento paralelo e distribuído quando várias unidades de processamento dispostas em mais um recurso computacional trabalham para a resolução de um único problema.

Em se tratando de tipos de paralelismos, pode-se classificá-los em três níveis [23]: (a) paralelismo de instruções, (b) paralelismo de dados e (c) paralelismo de tarefas. O paralelismo a nível de instrução diz respeito a execução simultânea de um grupo de instruções para a resolução de um único problema. O paralelismo de dados corresponde a distribuição de dados em diferentes unidades de processamento onde em cada uma se realiza o processamento de parte desses dados. Por fim, o paralelismo de tarefas consiste na execução de diversas instâncias completas de uma aplicação que ocorre simultaneamente sobre diferentes unidades de processamento.

Segundo De Rose [12], existem dois procedimentos básicos que são utilizadas para se medir o desempenho de uma aplicação paralela: o cálculo do fator de aceleração (Speed-Up) e o cálculo de eficiência.

O cálculo do Speed-Up visa indicar quantas vezes um programa paralelo ficou mais rápido do que sua versão sequencial. Quanto maior for o fator de aceleração obtido, maior é o desempenho de uma aplicação. Esse valor é obtido pela razão entre o tempo de execução sequencial do programa e o tempo da versão paralela do mesmo. É expresso pela seguinte fórmula:

$Speed-Up_p(w) = \frac{T(w}{T_p(w)}$

Onde p representa o número de unidades ativas utilizadas e w corresponde a aplicação que está sendo avaliada.

Para se ter uma métrica de eficiência de utilização das unidades de processamento utilizadas por uma aplicação paralela, faz-se o seguinte cálculo que objetiva obter a taxa de utilização média das unidades de processamento ativas utilizadas pela aplicação paralela:

$Eficiência_p(w) = \frac{SU_p(w)}{p}$

O valor obtido corresponde a porcentagem de utilização das unidades de processamento durante a execução da aplicação paralela. Quanto mais próximo o valor for de 100%, maior foi a eficiência de utilização dos recursos. Esse cálculo é apropriado quando se tem unidades de processamento com poder computacional idêntico.

Apresentados os conceitos fundamentais de computação de alto desempenho, segue a descrição dos principais ambientes que visam suportar aplicações paralelas.

2.1.2 Ambientes de Computação Paralela e Distribuída

Apresenta-se nesta seção uma descrição dos ambientes de computação paralela e distribuída que servirão como plataformas de execução no desenvolvimento do estudo. Dá-se ênfase especial a ambientes de baixo custo e alta popularidade. Serão apresentados os ambientes de grades e clusters computacionais, além dos ambientes formados por computadores de múltiplos núcleos de processamento (multinúcleos) que recentemente vêm se destacando pela seu alto poder computacional e baixo custo. Para essa descrição se utiliza da descrição comercial desses ambientes por se considerar mais intuitiva e mais presente em cenários corporativos que são o maior nicho de aplicações de mineração de dados.

Máquinas SMP

Segundo De Rose e Navaux [12], máquinas SMP (*Symmetric Multiprocessors*), em português multiprocessadores simétricos são "sistemas constituídos de processadores comerciais, também denominados ‘de prateleira‘ conectados a uma memória compartilhada, na maioria das vezes através de um barramento de alta velocidade". Nesses sistemas, não existe hierarquia de acesso à memória entre os processadores, ou seja, todos os processadores utilizam do mesmo barramento de acesso a memória e, por padrão, todos eles têm prioridade igual de escalonamento sob o ponto de vista do sistema operacional.

Essas máquinas existem a mais de uma década e inicialmente tinham um preço relativamente alto, compondo somente um nicho específico de mercado e geralmente compradas para um fim específico [11, 23].

A partir do início desta década, surge a tecnologia de processadores de múltiplos núcleos (multinúcleos) que consiste na disposição de mais de um núcleo de processamento em um único processador, compartilhando os mesmos barramentos e memória, por isso considerados máquinas SMP [20]. O único diferencial dessas máquinas em relação às máquinas SMP tradicionais diz respeito ao fato de que as arquiteturas *multi-cores* geralmente compartilham a memória cache entre suas unidades de processamento, enquanto que em arquiteturas SMP tradicionais, geralmente cada unidade de processamento tem uma memória cache individual.

Tal avanço resultou numa expansão do poder computacional de computadores de baixo custo, uma vez que, ao contrário de tempos atrás, hoje máquinas SMP estão presentes até mesmo em

computadores pessoais. Isso proporciona um novo cenário para aplicações paralelas, já que recursos antes escassos estão presentes em todas as escalas de uma organização, despertando novos desafios que visam criar ferramentas de desenvolvimento e execução de aplicações que possam aproveitar as características dessas máquinas.

Os principais desafios, no que tange a arquiteturas multinúcleos, dizem respeito à criação de aplicações que possam se utilizar de todos os núcleos presentes na máquina e que levem em conta que esses núcleos possuem memórias e caches compartilhados e, quando essas máquinas estão presentes num cluster de computadores (Seção 2.1.2), demandam soluções de escalonamento híbridas, que sejam capazes de gerenciar tanto a característica de memória distribuída (entre os nodos do cluster) e de memória compartilhada (entre os núcleos de processamento de cada nodo do cluster).

Clusters de Computadores

Clusters de computadores podem ser definidos como um conjunto de computadores interligados por uma rede de comunicação em um único domínio administrativo [23], [12]. A principal motivação que levou a criação desses ambientes, foi o alto custo de supercomputadores e máquinas SMP e também a baixa escalabilidade de máquinas SMP [12]. Para contornar esses obstáculos, cria-se uma rede de interconexão e um único domínio de administração para gerenciar usuários, armazenamento de dados e rede e conecta-se computadores nessa rede e domínio, de modo que esses possam trabalhar em conjunto para a execução de aplicações especialmente preparadas para esses ambientes.

De acordo com a classificação comercial de máquinas paralelas, pode-se dividir clusters de computadores em três classes: Máquinas Maciçamente Paralelas (MPP - *Massively Parallel Processors*), Redes de Estações de Trabalho (NOW - *Networks of Workstations*) e Máquinas Agregadas (COW - *Cluster of Workstations*) [12, 23].

Segue uma breve descrição de cada um desses ambientes:

Clusters MPP São máquinas constituídas por uma grande escala de processadores comerciais conectados por uma rede de alta velocidade e baixa latência, de dedicação exclusiva para a execução de aplicações paralelas e colocados num ambiente especialmente preparado em termos de fornecimento de energia e refrigeração. São geralmente soluções comerciais vendidas em blocos já preparados em uma arquitetura diferenciada, de modo a melhorar a comunicação e interligação dos nodos do cluster [11].

Essas máquinas são utilizadas para aplicações de alto desempenho e que demandam larga escala, comunicação de baixa latência e alta disponibilidade. São máquinas geralmente caras e com mecanismos especiais para armazenamento de dados, segurança e redundância de componentes.

Clusters NOW Este tipo de *cluster* é composto por máquinas de trabalho já presentes em uma rede comum. São utilizados mecanismos que detectam a ociosidade dessas máquinas e, quando isso

ocorre, designam trabalho para as mesmas. As principais características desses ambientes é o baixo custo para construí-los, o qual muitas vezes é nulo, pois utiliza-se estruturas já existentes e a baixa velocidade e instabilidade da rede de interconexão das máquinas do cluster, já que essas ficam à mercê da qualidade e topologia da rede a qual os nodos do cluster já estão conectados [12].

O uso mais comum dessas máquinas é em soluções de baixo custo, onde se visa principalmente o aproveitamento de recursos ociosos em detrimento do alto desempenho das aplicações propriamente dito, embora que, dependendo do cenário e aplicação, possa-se conseguir um bom desempenho de aplicações nesse tipo de *cluster*.

Cluster COW Os *clusters* COW são uma evolução das máquinas NOW, com a diferença de que se utiliza computadores comuns, ditos "de prateleira"para uso exclusivo de aplicações paralelas. Difere-se também de NOW no fato de que todo o ambiente de software e infraestrutura de hardware é projetado especialmente para propiciar o bom funcionamento do cluster na execução de aplicações paralelas. As vantagens desse ambiente é a possibilidade de construção de um ambiente de relativo alto desempenho e alta escalabilidade comparados com a cara solução de clusters MPP. Como o ambiente é concebido exclusivamente para aplicações paralelas, pode-se aumentar a eficiência do mesmo através da implantação de uma rede de alto desempenho, facilitando aplicações de troca de mensagens a terem um bom desempenho [12].

Comumente essa máquinas são utilizadas por aplicações paralelas que demandam alto desempenho e um grau de confiança maior do que o ambiente NOW pode oferecer, mas não têm recursos financeiros suficientes para implementar uma soluções MPP. São ideais para aplicações de troca de mensagens e com alta escalabilidade.

Grades Computacionais

A descrição de grades computacionais se faz importante, pois de acordo com alguns autores [7, 8,17], a configuração dos recursos utilizada pela arquitetura se constitui numa grade computacional, pois possui recursos que podem estar em diferentes *sites* sob diferentes domínios administrativos. Porém, apresenta a peculiaridade que nessa grade se tem recursos especialmente dedicados para uso como alto desempenho. O que de acordo com Buyya et al. [27] pode ser considerado uma grade privada.

Segue a descrição de Grades Computacionais.

O crescimento da Internet, acompanhado do crescente desenvolvimento tecnológico dos computadores e da disponibilidade de redes de alta velocidade a custos relativamente baixos, possibilitou o uso de recursos computacionais distribuídos como uma única abstração. Essa abstração, que permite que diversos recursos computacionais sejam acessíveis como se fossem um, é chamada de Grade computacional [19] ou ainda Computação em Grade, ambos termos derivados de *Grid Computing*.

O uso do termo *Grid* vem da área de engenharia elétrica, onde ocorre o uso de recursos distribuídos heterogêneos (energia elétrica proveniente de diversos geradores geograficamente distribuídos e de diferentes capacidades) de forma transparente, em termos de localização e acesso [6].

Essa definição, inicialmente, dizia respeito ao uso de diversos recursos computacionais de forma cooperativa para prover poder computacional para problemas de larga escala, instrinsecamente ligados à computação de alto desempenho. Esse conceito foi se refinando, tornando-se cada vez mais genérico, englobando utilizações além do uso para somente processamento de alto desempenho, ficando definida uma grade computacional como uma infraestrutura para compartilhamento de recursos para a solução de problemas de forma colaborativa [17]. Atualmente, a definição aceita para Grades classifica-as como infraestruturas para encapsulamento e virtualização de recursos, de modo que esses possam ser acessados de maneira transparente por seus usuários [18]. Com essa definição mais ampla, abre-se o leque de utilização de grades computacionais, estendendo seu uso para recursos que vão além do compartilhamento de ciclos de processamento, como armazenamento e compartilhamento de dados e, mais recentemente, para o provimento de serviços com a arquitetura OGSA (*Open Grid Services Architecture*) [15].

Segundo Nemeth et. al. [30] e Foster et. al. [19] uma Grade deve possuir algumas características. Entre elas destacam-se: escalabilidade, heterogeneidade, compartilhamento, controle distribuído, repositório virtual de recursos, dinamicidade, transparência, segurança, economia, imagem de sistema, tolerância a falhas, aplicações, escalonamento de recursos e heurísticas de escalonamento.

Dentre as características citadas, destacam-se a heterogeneidade e transparência de uma Grade. A maioria das grades existentes possuem recursos fisicamente distribuídos e de diferentes configurações. Por exemplo, uma mesma grade pode encapsular computadores com processadores e arquiteturas diferenciados uns dos outros. Esses podem estar sobre redes diversas e possuírem outras atribuições na organização, ou seja, estarem compartilhados para outras aplicações ou usuários. Essa característica, que por um lado impulsiona o desenvolvimento de Grades computacionais, pois permite que recursos heterogêneos que são utilizados para outras tarefas possam ser compartilhados numa Grade, agregando recursos a mesma,concomitantemente, gera uma série de fatores e situações que precisam ser tratados. Por exemplo: disponibilidade de nodos, instabilidade de recursos e escalonamento de tarefas a serem executadas numa Grade.

O requisito de transparência de uma Grade também deve ser destacado, pois é fundamental para permitir que usuários acessem os recursos da Grade sem precisar conhecer sua estrutura interna e também, para que desenvolvedores possam construir soluções escaláveis que possam ser executadas em diversas implementações de Grades.

Do ponto de vista dos elementos que compõem uma Grade se pode dividi-los em três camadas: infraestrutura, *middleware* da grade e aplicações da grade [6]. A camada de infraestrutura diz respeito ao conjunto de dispositivos de software e hardware individuais que integram uma grade. A camada do *middleware* da grade tem a função de agregar, gerenciar e disponibilizar os recursos da infraestrutura da grade de forma transparente aos seus usuários ou seja, é ponto central de uma

Grade, responsável por intermediar as camadas de infraestrutura e aplicação. A camada de aplicação da grade é composta por aplicações projetadas de forma que possam aproveitar os recursos oferecidos pelos *middlewares* de grades. Aplicações de Grades são geralmente formadas por um conjunto de tarefas (*tasks*), pertencentes a um trabalho (*job*).

Apresentados os ambientes de computação que fazem parte do escopo do trabalho, segue uma descrição das classes mais comuns e utilizadas para a implementação de aplicações paralelas e/ou distribuídas.

2.1.3 Classes de Aplicações para Ambientes de Computação Paralela e Distribuída

Existem diversos tipos ou classes de aplicações para ambientes paralelos. Segue as classes mais comuns.

Bag of Tasks (BoT):

Aplicações assim denominadas, "saco de tarefas", são aplicações nas quais não existe nenhuma dependência entre as tarefas que a compõe, não havendo relação de precedência entre elas. Aplicações desse tipo podem ter suas tarefas executadas em qualquer ordem e o resultado de uma tarefa não interfere no das demais, assim como o resultado das demais tarefas não interfere no resultado dessa tarefa.

Parameter Sweep:

Aplicações denominadas *Parameter Sweep* (PS) ou de troca de parâmetros são compostas por diversas instâncias de um único programa (algoritmo), onde cada instância deste é executada com diferentes parâmetros. Geralmente se apresentam como uma subclasse de *Bag of Tasks*.

Workflow:

São aplicações de fluxo de execuções, nas quais existe uma certa dependência na ordem de execução das mesmas, sendo esta relação uma precedência, dependência de dados ou de controle entre as tarefas.

Troca de mensagens:

São aplicações que utilizam mecanismos de troca de mensagens para trocar dados durante a sua execução. Embora esse tipo de programação possa ser utilizado para implementar aplicações PS e BoT, geralmente é utilizado para aplicações que apresentam dependência de dados entre as tarefas em tempo de execução. Geralmente são implementadas através das tecnologias de *sockets*, da biblioteca *MPI*, entre outras. Entre os modelos de programação utilizados para implementação dessa classe, pode-se citar os modelos *mestre-escravo*, *Fases Paralelas*, de *Pipeline*, entre outros.

Apresentados os conceitos de processamento paralelo e distribuído, mostra-se uma introdução aos conceitos de descoberta de conhecimento, especificamente à disciplina de mineração.

2.2 Mineração de Dados

Conforme Tan [35] Mineração de Dados (*Data Mining*), de uma maneira generalizada, consiste "na aplicação de algoritmos sobre bases de dados a fim de se extrair conhecimento útil não trivial dessas". Ainda de acordo com Larose em [25], citando o Gartner Group [37], pode-se definir mineração de dados como sendo "um processo de descoberta de novas correlações, padrões e tendências significativas através da examinação de grandes quantidades de dados armazenados em repositórios através do uso de tecnologias de reconhecimento de padrões, de estatística e de matemática".

O surgimento de mineração de dados é devido à necessidade que organizações como empresas, governos e instituições acadêmicas têm de buscar conhecimentos, padrões e características até então desconhecidos, de modo que esse conhecimento possa ser aplicado para melhorar o funcionamento dessas instituições, principalmente no que tange ao auxílio para tomadas de decisões [35]. Um fator importante que possibilitou a utilização de tecnologias de mineração de dados da forma como é conhecida hoje, foi a informatização dos processos das organizações e a utilização de tecnologias dedicadas ao armazenamento de dados, como banco de dados relacionais, por exemplo. Essa informatização faz com que todos os dados provenientes de pesquisas ou de processos que ocorrem dentro de uma organização sejam devidamente armazenados em bases de dados de forma trivial, oferecendo quantidades significativas de registros para análise. Sem esse suporte de tecnologias que facilitam a coleta e armazenamento de dados, o processo de mineração de dados, da forma como se conhece hoje, seria de difícil implementação [25].

Embora tarefas de mineração de dados propriamente ditas se constituem na aplicação de técnicas de descoberta de conhecimento sobre bases de dados, salienta-se que essas bases devem estar previamente preparadas e corretamente populadas. Mineração de dados só obtém sucesso em suas aplicações quando todo o processo de coleta de dados de uma organização estiver consolidado de maneira a garantir a integridade e acuracidade dos dados armazenados em suas bases. Portanto, pode-se considerar as tarefas de mineração de dados como sendo parte de um processo maior que passa por fases que vão desde a correta alimentação de bases de dados e definição de objetivos a serem alcançados no processo, até as fases de preparação, consolidação e, efetivamente, a mineração dos dados. O processo termina com a análise e validação dos resultados da mineração e interpretação desses resultados de forma que se possa extrair conhecimento útil para ser incorporado nos processos de negócios da organização [25, 35].

A realização de um processo de mineração pode se dar por diversos fins. Esses fins são denominados tarefas de mineração. Por sua vez essas tarefas podem ser executadas por meio de diversos métodos que finalmente, podem ser implementados através de diversos algoritmos. Segue uma breve descrição das quatro tarefas de mineração citadas no decorrer do trabalho:

2.2.1 Agrupamento

Consiste na identificação de grupos (categorias) de indivíduos de uma base de dados que contenham algum significado e/ou utilidade. É obtida através da análise de similaridade entre seus atributos [35]. É amplamente aplicada em áreas como prospecção de clientes, biologia, psicologia e medicina. Também é utilizada como preparação de dados para a aplicação de outras tarefas de mineração, principalmente para a tarefa de classificação.

2.2.2 Classificação

Tarefas de classificação consistem na tarefa de se classificar indivíduos de uma base de dados em categorias já conhecidas. Nesse processo, cada indivíduo a ser classificado é submetido a um modelo classificatório que atribui a esse indivíduo uma classificação entre àquelas já pré-definidas [35]. É uma tarefa que com frequência complementa a tarefa de agrupamento [36].

2.2.3 Associação

Consiste em encontrar relações associativas entre atributos de um conjunto de dados, ou seja, atributos que têm uma tendência em terem valores "casados"em cada registro da base [36]. São geralmente expostas como regras lógicas que possuem uma taxa de confiança e suporte para validação, como por exemplo: se X, então Y, com um suporte de 75% e confiança de 80%.

2.2.4 Regressão

São técnicas preditivas utilizadas para se classificar tuplas de uma base de dados em valores não discretos, isto é, contínuos [35]. Pode-se citar como exemplos de utilização de métodos de regressão a predição de valores do mercado de ações em razão de outros indicadores econômicos e projeção de vendas baseada na quantidade gasta em publicidade por uma companhia.

Terminado a fundamentação teórica, seguem os estudos de caso estudados de aplicações de mineração paralela e/ou distribuída.

3. Estudos de Caso em Mineração de Dados Paralela e Distribuída

Este capítulo apresenta estudos de casos de paralelização de algoritmos para tarefas de mineração de dados. Tal levantamento foi realizado com o intuito de melhor conhecer o comportamento dos algoritmos de mineração e de se levantar os requisitos que esses demandam para as suas execuções quando implementados de forma paralela. Para esse estudo se considerou importante a escolha de estudos de caso já existentes no Estado da Arte, a experimentação prática envolvendo a execução de um algoritmo de mineração paralelo e o desenvolvimento de uma solução paralela para um caso ainda não citado no Estado da Arte.

Segue, portanto, a descrição dos estudos de casos realizados através da observações dos trabalhos destacados do Estado da Arte:

3.1 Estudos de Caso Destacados do Estado da Arte

A fim de se reduzir o escopo deste estudo, foram escolhidos os seguintes técnicas para serem estudados: classificação, agrupamento e regressão. Tal escolha é devida ao ambiente de pesquisa o qual este trabalho está vinculado. Nesse ambiente se tem uma necessidade especial em se realizar classificações de bases de dados em valores categóricos ou contínuos, tarefas essas que são bem atendidas por técnicas de classificação e regressão. O estudo das técnicas de agrupamento é realizado pelo fato de essa técnica ser costumeiramente utilizada como preparatória para processos de classificação.

Seguem as descrições dos estudos de cada técnica se apresentando uma breve descrição do funcionamento de cada algoritmo estudado e, as paralelizações encontradas. Após, se apresenta uma análise de cada técnica estudada.

3.1.1 Agrupamento

Para esta tarefa, pesquisou-se implementações paralelas do algoritmo k-Means. Essa escolha é devida às boas referências que se tem quanto a eficácia e eficiência do mesmo e pela ampla aceitação desse.

O algoritmo k-Means busca descobrir k grupos de indivíduos representados pelos seus centróides que geralmente representam a média do grupo. Segundo Tan et al. [35] o algoritmo básico para se realizar esta técnica inicia pela escolha dos k centróides iniciais, onde k é o número de grupos que se deseja dividir a base de dados. Após, até que não haja mais mudanças nos centróides, designa-se cada ponto para o seu centróide mais próximo. Assim, cada coleção de pontos designadas a um centróide é considerada um grupo. Em seguida realiza-se a redefinição dos centróide de cada grupo com base nas designações antes realizadas. Quando não houver mais mudanças nos centróides de cada grupo se entende que os grupos já estão definidos.

Soluções Paralelas Estudadas:

O algoritmo paralelo para k-Means consiste basicamente na definição dos $k-centroids$ iniciais e na subdivisão da base de dados em partições, onde cada partição é designada a uma tarefa e essa aplica o reagrupamento dos indivíduos dessa partição. Após, é realizada uma redução global dos resultados computados em cada partição e se computa os novos $k-centroids$ da base. Então, verifica-se se foi atingido o critério de parada, caso contrário, repete-se o procedimento acima descrito [41].

Outra abordagem, apresentada por Li et al. em [26], consiste na computação distribuída dos novos centróides, onde se evita a retransmissão das partições dos processos escravos para o mestre, já que a computação dos novos centróides é realizada através da troca de mensagens entre os processos escravos.

Encontrou-se diversos trabalhos relacionados a abordagens paralelas para o k-Means. Tanto trabalhos que utilizam ambientes de memória compartilhada por meio de *threads* [13,41], quanto os que utilizam memória distribuída [33,36] através de troca de mensagens. Todas as soluções estudadas apresentaram a mesma estratégia, com diferenciação no modo como é realizada a atualização dos centróides de cada grupo. As soluções tiveram resultados de aceleração linear ou próxima disso. Quanto à eficiência, com exceção de Meira et al. [36] que apresentou eficiência próxima dos 90%, observou-se uma taxa relativamente baixa, variando de 50% a 70%.

Também se realizou um experimento prático onde se testou uma paralelização do algoritmo k-Means realizada por Halil [21]. Nesse experimento, comparou-se a solução sequencial com soluções paralelas implementadas através de OpenMP (*multi-threading*) e com troca de mensagens (MPI). Os resultados apresentados pelo autor indicam um crescimento linear do speed-up do algoritmo para a solução com OpenMP e próximo ao linear com a solução MPI, ambas com uma taxa de eficiência acima de 70%. No entanto a experimentação prática dessas implementações mostrou que para o melhor caso com OpenMP uma aceleração foi de 5.1 vezes com 8 processadores (64% de eficiência) e de 4.4 vezes com 6 processadores com MPI (73% de eficiência), ambos em relação à versão sequencial. Para os testes, utilizou-se bases de 100 a 500MB variando o número de grupos de 4 a 10. Os melhores resultados foram obtidos com bases de 500MB com 6 clusters.

Conclusões

Percebe-se que pela natureza dos algoritmos de agrupamento, que implica em diversas iterações para se atingir o resultado esperado, a paralelização eficiente desses fica restrita a ambientes que permitam a troca de mensagens entre processos, indicando dificuldade na implementação de soluções paralelas com abordagens do tipo BoT.

3.1.2 Classificação

Os processos que envolvem uma tarefa de classificação partem da construção de um classificador que, a grosso modo, inicia-se com a indução de um modelo de aprendizado baseado em um conjunto de treinamento. A Partir desse treinamento, cria-se um modelo sobre o qual se realiza uma dedução sobre a base de testes a fim de verificar a acurácia desse classificador. Finalizado o processo de construção do classificador, finalmente se faz a classificação dos indivíduos os quais se desconhece as suas classes (base de dados real) [35].

Soluções Paralelas Estudadas:

Entre os trabalhos encontrados, procurou-se focar em soluções para os algoritmos C4.5, SVM e combinação de classificadores por *bagging*. Salienta-se que SVM também pode ser utilizado para tarefas de regressão.

Neste escopo, encontrou-se duas abordagens para a paralelização de classificação: (i) por meio da paralelização da construção do classificador e (ii) através da construção de diversos classificadores em paralelo, onde cada uma dessas construções acontece em paralelo (métodos de *ensemble* [35])e, após essas construções, realiza-se uma redução global desses classificadores a fim de se obter o que se considera ser o mais apto, dado um determinado critério.

Em Zaki et al. [40], realiza-se a construção da árvore de decisão em paralelo do algoritmo C4.5, através da decomposição de cada sub-nível da árvore correspondente a um atributo criado durante a construção da árvore de decisão e se processa a construção dessas sub-árvores em paralelo. Em termos de paralelização do fluxo de dados, a abordagem utiliza memória compartilhada (*threads*), portanto o acesso a dados é realizado sobre uma mesma estrutura de dados. Quanto aos resultados, o abordagem atinge speed-up próximo ao linear quando existe uma razoável quantidade de atributos de dados a serem analisados com uma eficiência de quase 90%. Salienta-se que os testes foram realizadas em uma máquina com somente 4 unidades de processamento.

Nos estudos de caso encontrados para SVM (*Support Vector Machine*, destaca-se o trabalho de Meligy [29] que realiza uma paralelização do tipo BoT para o algoritmo citado.

Na versão sequencial do algoritmo SVM, são definidos k conjuntos de testes a serem aplicados sobre uma base de treinamento. Após essa aplicação, faz-se operações de *merge*(junção) entre as k aplicações antes realizadas sobre a base de treinamento. Finalmente, realiza-se a soma de cada um dos *merges* realizados e, sobre essa soma, descobre-se o maior valor encontrado, que será o parâmetro a ser utilizado para o conjunto de validação.

A paralelização do algoritmo acima citado escalona as k aplicações dos conjuntos de testes sobre a base de treinamento em k processos que são independentes entre si. Esses processos são distribuídos em diversas unidades de processamento. Nos experimentos realizados pelo autor, submeteu-se essas tarefas a um ambiente de grade Globus [16]. Após o término desses processos paralelos, realiza-se sequencialmente os passos seguintes, análogos ao algoritmo sequencial. Esta abordagem se mostrou

bastante escalável, com um potencial de aceleração linear e com eficiência de 90%. Além disso, possibilita a execução da solução paralela em diversos tipos de recursos computacionais, uma vez que a paralelização cria tarefas do tipo BoT.

Por fim, apresenta-se uma solução de paralelização de um processo de *bagging* do algoritmo C4.5 [5]. A técnica de *Bagging* consiste na criação de diversos conjuntos de treinamentos derivados de um conjunto de treinamento inicial que servem para a criação de diversos classificadores, um para cada conjunto de treinamento criado. Sobre esses classificadores criados, aplica-se uma função a fim de definir qual é o mais adequado. Essa técnica é utilizada a fim de gerar melhores classificadores.

No trabalho estudado, é apresentado um *framework* para criação de milhares de classificadores sobre pequenas fatias do conjunto de dados de treinamento. A metodologia proposta pelo autor objetiva permitir o uso de conjuntos de treinamento com tamanhos significativamente grandes ao ponto que tornaria a criação de um classificador em um único recurso computacional inviável computacionalmente. A abordagem para atingir tal objetivo consiste na subdivisão aleatória do conjunto de treinamento em centenas ou milhares de pequenas fatias de dados, criando-se uma enorme gama de classificadores derivados dessas fatias.

Os resultados apresentados pelos autores mostram que este método, quando utilizado com votadores para escolha do melhor classificador adequado, é eficaz em comparação ao uso de outras técnicas tradicionais, especialmente quando se trata da utilização de conjunto de dados de treinamento de tamanho significativo. O trabalho visou verificar a acurácia da solução proposta, não apresentado testes de desempenho em relação à paralelização da mesma. No entanto, conforme se viu nos casos previamente citados, a paralelização de técnicas como essas é natural e fácil de ser implementada, gerando possibilidades de alta escalabilidade e aceleração.

Conclusões

Pôde-se perceber que a tarefa de classificação apresenta diversas alternativas para a geração de soluções paralelas tanto para ambientes de memória compartilhada, quanto para memória distribuída. As técnicas pesquisadas oferecem oportunidades de paralelismo híbrido, uma vez que se pode combinar a utilização de *bagging* com a paralelização da geração de cada um dos classificadores presentes nesse tipo de abordagem.

Apresentados os estudos de caso do estado da arte, a próxima seção descreve a paralelização realizada para algoritmo E-Motion [1].

3.2 Estudo de Caso Prático: Construção de uma Solução Paralela para Regressão

Entre as formas existentes para se efetuar métodos de regressão, destaca-se a metodologia de indução de árvores modelo [35]. Esta abordagem é realizada frequentemente com algoritmos ditos gulosos através da técnica de divisão e conquista.

Em Barros [1] se apresenta uma nova abordagem para indução de árvores modelo para regressão através do paradigma de algoritmos evolutivos - o algoritmo E-Motion (*Evolutionary Model Trees Induction*). É sobre a implementação desse algoritmo que se fez o estudo de caso de paralelização aqui apresentado. O algoritmo proposto tem seu fluxo de execução que se inicia a partir do particionamento da base de dados a ser utilizada em n partições (*cross-fold-validation)*, de acordo com um algoritmo para esse fim. Sobre cada partição, são executados os passos que dizem respeito à aplicação das técnicas do paradigma evolutivo. Finalmente, faz-se uma computação sobre os n resultados derivados da execução a fim de se encontrar a arvore modelo a ser utilizada para regressão.

Os passos realizados em cada uma das partições correspondem aos passos de um modelo clássico para algoritmos evolutivos que segue os seguintes passos:

1. Cria população inicial.
2. Realiza avaliação.
3. Até atingir critério de parada, evolui população.

Os passos para evolução da população são compostos, sequencialmente, pelas atividades de seleção, *crossover* e mutação. Esses processos correspondem, analogicamente, à seleção natural biológica. Na seleção é realizada a escolha aleatória de indivíduos da população e sobre esses é aplicada uma função que busca avaliar qual é o indivíduo apto para ser selecionado para a fase de *crossover*. Durante o *crossover*, faz-se a simulação da reprodução entre os indivíduos da população selecionada. Esta reprodução é realizada pela escolha aleatória de dois indivíduos que se "reproduzem"gerando dois novos indivíduos. Na mutação, induz-se modificações sobre os indivíduos da nova população, finalizando o processo evolutivo. Após cada evolução, realiza-se um processo denominado *filtragem* o qual verifica em cada individuo se há inconsistências em sua árvore devido à mutação e, caso necessário, realiza modificações nesse a fim de corrigir essas inconsistências.

Segue a abordagem que se utilizou para paralelizar a execução do algoritmo.

3.2.1 Abordagem de Paralelização Utilizada:

Na paralelização do algoritmo de Barros, buscou-se atingir dois grãos de paralelismo:

1. Paralelismo externo: possibilitando a execução da evolução simultânea e independente das n partições de dados
2. Paralelismo interno: buscando realizar os passos de cada evolução em mais de um fluxo de execução (em paralelo).

Para se atingir o primeiro objetivo (1), realizou-se uma adaptação na implementação da solução que permite que diversas referências de partições de dados pudessem ser utilizadas por diversas aplicações (tarefas) em um mesmo nodo. Essa modificação permitiu que pudessem ser executados

simultaneamente n instancias do algoritmo sobre n partições de dados, num mapeamento um para um. Também se realizou uma adaptação que permite que essas instâncias possam ser executadas de forma distribuída, distribuindo-se os dados e carga a diversos nodos computacionais.

Para a paralelização dos passos do processo de evolução (2), analisou-se quais desses eram computacionalmente onerosos e apresentavam trechos de execução com independência entre si. Diante dessa análise, percebeu-se que os processos de *crossover* e de filtragem eram candidatos a paralelização, pois além de apresentarem trechos de execução que podem ser executados independentemente um do outro, são as fases da execução que correspondem a mais de 70% de todo o processo evolutivo.

A abordagem utilizada para paralelização da filtragem consiste em dividir a população de indivíduos p em partições de acordo com número de processos n que se deseja particionar a execução da filtragem. Assim sendo, cria-se n partições cada uma com tamanho $\frac{p}{n}$, sendo que a última partição comporta os indivíduos excedentes do processo de divisão (resto da divisão). Então, cada partição é designada a um processo independente (*thread*) que realiza a filtragem de cada indivíduo em paralelo. Essa abordagem se aplica, pois a filtragem de cada individuo é realizada sem ordem de precedência e não implica em dependência entre os indivíduos.

No caso da paralelização do *crossover*, calcula-se k reproduções que devem acontecer, baseando-se no total da população. Essas k reproduções são divididas pelo número n de processos que compõem a execução em paralelo. Dessa forma, cada tarefa realizará $\frac{k}{n}$ reproduções, gerando cada uma um vetor de tamanho $t\left\{2 \times \frac{k}{n}\right\}$ de novos indivíduos. A medida que as tarefas forem sendo concluídas, os vetores de novos indivíduos criados em cada processo são agrupados num mesmo vetor para que o algoritmo possa continuar sua execução.

3.2.2 Testes Realizados e conclusões:

Para avaliar o desempenho da solução, testou-se execuções com 10 gerações de 1000 e 3000 indivíduos, com paralelização através de *threads*. Para os resultados aqui apresentados, utilizou-se uma máquina com dois processadores Intel Xeon 2.8Ghz com quatro núcleos de processamento cada um e com 16GB de memória RAM e uma base de dados com 10000 registros. A execução da aplicação consumiu em média 5GB. Outros testes foram realizados e serão apresentados no Capítulo 6.

Os resultados mostram que para gerações com 1000 indivíduos, a partir de 6 processos, a sobrecarga de criação de processos acaba diminuindo o desempenho da execução. Esse comportamento diminui à medida que se aumenta o número de indivíduos da população fazendo com que se aumente a carga de trabalho de cada processo, diminuindo a relação dessa com a sobrecarga de geração de processos. A Figura 3.1 mostra o gráfico de Speed-up dos testes realizados.

Como se pode observar no gráfico, a taxa de eficiência apresentada fica abaixo dos 50% a partir de sete processadores, principalmente para gerações com poucos indivíduos, tal comportamento

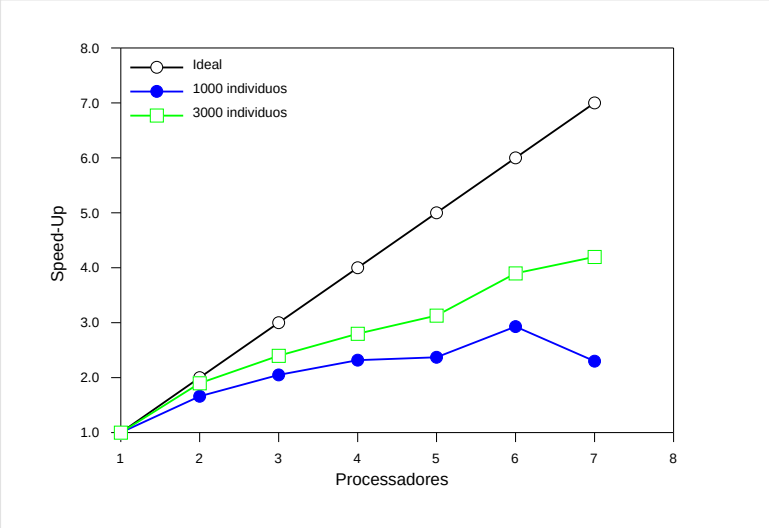


Figura 3.1 – Cálculo de Speed-up para os testes realizados.

também pôde ser observado em Qiu et al. [33]. Devido a isso, os testes com a solução que aproveita o paralelismo externo combinado com paralelismo interno, mostraram que, no ambiente utilizado para teste, pode-se executar três processos externos com, respectivamente, três, três e dois processos internos cada um, sem que ocorra uma sobrecarga de memória do sistema. Dessa maneira, combinando-se a execução com paralelismo interno e externo, pode-se atingir um Speed-Up maior e com eficiência de utilização de recursos maiores do que na execução de uma única instância da aplicação. Quanto ao desempenho da solução, atingiu-se desempenho compatível com a paralelização de algoritmos genéticos fora do campo de mineração [10].

A estratégia utilizada também pode ser aplicada sobre um algoritmo evolutivo para indução de árvores de decisão para classificação.

Tendo-se o embasamento obtido com os estudos de caso, buscou-se realizar um estudo dos trabalhos relacionados com o objetivo geral proposto neste trabalho. Segue a discussão sobre os trabalhos relacionados.

4. Trabalhos Relacionados

Este capítulo apresenta uma visão dos trabalhos relacionados com o aqui proposto. Discute-se alguns trabalhos que objetivam criar um ambiente para a execução de tarefas de mineração de dados em arquiteturas de alto desempenho. Buscou-se avaliar a forma como essas soluções funcionam, suas contribuições e desempenhos.

Nos trabalhos estudados se percebeu que esses se dividem basicamente entre dois grupos: trabalhos que apresentam *frameworks* para a construção de algoritmos de mineração de dados paralelos e trabalhos que apresentam ferramentas ou arquiteturas de serviços para a execução dessas técnicas em ambientes de computação de alto desempenho.

No primeiro grupo, cita-se os trabalhos de Meira et al. com o Anthill [36] e Jin et al. [24]. Ambos fornecem ferramentas que permitem a construção de algoritmos paralelos para mineração de dados com transparência de recursos.

Segue a descrição dos trabalhos que se destacaram nesse primeiro grupo.

4.0.3 Anthill

Foi desenvolvido pela Universidade Federal de Minas Gerais, Brasil. Seu desenvolvimento ocorreu para que fosse utilizado como plataforma de execução no Projeto Tamanduá [38]. Seguem as funcionalidade desse.

O Anthill cria uma modificação do sistema fluxo-instrução que permite a execução de fluxos de *pipelines* de dados em paralelo. Mostrou ser apto a auxiliar na construção de soluções paralelas para tarefas de associação, agrupamento e classificação. A solução consegue, de forma eficiente, aproveitar tanto o paralelismo de dados quanto de fluxo de instruções. O paralelismo de dados é explorado através do particionamento da base de dados a ser utilizada. As implementações realizadas através da ferramenta devem seguir a classe de troca de mensagens e serem implementadas através do modelo de *pipeline*.

O alcance dos dois níveis de paralelismo é realizado da seguinte maneira: são definidas n fases de execuções que compõem o *pipeline* de execução, como se vê na classe de aplicações de *workflow*. As partições de dados que compõe a execução são passadas de fase em fase e, em cada uma dessas fases, realiza-se operações em paralelo (paralelismo de fluxo de execução). O paralelismo de dados é atingindo no momento em que mais de uma fase está sendo executada em paralelo, onde diversas operações são feitas em diversas partições de dados.

Considerações

A solução apresentada pelo Anthill se mostrou uma das mas eficazes e eficientes entre as estudadas. Porém, algumas considerações devem ser levadas em conta.

O método obriga que as aplicações sejam implementadas no modelo de *pipeline* e utilizem o paradigma de troca de mensagens, limitando o campo de atuação da ferramenta. Além disso, somente permite o uso de ambientes de memória compartilhada, somente conseguindo atingir uma boa aceleração quando executado em clusters.

Não foram mencionados detalhes referentes a como a ferramenta faz o escalonamento de suas aplicações, deixando a entender que essa utiliza os mecanismos de escalonamento da ferramenta utilizada para implementação do mesmo, o PVM.

4.0.4 *A Middleware for Developing Parallel Data Mining Applications*

O trabalho de Jin et al. [24] fornece uma abstração para implementações de algoritmos de mineração de dados paralela que permite a identificação, em tempo de execução, através de um *middleware*, das reduções locais presentes nos algoritmos paralelos. Esse procedimento é feito por um processo produtor que analisa essas possibilidades de paralelização e designa trabalho aos processos consumidores que realizam o processamento distribuído e/ou paralelo dos dados.

Considerações

A principal contribuição do trabalho está em possibilitar que aplicações que são implementadas sequencialmente possam ter seus fluxos de dados paralelizáveis automaticamente identificados e submetidos à execução paralela. O modelo de aplicação utilizado para tal é também o de troca de mensagens.

A solução apresentada mostrou que em casos ótimos, onde se tem grande utilização de laços de execução cujas as iterações são independentes, pode-se atingir um resultado bastante eficiente. Porém, a metodologia utilizada dificulta a utilização de aplicações que não apresentam esses requisitos e que também não foram implementadas com as diretivas suportadas pela aplicação. Também se salienta a necessidade de um ambiente que permita troca de mensagens para a utilização do *framework*. O escalonamento das aplicações também é realizado pelos métodos padrões do ambiente MPI.

Seguem as descrições do segundo grupo de trabalhos:

No segundo grupo, destacam-se os trabalhos derivados do Projeto Tamanduá [38] e o Weka4WS [34].

4.0.5 Projeto Tamanduá

Para o Projeto Tamanduá, criou-se uma arquitetura que permite a construção interativa de execuções de tarefas de mineração de dados implementadas com a ferramenta Anthill. Essa arquitetura é composta por um conjunto de módulos que são responsáveis desde a preparação de bases de dados até a visualização de resultados de execuções de mineração e controle de acesso a seus usuários. A atual implementação da arquitetura tem como alvo principal a execução de tarefas de associação.

Considerações

A arquitetura criada para o Tamanduá possui uma gama de funcionalidades e uma organização modular bastante lógica e pertinente. Porém apresenta algumas carências no que diz respeito ao não suporte de ambientes heterogêneos e a necessidade de utilização do Anthill como solução de paralelização. Como a ferramenta é dedicada exclusivamente a clusters, assume-se que haja um ambiente de compartilhamento de área de armazenamento sempre presente, não sendo fornecidas funcionalidades que possam otimizar o acesso a dados. Também não há nenhum mecanismo que ajude minimizar o tempo gasto com os processos de manipulação de dados. Em termos de métodos de escalonamento, a ferramenta usa os disponibilizados pelo ambiente PVM.

4.0.6 Weka4WS

O Weka4WS é uma extensão do software Weka [39] que permite que se realize execuções de algoritmos presentes no Weka em recursos distribuídos. O acesso a esses recursos é realizado por meio de Web Services implementados e implantados numa infraestrutura de grade Globus. Além disso, o Weka4WS permite a distribuição de tarefas independentes presentes num fluxo de execução que compõe a execução de uma tarefa de mineração, permitindo a execução distribuída e, quando o fluxo do processo permitir, paralela de algumas etapas desse fluxo, como por exemplo, preparação de dados e filtragens dos mesmos.

Considerações

A contribuição do Weka4WS é possibilitar a execução remota de minerações em uma das ferramentas mais utilizadas para tal fim, o Weka. O maior problema da ferramenta desenvolvida está na considerável dificuldade de implantação da mesma que exige que cada nodo computacional tenha um *middleware* de Globus instalado e tenha instalado o Web Service para receber as tarefas do Weka. Além disso, o mecanismo de transferência de dados não dá alternativas de transferência que não sejam via Web, o que, para bases grandes, pode acarretar um *overhead* considerável, fazendo com que o ganho que se pode obter com a execução de diversas execuções em paralelo possa ser anulado por esse *overhead*. Outro ponto a ser lembrado é o fato de o mecanismo de distribuição de execuções não distinguir os recursos heterogêneos, fazendo que não se priorize a utilização de um recurso em virtude de outro de menor desempenho.

Tendo-se concluído a discussão dos trabalhos relacionados, no próximo capítulo se apresenta a solução proposta neste documento.

5. Solução Proposta

Embora tenha sido possível encontrar boas soluções de ambientes para apoio à execução de tarefas de mineração de dados em arquiteturas de computação de alto desempenho, observou-se algumas características desses ambientes que os tornam específicos para determinadas implementações ou que dificultam a integração de algoritmos já existentes que não foram implementados com as ferramentas desses ambientes. Por exemplo, Meira et al. [36] (Anthill) e Jin [24] oferecem ambientes eficientes para o desenvolvimento e execução de tarefas de mineração paralelas, porém suas soluções demandam a utilização de uma metodologia de desenvolvimento que limita a generalidade das soluções, demandando que seus utilizadores se restrinjam a uma única linguagem de programação e a um único tipo de ambiente de execução para implementarem e executarem seus algoritmos. Além disso, essas soluções demandam do usuário a instalação de uma estrutura que implica na modificação dos nodos computacionais do ambiente de execução onde se pretende executar os algoritmos, o que acarreta na dificuldade de instalação dos ambientes e na dificuldade de utilização de ambientes computacionais os quais o usuário não tem permissão para tais modificações. Tal característica também é apresentada em Talia et al. [34] (Weka4WS) que demanda a instalação do middleware de grade Globus. No caso do Weka4WS também se salienta que este não implementa soluções para execução de algoritmos em paralelo, mas somente de forma distribuída e suas opções de escalonamento são bastantes limitadas. Outro ponto importante, é a dificuldade que os ambientes existentes têm em utilizar recursos heterogêneos e multinúcleos, dificultando a utilização das ferramentas em ambientes com essas características.

Além dos aspectos acima citados, procurou-se observar a forma como esses trabalhos abordam a gerência dos dados necessários à execução das tarefas de mineração. Nesse estudo, constatou-se que o Anthill e o *framework* proposto por Jin apresentam soluções para manipulação de dados para a execução de aplicações em seus ambientes. No entanto, suas soluções se restringem a ambientes de clusters e como já dito anteriormente, demandam a instalação de softwares nos recursos onde se executarão as aplicações de mineração e restringem a utilização de outros ambientes de execução.

Diante desse cenário, percebe-se a lacuna de um ambiente que permita a seus usuários a utilização de diferentes implementações de soluções de mineração, escritas em diversas linguagens e que possam ser executadas em mais de um ambiente computacional. Também se observa, diante da relevância em termos de tempo de execução que a transferência de dados implica nas tarefas de mineração de dados, a necessidade de mecanismos que apoiem a gerência de bases de dados e o acesso e transferência desses dados que ocorrem durante uma execução de uma tarefa de mineração paralela em recursos computacionais distribuídos.

Assim, apresenta-se uma arquitetura que visa preencher as lacunas acima expostas, de modo que a execução de tarefas de mineração possa ser realizada de forma transparente e fácil para usuários não habituados à utilização de ambientes de computação de alto desempenho, permitindo que esses

usuários possam utilizar a maior quantidade de recursos computacionais e aplicações de mineração que tenham à disposição, independente da heterogeneidade desses recursos.

Essa arquitetura é derivada do estudo apresentado nos Capítulos 4 e 3 onde se buscou descobrir quais as necessidades e características dos algoritmos de mineração de dados quando implementados para ser executados de forma paralela e onde foram analisadas as soluções de ambientes para execução de aplicações de mineração de dados paralela já existentes.

Antes de se entrar na descrição dos componentes da arquitetura apresentada, apresenta-se uma taxonomia das aplicações que são suportadas para serem executadas através da arquitetura e da representação dos recursos computacionais utilizados para a execução dessas aplicações.

5.1 Taxonomias Utilizadas na Arquitetura

Estas taxonomias visam esclarecer quais são as aplicações paralelas para mineração suportadas e como os recursos computacionais são vistos e representados pelos componentes da plataforma.

5.1.1 Taxonomia de Aplicações Paralelas para Mineração de Dados

De acordo com o estudo apresentado na Seção 3, pode-se classificar as aplicações de mineração de dados paralelas e/ou distribuídas em quatro tipos. Essa classificação é obtida através de como essas aplicações se comportam no que diz respeito ao paralelismo externo e interno e como essas aplicações podem variar a escalabilidade de cada um desses níveis de paralelismo. Considera-se o paralelismo externo aquele no qual a aplicação pode dividir sua execução em tarefas independentes, do tipo BoT, não demandando memória compartilhada. Já o paralelismo interno, corresponde à possibilidade que a aplicação tem de dividir sua execução em tarefas que demandam ser executadas em um ambiente de memória compartilhada. Enquadram-se nesse caso, aplicações implementadas com troca de mensagens (p.ex.: MPI) e aplicações que utilizam *threads* como ferramenta de paralelização. Acrescenta-se ainda o paralelismo híbrido, onde se pode combinar tarefas de paralelismo externo que possuam subtarefas com paralelismo interno.

Segue a descrição dessa taxonomia.

Particionamento Externo com Escalabilidade Estática

Esta classe diz respeito àquelas aplicações onde a escalabilidade externa não se altera, ou seja, aplicações nas quais o número de tarefas do tipo BoT é sempre o mesmo. Enquadram-se nessa classe todas as aplicações que realizam processos de validação como o *cross-fold-validation* e que utilizam técnicas de *bagging* ou *random forests*. Também se incluí aplicações do tipo PS. Essa classe é adequada para execução em ambientes de cluster NOW e COW e grades.

Esse tipo de aplicação ainda pode variar o comportamento de seu escalonamento interno, conforme segue:

a) **Particionamento Interno Com Escalabilidade Estática:** São aplicações que demandam um número fixo de particionanento interno. São exemplos dessa classe aplicações que não possuem paralelização interna e utilizam somente as técnicas acima citadas, tendo sua escalabilidade interna fixada em somente uma tarefa (execução sequencial).

b) **Particionamento Interno Com Escalabilidade Dinâmica:** São aplicações híbridas que podem variar o número de tarefas internas, de acordo com a disponibilidade de recursos. Enquadram-se nessa classe aplicações que mesclam as técnicas de replicação com a paralelização do processo de construção de classificadores ou de realização de agrupamento, ou então aplicações de troca de mensagens do tipo MPI, por exemplo, que possuem somente uma instância da aplicação e podem variar o número de processos que esta instância pode ser executada. Esta classe é especialmente adequada para execução em ambientes com nós computacionais multiprocessados.

Particionamento Externo com Escalabilidade Dinâmica

Corresponde às aplicações cujo o grau de paralelismo externo pode variar. Os casos de aplicações desse tipo são mais raros. Como exemplo, pode-se citar aplicações que realizam a classificação de indivíduos de uma base de dados a qual o classificador já está construído. Dessa forma, pode-se dividir dinâmicamente essa base em diversas instâncias de execução e em cada uma dessas instâncias é realizada a classificação dos indivíduos da partição de dados designada à mesma.

Esse tipo de aplicação ainda se divide de acordo com a escalabilidade do paralelismo interno, conforme segue:

a) **Particionamento Interno com Escalabilidade Estática:** Neste caso, utiliza-se aplicações que não possuem implementações paralelas do processo de classificação dos indivíduos de um conjunto de dados sobre um classificador já pronto, fazendo esse processo sequencialmente.

b) **Particionamento Interno com Escalabilidade Dinâmica:** São os casos onde a passagem de cada indivíduos de um conjunto de dados é realizada de forma paralela, podendo-se classificar vários indivíduos simultaneamente, de acordo com número de processadores disponíveis no recurso onde a aplicação está sendo executada.

5.1.2 Taxonomia de Recursos

Esta seção define a maneira como os recursos computacionais são classificados e vistos pela arquitetura. Essa definição é feita de modo a facilitar o entendimento do usuário ao cadastrar seus recursos para utilização na arquitetura e para facilitar a automatização do mapeamento de tarefas aos recursos. Segue os tipos de recursos presentes na arquitetura:

a) **Unidade de Processamento:** Representa uma unidade de processamento existente em um nodo computacional. Pode ser um processador mononúcleo ou núcleo de um processador multinúcleo.

b) **Nodo Computacional:** Representa a máquina de processamento em si. Caracteriza-se por ser uma máquina com uma ou mais unidades de processamento e com memória própria. Exemplos que se enquadram nesta classe são máquinas comuns e SMP.

c) **Cluster de Computadores:** São a representação de um ou mais nodos computacionais os quais estejam sobre o mesmo domínio administrativo. Exige-se que esses possam se comunicar diretamente através de uma rede comum e possuam um nó especial que atua como um *proxy* para acesso aos recursos do aglomerado. Representam os clusters NOW e COW.

d) **Grade:** No contexto da arquitetura, considera-se uma grade o conjunto de todos os recursos computacionais disponíveis, sejam eles nodos computacionais, clusters ou até mesmo outras grades computacionais.

5.1.3 Taxonomia de Tarefas

Esta classificação também é realizada para auxiliar no entendimento de como a arquitetura trabalha com os diversos níveis de tarefas e seus graus de paralelismo. Diz respeito ao relacionamento desses níveis com os recursos computacionais. Este relacionamento é útil na fase de execução da arquitetura onde ocorre o mapeamento e escalonamento de uma aplicação paralela. Segue os tipos de tarefas tratados[1].

a) **Processo:** Um processo corresponde à execução efetiva de um fluxo de execução em uma unidade de processamento. É mapeada a uma unidade de processamento de um nodo computacional. Corresponde ao conceito de escalabilidade interna utilizado na arquitetura.

b) **Task (Tarefa):** Uma task corresponde ao conjunto de processos. Uma task pode ser mapeada a um nodo computacional ou então a um *proxy* de um cluster, quando se trata de aplicações de troca de mensagens. Corresponde ao conceito de escalabilidade externa utilizado na arquitetura.

c) **Job (Trabalho):** Corresponde a um conjunto de tasks da aplicação. Um job é mapeado ao conjunto total de recursos disponíveis, ou seja, à grade.

d) **Aplicação:** Diz respeito ao conjunto de jobs que compreendem a execução de uma tarefa de mineração através da arquitetura.

A Figura 5.1 mostra a relação entre as taxonomias apresentadas, também se ilustra a cardinalidade dessa relação.

Terminada a apresentação dos conceitos e definições a serem utilizadas pela solução, segue a apresentação da arquitetura em si.

[1] Os nomes tasks (tarefas) e job (trabalho) foram mantidos em inglês a fim de não confundir *tarefa* do contexto de divisão de trabalho em paralelo com o contexto de *tarefa* de mineração de dados.

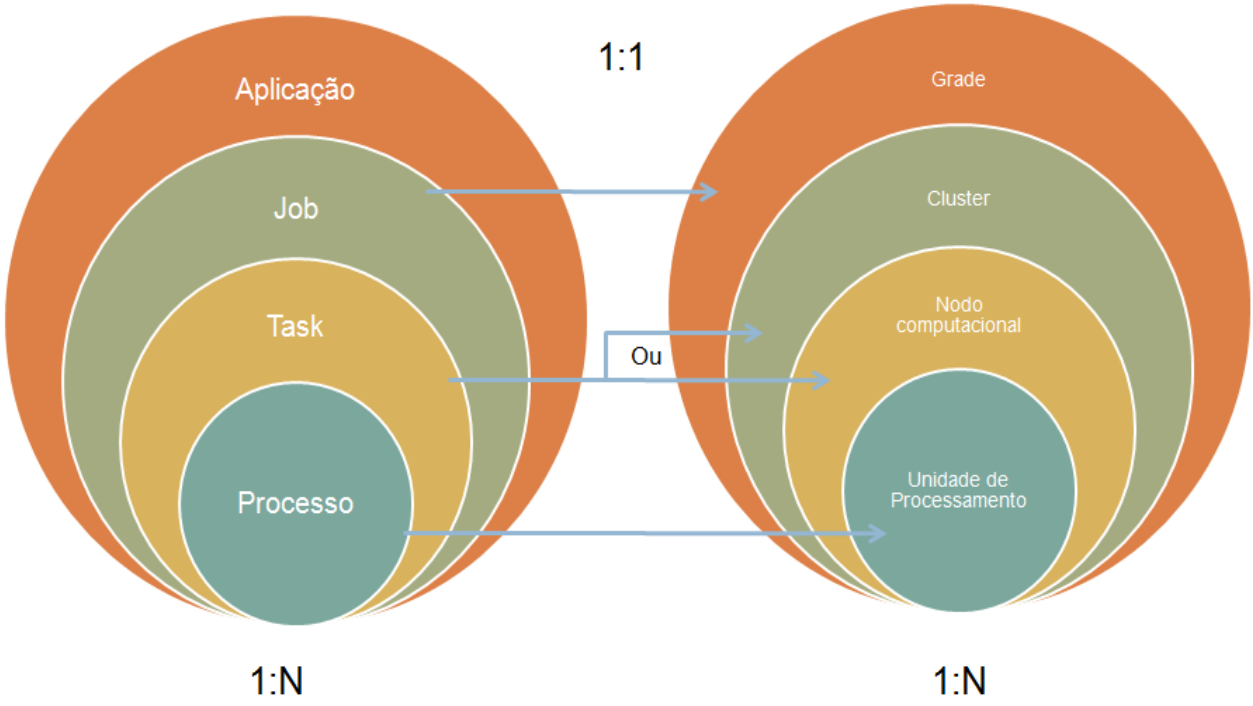


Figura 5.1 – Relações entre as taxonomias de tarefas e recursos.

5.2 Descrição da Arquitetura

Propõe-se uma arquitetura que possibilite que algoritmos de mineração escritos em diversas linguagens possam ser executados em diversos recursos computacionais, realizando, de forma dinâmica, o mapeamento entre as tarefas do algoritmo aos recursos computacionais disponibilizados pelo usuário. Além disso, a arquitetura fornece ferramentas de gerenciamento das bases de dados utilizadas nas tarefas de mineração. Essas ferramentas também integram essas bases à execução das tasks propriamente ditas, fornecendo a essas diversos meios de acesso a dados de acordo com a característica do recurso computacional onde a task está sendo executada.

Almeja-se que, com base na escolha do usuário de qual base de dados e aplicação esse deseja utilizar, a arquitetura possa, de maneira automática e dentro das possibilidades, mapear a execução dessa tarefa de mineração aos recursos computacionais disponibilizados, propiciando uma maior rapidez na execução da tarefa. Outra possibilidade oferecida é que o usuário possa indicar qual a base de dados e recursos computacionais que deseja utilizar e qual o algoritmo compatível com os recursos indicados e, com base nessas escolhas, a arquitetura possa realizar essa execução da forma mais otimizada possível.

Organiza-se essa arquitetura de modo que ela possua um gerenciador de execuções que controle a execução das tarefas de mineração de dados de acordo com os parâmetros definidos pelo usuário. Através dessas definições, por meio da interação com os outros componentes da arquitetura, esse gerenciador se encarrega de realizar a execução da tarefa do usuário nos recursos computacionais disponíveis, de forma transparente. A organização da arquitetura foi baseada na maneira como

as aplicações estudadas são modeladas e na forma como seus fluxos de execução se desenvolvem. Tem-se como base principalmente o algoritmo ao qual se fez um estudo de paralelização (E-Motion).

Assim, buscou-se os pontos em comum desses estudos de casos, moldando-se a arquitetura de forma que a mesma atenda a essas generalidades. De acordo com essa análise, define-se que a arquitetura aqui proposta deve dar suporte para as seguintes operações, que correspondem ao fluxo de execução comum entre as aplicações estudadas:

1) Aquisição de dados de bases de dados;
2) Adequação dos dados ao formato da aplicação;
3) Dimensionamento da escalabilidade da aplicação;
4) Criação de tarefas e processos da aplicação paralela;
5) Mapeamento das aplicações aos nodos computacionais;
6) Particionamento dos dados a serem minerados;
7) Distribuição dos dados para as tarefas nos nodos;
8) Execução das tarefas e processos;
9) Recuperação dos resultados gerados nos nodos;
10) Consolidação dos resultados;
11) Armazenamento dos resultados;
12) Limpeza de arquivos gerados na execução.

A Figura 5.2 mostra, de forma geral, os componentes da arquitetura que são responsáveis pela execução dos passos acima apontados.

Como pode-se observar na Figura 5.2, além do gerenciador de execuções a arquitetura possui uma série de componentes auxiliares, cada um responsável por uma etapa da execução das tarefas de mineração de dados coordenadas pelo gerenciador de execuções. Etapas essas que vão desde a aquisição dos dados até a consolidação dos resultados gerados pela execução dessas tarefas.

A seguir, descreve-se as características e funcionalidades de cada um desses componentes presentes na arquitetura para que seja possível o entendimento do funcionamento e configuração do ambiente aqui proposto. Salienta-se que essa descrição não corresponde a descrição de uma implementação da arquitetura, mas sim a uma organização em um nível mais alto de abstração que pode servir de base a uma possível implementação. Detalhes do protótipo implementado para avaliação da arquitetura proposta são discutidos no Capítulo 6.

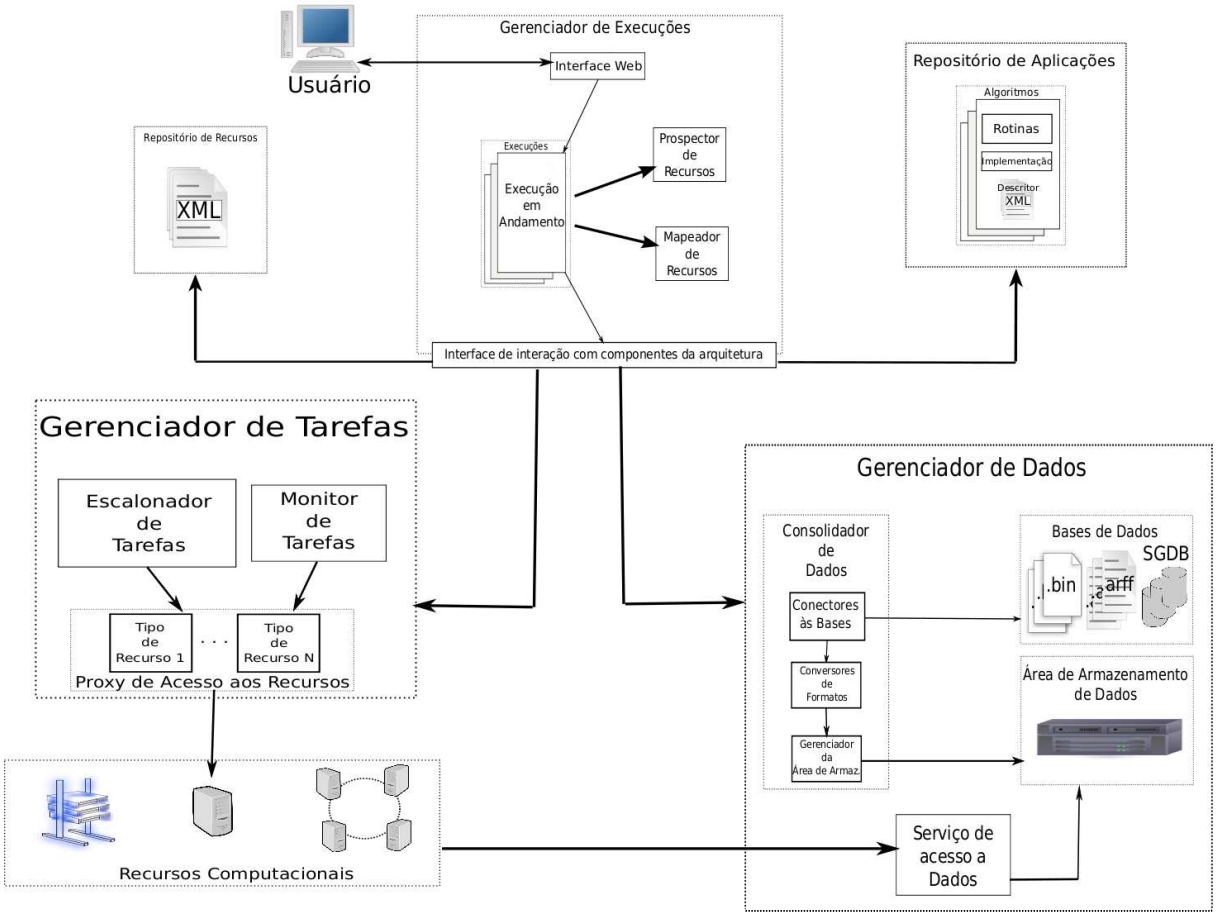


Figura 5.2 – Arquitetura para Apoio à Execução de Mineração de Dados Paralela.

5.2.1 Gerenciador de Execuções

É o componente central e principal da arquitetura. É através do Gerenciador de Execuções que o usuário define os seus cenários de execução e tem a execução desses efetivada. Considera-se como cenário de execução um conjunto de parâmetros que compõem a execução de uma tarefa de mineração paralela através da arquitetura. Esses parâmetros envolvem a base de dados, aplicação, parâmetros de execução dessa aplicação e os recursos utilizados para a execução dessa.

Dados os parâmetros de um cenário de execução, o gerenciador de execuções se encarrega de utilizar dos meios oferecidos pelos demais componentes para preparar e gerenciar a execução desse cenário.

O gerenciamento de uma execução se inicia a partir da escolha, por parte do usuário, de qual base de dados e qual aplicação e parâmetros esse deseja utilizar. A partir dessas escolhas, são disponibilizadas ao usuário duas formas de execução: uma que permite ao usuário escolher manualmente quais os recursos entre os possíveis para o algoritmo escolhido e outra, na qual se define automaticamente quais são os recursos mais adequados para o algoritmo escolhido. Após essa escolha, aciona-se o sub-componente mapeador de recursos [2] que realiza o dimensionamento e mapeamento

[2] O algoritmo que realiza o dimensionamento e mapeamento da aplicação aos recursos é descrito com detalhes na

da aplicação aos recursos. Feito esse mapeamento, aciona-se a rotina de pré-processamento da aplicação se informando, de forma semi-automática, qual a base de dados a ser utilizada e qual o número de tasks que comporão a execução para que essa rotina possa realizar o particionamento da base, criando as sub-partições que são designadas a cada tarefa. Passadas essas fases, aciona-se o gerenciador de tarefas para que esse gere tasks concretas de acordo com o recurso onde cada uma irá ser executada e gerencie a execução das mesmas nos recursos computacionais indicados. Quando verificada a finalização das execuções das tasks da aplicação, o gerenciador de execuções recolhe os arquivos de resultados dessas e os repassa para a rotina de pós-processamento da aplicação para que essa realize a consolidação e processamento desses resultados. Enfim, o resultado consolidado é armazenado na Área de Armazenamento e indicado ao usuário para que esse possa utilizá-lo. Finalizado o processo, é realizada a limpeza dos arquivos temporários criados durante a execução.

A fim de otimizar a execução de cenários de execuções e possibilitar ao usuário um melhor controle de suas tarefas de mineração, quando é criado um cenário de execução todos os parâmetros e componentes que envolvem essa execução, como por exemplo, o algoritmo e base de dados utilizados, os parâmetros de execução do algoritmo, os recursos computacionais, bem como os arquivos das partições de dados utilizadas e os resultados dessa execução são associados a um cenário de execução. Assim, quando se deseja criar uma nova execução, pode-se verificar se já não houve um cenário semelhante e, dessa maneira, poupar alguns procedimentos principalmente quanto ao particionamento de dados. Tal armazenamento de cenários de execuções também é útil para verificação posterior dos resultados de experimentos anteriormente executados.

Esse componente também é responsável pela interação entre o usuário e a arquitetura. Para tal, fornece uma interface Web, de modo que se possa interagir remotamente com a arquitetura. A opção por uma interface Web foi adotada para facilitar o acesso ao usuário sem que ele precise instalar softwares clientes em seu ambiente de trabalho e para possibilitar que todos os componentes de gerência da arquitetura possam estar dispostos em uma única rede, diminuindo as latências de comunicação e transferências de dados entre esses componentes.

5.2.2 Repositório de Recursos

Este componente é responsável por armazenar as descrições dos recursos que estão disponíveis para a execução das aplicações dos usuários. Para cada recurso que se deseja disponibilizar à arquitetura se deve disponibilizar um descritor que informe o tipo de recurso computacional (máquina comum, *cluster* ou grade), número de processadores, número de núcleos de processamento por processador, memória, sistema operacional, portas para comunicação *rsh* e *web* e softwares oferecidos pelo recurso.

Quando um recurso é adicionado ao repositório é acionado o prospector de recursos presente no gerenciador de execuções. Esse prospector irá executar uma aplicação que visa analisar os recursos

Seção 5.3.

disponíveis no nodo, tais como memória, unidades de processamento, sistema operacional, entre outros. Além disso, executa um *benchmark* a fim de ter uma referência da capacidade computacional de um núcleo de processamento do recurso que está sendo prospectado. Dessa maneira, mesmo que a descrição dos recursos seja incompleta, pode-se obter todos os dados necessários para o escalonamento de aplicações nesses recursos por meio desse mecanismo.

As informações disponibilizadas para cada recurso servirão como parâmetro para o gerenciador de tarefas e os particionadores de dados das aplicações realizarem suas tarefas assim que o gerenciador de execuções os acionar.

No que tange à descrição de recursos e como a arquitetura os classifica, utiliza-se a taxonomia citada na Seção 5.1.2. Assim, para se cadastrar um cluster, por exemplo, deve-se cadastrar todos os nodos desses e então cadastrar um proxy para esses. Esse proxy corresponde ao nodo gerenciador do cluster. Então, associa-se a esse proxy todos os nodos do cluster previamente cadastrados.

5.2.3 Repositório de Aplicações

Este componente da arquitetura é responsável por armazenar as aplicações de mineração de dados paralelas. Para cada aplicação deve haver um descritor correspondente. Esse descritor deve conter uma breve descrição textual da aplicação, qual a tarefa de mineração e algoritmo que essa implementa, quais recursos de *hardware e software* damanda, em qual formato de base de dados que trabalha, quais os parâmetros para execução, para qual ambiente de computação se destina, o caminho de diretório que deve escrever seus resultados e suas escalabilidades internas e externas.

Uma aplicação deve ser composta por três elementos:

a) **Implementação:** Corresponde à implementação executável de uma instância de um algoritmo de mineração. Essa implementação corresponde a uma task que será executada remotamente em um recurso computacional. Deve receber como parâmetro, além dos parâmetros próprios, o arquivo de dados a ser utilizado e o número de processos internos que essa executará.

b) **Rotina de pré-processamento:** É a rotina a ser executada uma vez antes de uma execução da aplicação na arquitetura. É responsável por realizar o particionamento da base de dados e, caso necessário, preparar outros dados necessários à execução da aplicação. Deve receber por parâmetro o arquivo de dados a ser particionado, o número de partições e o padrão de nome de arquivo que deve gerar. A saída produzida é repassada ao gerenciador de tarefas para a construção das tarefas que comporão a execução da aplicação.

c) **Rotina de pós-processamento:** Esta rotina é acionada após a execução da aplicação nos recursos. É responsável por receber todos os arquivos de saída das tarefas e realizar a consolidação desses. Como exemplo, pode-se citar a consolidação que é realizada após uma execução de uma aplicação com cross-fold-validation.

Como cada execução realizada pelo usuário é armazenada no gerenciador de execuções como um cenário de execução. O gerenciador de execuções, antes de acionar os particionadores de dados de uma aplicação, verifica se esse cenário já foi realizado alguma vez e se as partições de dados geradas por esse cenário anterior ainda estão persistentes e consistentes em sua área de armazenamento. Em caso afirmativo, evita-se o reparticionamento desses dados, utilizando-se as partições já criadas anteriormente. Caso contrário, o Controlador aciona o particionador de dados para a criação de novas partições.

5.2.4 Gerenciador de Dados

O gerenciador de dados é o componente da arquitetura responsável por prover serviços e funcionalidades de manutenção e acesso a dados. Serve de apoio à execução das aplicações disponibilizadas na arquitetura. O objetivo desse componente é acelerar o processo de aquisição, transformação e particionamento de dados necessários no processo de mineração. Além disso, visa atuar como um *proxy* de acesso a dados, disponibilizando às tarefas em execução nos recursos remotos diversas maneiras de acesso a esses dados podendo aumentar o desempenho da execução total do processo de mineração. Assim, pode-se prover métodos mais adequados a determinados tipos de recursos, que podem obter melhor performance do que um método genérico para todos tipos de recursos.

O Gerenciador de dados é organizado com os seguintes componentes:

Bases de Dados:

São referências de localização para acesso às bases de dados que contém os dados a serem minerados. Essas bases podem estar atreladas a um SGBD ou arquivos em formato texto ou binário. Para cada base cadastrada, é informado o tipo de base, seu formato, driver de acesso (para bases em SGBDs) e uma descrição sobre o que representa tal base.

Em relação ao conteúdo dessas bases, presume-se que essas já contenham os dados resultantes das primeiras etapas do processo de mineração, ou seja, contenham dados já consistentes e adequados para a execução das tarefas de mineração.

Área de Armazenamento de Dados:

Consiste em um ponto de montagem de disco responsável por armazenar tanto as partições de dados quanto os resultados gerados pela execução de uma tarefa de mineração. A localização física desse ponto de montagem não implica necessariamente em um recurso local, podendo-se assim utilizar um recurso externo para o armazenamento de dados.

Esse componente também pode agregar desempenho à solução proposta pela arquitetura já que esse pode ser implantado em um recurso de armazenamento que provê formas diferenciadas e otimizadas de acesso aos dados, como um sistema de storage por exemplo, oferecendo assim melhor vazão e latência em transferências de dados.

Consolidador de Dados:

O consolidador de dados é responsável por fornecer os dados contidos nas bases de dados no formato adequado para a execução das aplicações. Quando se escolhe uma base e uma aplicação para compor uma execução, esse componente acessa os dados da base desejada e os converte para o formato requisitado pela aplicação, gravando esses dados em um arquivo na área de armazenamento de dados. Assim, na próxima utilização dessa base, o componente irá utilizar o arquivo previamente criado, ao invés de recriá-lo.

A fim de garantir que o arquivo armazenado corresponda a uma versão atualizada da base de dados, esse componente deve guardar uma referência dos arquivos de bases de dados já convertidos para um determinado formato e disponibilizar um método que atualize esses arquivos em relação ao conteúdo das bases, eliminando também as partições de dados armazenadas que derivaram de uma base já desatualizada, evitando que se minere uma versão dos dados desatualizada. Assim, quando uma base de dados for atualizada, pode-se acionar esse método para essa base e garantir a consistência dos arquivos que representam a base atualizada na área de armazenamento.

No que diz respeito ao tratamento das partições de dados geradas pelos particionadores dos algoritmos, esse componente, quando informado pelo gerenciador de execuções sobre a criação de novas partições, mantém uma referência dessas partições, atrelando à cada partição qual a base e versão dessa e qual o particionador de dados que a gerou, bem como em qual cenário de execução esta foi gerada. Esses dados são mantidos com o intuito de facilitar o reaproveitamento de particionamento de dados já realizados.

Serviço de Acesso a Dados:

Este serviço atua como um *proxy* para acesso às bases de dados, disponibilizando três formas de acesso a essa: acesso direto por ponto de montagem, acesso por cópia remota por rsh e acesso via Web service. Os dois primeiros meios são especialmente adequados, respectivamente, para nodos computacionais que estão na mesma rede da área de armazenamento e possuem um ponto de montagem que permite acesso direto a essa e para nodos computacionais que permitem conexão remota com a área de armazenamento via rsh. O terceiro meio, via Web service, traz a possibilidade de transferência remota de arquivos de dados contidos na área de armazenamento para as tarefas em execução em uma grade ou cluster que possuem nós diretamente inacessíveis via rsh ou ponto de montagem, possibilitando que esses acessem os dados por meio do serviço disponibilizado.

5.2.5 Gerenciador de Tarefas

É o componente que realiza a alocação das tarefas geradas em uma execução aos recursos computacionais e gerência a execução das mesmas nesses recursos. Este gerenciador é composto pelos seguintes componentes:

Escalonador de Tarefas

É o componente responsável pela submissão das tarefas criadas em uma execução aos recursos computacionais. É acionado pelo controlador de execuções que indica quais são as tarefas a serem executadas, seus requisitos, comandos de execução, partição de dados, arquivos necessários para as suas execuções e em quais recursos devem ser escalonadas.

De posse das tarefas repassadas pelo gerenciador de execuções, esse componente realiza a efetiva submissão de cada tarefa a um recurso e aciona a execução remota dessas. Para efetivar essa operação, é criado, para cada tarefa, um *script* que é enviado ao recurso onde a tarefa vai ser executada, juntamente com a tarefa em si (em um arquivo compactado). Esse *script* irá criar a estrutura de diretórios onde a tarefa será executada, providenciar a aquisição de dados junto ao gerenciador de dados e então, descompactar e executar a tarefa. Após essa execução, é realizado o envio dos resultados ao gerenciador de execuções, bem como o status de finalização da tarefa. Por fim, realiza-se a limpeza dos arquivos utilizados na execução remota.

O mecanismo de execução de tarefas remotas por meio de *scripts* que automatizam a execução da tarefa é modelado de forma a possibilitar que se possa disponibilizar recursos à arquitetura sem que se demande a instalação de softwares nos recursos e que a utilização desses possa ser realizada da forma menos invasiva possível.

Salienta-se que o acesso aos recursos é realizado através de seus *proxys*. Estes *proxys* têm uma interface de interação de comandos comuns a todos os proxys que faz a tradução desses comandos conforme o tipo de recurso a que tal proxy se destine. Por exemplo, se o *proxy* for destinado a clusters que demandam um gerenciador de execuções como o PBS, esse *proxy* irá converter os comandos de interação de modo que esses possam ser repassados no formato dos comandos PBS que são aceitos pelo recurso.

Monitor de Tarefas:

Este é o componente responsável por acompanhar a execução das tasks pertencentes a uma determinada execução. A monitoração é realizada através da interação com o escalonador de tarefas e visa verificar qual o *status* de cada task: se está sendo executada, foi abortada ou finalizada.

5.2.6 Considerações sobre a Arquitetura

Com a organização dos componentes apresentadas na descrição acima, entende-se que é possível gerenciar o fluxo de execução de uma aplicação de mineração paralela agregando desempenho a essa execução. Recapitulando-se, seguem esses passos (divididos em grupos) e a discussão de como a arquitetura pode agregar desempenho em cada um desses grupos:

a) (**1) Aquisição de dados de bases de dados e (2) Adequação dos dados ao formato da aplicação:** Ambos os processos são realizados pelo gerenciador de dados. Pode-se agregar

desempenho a partir da segunda execução de uma aplicação que vá utilizar as mesmas bases de uma execução prévia. Assim, essas próximas execuções utilizarão os dados já previamente adquiridos e convertidos ao formato ideal, ganhando o tempo que se levaria para realizar essas operações.

b) **(3) Dimensionamento da escalabilidade da aplicação, (4) Criação de tarefas e processos da aplicação paralela e (5) Mapeamento das aplicações aos nodos computacionais:** São processos realizados pelo gerenciador de execuções. É onde há o maior potencial de ganho de desempenho. A tarefa de mapear uma aplicação paralela a um conjunto de recursos heterogêneos é bastante complicada, a ponto de ser uma tarefa NP-completa, tornando-se ainda mais difícil quando realizada por usuários que não tem total domínio dos parâmetros a serem levados em conta nessa tarefa. Assim, um mecanismo capaz de dimensionar tarefas de acordo com os recursos existentes e levando em conta a heterogeneidade desses recursos, tende a ter uma performance muito maior do que se a mesma tarefa fosse realizada manualmente. Além disso, o mecanismo de prospecção de recursos ajuda na tarefa de obter informações dos recursos úteis para um bom mapeamento. Maiores detalhes do algoritmo utilizado para tal fim são descritos na próxima seção.

c) **(6)Particionamento dos dados a serem minerados e (7) Distribuição dos dados às tarefas nos nodos:** São realizados respectivamente pela rotina de pré-processamento da aplicação e pelo script de execução das tasks da aplicação, por meio do gerenciador de dados. No primeiro caso, destaca-se a possibilidade de o particionamento de dados ser feito de forma automatizada e de acordo com o quantidade de tasks que comporão a execução da aplicação. Além disso, há a possibilidade de se reutilizar um particionamento idêntico para futuras execuções, poupando o tempo necessário para tal. No caso da distribuição dos dados (7), destaca-se a capacidade de uma task, com base no recurso onde foi escalonada, dinamicamente buscar o meio mais eficiente para aquisição dos dados necessários, podendo agregar um desempenho considerável se comparado com uma solução genérica na qual os dados são sempre transmitidos da mesma forma. Esse desempenho tende a aumentar à medida que o tamanho das partições de dados cresça.

d) **(8) Execução das tarefas e processos:** Entende-se que a arquitetura proporciona ferramentas que permitem execução de aplicações em recursos heterogêneos sem demandar conhecimento técnico para isso, aproveitando-se da característica multinúcleo desses recursos. Além disso, graças ao sistema de *proxys* de recursos, é possível a inclusão de gerenciadores de recursos heterogêneos com relativa facilidade. Do ponto de vista de como a execução de tarefas é realizada, destaca-se a automatização do processo e integração com os demais passos da mineração. Essa etapa é efetivamente realizada pelo gerenciador de tarefas.

e) **(9) Recuperação dos resultados gerados nos nodos, (10) consolidação dos resultados e (11) Armazenamento dos resultados:** Destaca-se a automatização e dinamicidade desse

processo. Independente do número de tarefas e dos tipos de recursos utilizados, a existência de um gerenciador de dados com diversos meios de acesso facilita a centralização de todos os resultados e, consequentemente, facilita o processo de consolidação desses. Essas etapas são realizadas respectivamente pelo gerenciador de tarefas, pós-processamento da aplicação e gerenciador de tarefas, por meio da área de armazenamento.

f) **(12) Limpeza de arquivos gerados na execução:** É um processo importante, uma vez que evita a sobreutilização de disco dos recursos utilizados na execução. É realizado pelo gerenciador de tarefas.

Do ponto de vista da facilidade de utilização da estrutura, acredita-se que essa pode ser alcançada com a implementação de interfaces de usuário que auxiliem o cadastramento de recursos, aplicações e bases de dados e, por fim, a montagem de cenários de execuções, facilitando a o gerenciamento de padrões de execução e acompanhamento das tarefas nos recursos.

5.3 Algoritmo para Dimensionamento e Mapeamento Automático de Aplicações nos Recursos Computacionais

Conforme já afirmado, o dimensionamento e o mapeamento automático de uma aplicação de mineração paralela nos recursos disponibilizados pelo usuário é um dos principais objetivos e principal contribuição deste trabalho. Por isso, descreve-se com maior ênfase, nesta seção, o algoritmo desenvolvido para atingir esse objetivo. Esse algoritmo é utilizado pela arquitetura no gerenciador de execuções, especificamente no módulo de mapeamento de tarefas.

Entende-se o dimensionanento de uma aplicação paralela (*sizing*) como sendo o número de processos em que essa será dividida para a sua execução [14]. Já Mapeamento consiste na atribuição de um processo a uma unidade de processamento de um nodo computacional e a atribuição de ordem de execução a esse processo [28]. Esse mapeamento é utilizado no processo de escalonamento dos processos. De acordo com [22], o algoritmo aqui descrito, define-se como um algoritmo de mapeamento estático e *on-line*, pois o mapeamento é obtido em tempo de execução e as atribuições de tarefas a recursos não se redefine ao longo da execução.

5.3.1 Objetivo do Algoritmo

O algoritmo tem como objetivo descobrir, com base nos recursos disponíveis e na escalabilidade externa e interna da aplicação de mineração paralela a ser dimensionada e mapeada, qual o melhor dimensionamento aproximado que se pode obter para essa aplicação em virtude dos recursos compatíveis com os requisitos da mesma. Além disso, visa fornecer o mapeamento de cada processo obtido no dimensionamento da aplicação a uma unidade de processamento. Entende-se como melhor dimensionamento e mapeamento aqueles que conseguem diminuir ao máximo o tempo de execução total da aplicação (*makespan*).

O algoritmo proposto visa atender às classes de aplicações descritas na Seção 5.1.1. Visa também ser capaz de escalonar tanto tarefas BoT quanto de trocas de mensagens.

5.3.2 Entrada do Algoritmo

O algoritmo exige a seguinte entrada para efetuar suas operações:

a) **Descrição da Aplicação** - Essa descrição deve conter as escalabilidades internas e externas da aplicação, quantidade de memória necessária para executar uma instância da aplicação e os requisitos de sistema operacional e software demandados pela aplicação. Assume-se que cada instância da aplicação requisitará sempre a mesma quantidade de memória, independente de sua entrada.

   Tanto a escalabilidade externa quanto interna devem ser fornecidas no formato de um vetor. Deve conter, em ordem decrescente, todas as possibilidades de quantidades de tasks e de processos que, respectivamente, cada nível de escalabilidade pode ser dividido. Por exemplo uma aplicação cuja sua escalabilidade externa (Ee) varie de uma até dez task e contenha somente números múltiplos de 2 ($Ee[n] = Ee_{n-1} + 2 | Ee_1 = 1, Ee_2 = 2, n < 10$) e que sua escalabilidade interna (Ei) varie de um até cinco, crescendo de um em um ($Ei[n] = Ee_{n-1} + 1 | Ei_1 = 1, n < 5$) , devem fornecer os seguintes vetores:

   $Ee = \{1, 2, 4, 6, 8, 10\}$ e $Ei = \{1, 2, 3, 4, 5\}$.

   Assume-se que todas as tasks devem ter a sua escalabilidade interna regida por um único vetor Ei.

b) **Conjunto de recursos a serem utilizados** - Deve conter a descrição de todos os recursos computacionais disponíveis. Para cada recurso, deve existir a informação de quantos núcleos de processamento este possui, quantidade de memória, softwares disponíveis e um valor numérico que indique a capacidade de processamento de cada núcleo de processamento desse recurso. Esse último valor pode ser obtido por meio da execução de *benchmarks*. Para um bom resultado, cada recurso deve ser submetido ao mesmo benchmark. A estratégia de se utilizar *benchmarks* para diferenciar a capacidade de processamento em conjuntos de recursos heterogêneos é utilizada também por Bohn e Lamont em [3].

   Como o algoritmo precisa atender tanto a aplicações BoT quanto de troca de mensagens, quando um recurso se trata de um cluster e o ambiente de gerência desse cluster permite o acesso direto aos seus nodos computacionais, deve-se, além da descrição do cluster, informar a descrição de cada nodo computacional desse cluster. Desse modo, o algoritmo pode ser capaz utilizar esses nodos individualmente, facilitando a alocação de aplicações híbridas que possuem tasks do tipo BoT com processos internos implementados por meio de *threads* que demandam que os processos de cada tasks estejam alocadas no mesmo nodo computacional. Como será visto, o

algoritmo possui mecanismos para não utilizar um recurso como membro cluster e como nodo computacional isolado simultaneamente.

c) **Técnica de escalonamento de tasks** - Corresponde à técnica que deve ser utilizada no etapa de mapeamento de tasks aos recursos. Essas técnicas são discutidas na Seção 5.3.5.

5.3.3 Saída

A saída gerada pelo algoritmo é composta pelo conjunto de jobs que realizarão a execução da aplicação nos recursos. Um job significa uma submissão de um conjunto de tarefas (tasks) a um conjunto de recursos. A saída é composta de mais de um job quando os recursos disponíveis não são suficientes para realizar a execução do número mínimo de tasks demandadas pela aplicação em uma única etapa. Dessa maneira, a divisão dessa carga de trabalho em n jobs possibilita a execução do conjunto total de tarefas em n etapas separadas, que não demandam serem executadas simultaneamente. No caso de o número de tarefas mínimas exigidas pela aplicação não ter recursos suficientes para executar nem ao menos uma única tarefa do job, a saída do algoritmo será uma mensagem indicando a impossibilidade de se mapear a aplicação aos recursos disponíveis.

As possíveis saídas geradas pelo algoritmo, onde n corresponde ao número de jobs, x ao de tasks, y ao de nodos, $nProc$ ao de processos e id à chave de identificação de um recurso, são as seguintes:

a) **Saída para aplicações com tarefas BoT e processos com memória compartilhada** - É o caso para aplicações cujas tasks são mapeadas diretamente para nodos computacionais. É expressa da seguinte maneira:

$$Mapeamento = job_1 \left\{ tasks_1^{nProc} \rightarrow nodo_{id}, ..., tasks_x \right\}, ..., job_n.$$

b) **Saída para aplicações com tarefas BoT e processos com memória distribuída** - É o caso para aplicações de troca de mensagens em que suas tasks são mapeadas para escalonamento no cluster e, por isso, devem incluir quais as máquinas que vão ser utilizadas e quantos processos serão alocados a cada uma delas.

$$Mapeamento = job_1 \left\{ tasks_1^{nProc} \rightarrow \left(nodo1_{id}^{numProc,...,nodoy} \right), ..., tasks_x \right\}, ..., job_n.$$

5.3.4 Descrição do Algoritmo

Dada a entrada requerida pelo algoritmo o algoritmo realiza os seguintes passos para criar a saída esperada:

1) Seleção dos recursos compatíveis com a aplicação;

2) Definição do número de tasks e jobs;

3) Mapeamento das tasks criadas;

4) Dimensionamento do número de processos de cada task.

Segue a descrição e justificativa de cada passo.

Seleção dos Recursos Compatíveis com a Aplicação

Nesta fase, é realizada uma varredura em todos os recursos dados como entrada verificando quais possuem os requisitos de plataforma, software, quantidade de unidades de processamento e memória demandados pela aplicação. Como resultado desta fase, tem-se somente os recursos habilitados para serem utilizados no decorrer dos demais passos.

Definição do Número de Tasks e Jobs

Nesta etapa, realiza-se a primeira fase do dimensionamento automático da aplicação, o dimensionamento externo. Esse processo é feito com base nas possibilidades contidas no vetor Ee da aplicação e no total possível de alocações de tasks que podem ser realizadas nos recursos filtrados na primeira etapa. Se o total possível de alocações for menor que o menor número de tasks permitidas pela aplicação, divide-se a execução da aplicação em mais de um job, cada um contendo um subconjunto das tasks da aplicação.

A definição do número de tasks que cada recurso pode comportar ($total_{recId}$) é dada pela razão entre a quantidade de memória disponível no recurso e a quantidade de memória requisitada pela aplicação para executar uma task. Por definição, cada task deve ter no mínimo um processo e um processo deve corresponder a uma unidade de processamento. Assim sendo, caso o número de alocações de um recurso obtido pela expressão $total_{recId} = \frac{MemDoRecurso}{MemReqApp}$ exceda o número de unidades de processamento do recurso, estabelece-se que o número de tasks suportadas pelo recurso corresponde ao número de unidades de processamento disponíveis no mesmo dividido pelo número mínimo de processos internos requeridos pela aplicação, ou seja, $\frac{(Min)Ei}{NumUnidadesRec_{id}}$. Dessa maneira, o número total de tasks suportadas pelo conjunto de recursos é definido por $MaxTasks = \sum(total_{rec_1}, total_{rec_2}, ..., total_{rec_n})$.

Tendo-se o valor de $MaxTasks$, o próximo passo é procurar no vetor Ee o maior elemento cujo valor seja menor ou igual ao valor $MaxTasks$. Caso esse elemento seja encontrado, define-se o número de tasks da aplicação (dimensionamento externo) como sendo o valor do elemento encontrado e, como essa quantidade pode ser alocada nos recursos simultaneamente, define-se que a execução da aplicação se dará em um único job. A não existência de um elemento em Ee que se enquadre nessa restrição indica que é necessária a criação de mais de um job para comportar o dimensionamento externo demandado pela aplicação.

Quando é necessário mais de um job para atender a escalabilidade externa exigida pela aplicação, procura-se a escalabilidade externa que possa ser alocada em um menor número possível de

jobs e que, em cada job, melhor aproveite a capacidade de alocação dos recursos disponíveis. O procedimento que realiza essa divisão opera da seguinte forma: partindo do número de jobs (nj) igual a 2, incrementando-o em um a cada interação, busca o maior elemento (Ee_i) em Ee cujo valor atenda a seguinte regra: $\frac{Ee_i}{nj} \leq MaxTasks$. Se não for encontrado nenhum elemento que satisfaça a regra, incrementa-se o número de jobs ($nj+1$). Caso contrário, define-se nj como o sendo o total de jobs a serem criados e o total de tasks é então dividido igualmente entre a quantidade de jobs definidos. Caso a divisão de tarefas por job não seja exata, distribui-se, para cada job, a quantidade de tarefas que correspondem a parte inteira dessa divisão e se distribui as tasks que correspondem ao resto da divisão da maneira igual entre os jobs. Assim, obtem-se uma divisão onde a diferença da quantidade de tasks entre um job e outro será de no máximo uma tasks e cada job consegue aproveitar o máximo possível os recursos aptos a execução de tasks da aplicação. Por exemplo, a alocação de 15 tasks em um conjunto de recursos que suportam quatro tasks por vez, resultaria em três jobs com quatro tasks e um job com três tasks.

5.3.5 Mapeamento das Tasks nos Recursos

Esta etapa do algoritmo corresponde à fase a qual ocorre o mapeamento de cada task de um job a um recurso computacional. Esse recurso pode ser de dois tipos, dependendo do tipo de tarefa que se está mapeando: (a) tarefas com processos que demandam exclusivamente memória compartilhada e (b) tarefas com processos que permitem memória distribuída. Para o primeiro caso, pode-se mapear as tarefas em nodos computacionais. No segundo caso, pode-se mapear as tarefas tanto para nodos computacionais, quanto para um cluster. No caso de mapeamento para um cluster, ainda são indicados quais os recursos utilizados pela tarefa e quantas tarefas devem ser alocados em cada recurso.

Para realizar essa tarefa se modelou o problema de modo que esse pudesse ser solucionado pelas técnicas de escalonamento do problema de *Bin Packing* [4, 9]. Segue uma breve descrição desse problema e suas soluções.

Problema de Bin Packing

É um problema de otimização combinatória [4] que consiste em alocar (*packing*) n itens de tamanho Ti em k caixas ou repositórios (*bins*) de tamanho Tc, sendo que esses tamanhos podem variar. Existem diversas variações dessa técnica dependendo do objetivo ao qual essa é aplicada. Esses objetivos podem utilizar o melhor balanceamento de carga entre as caixas, utilizar o menor número de caixas, entre outros. Ainda pode haver variações onde o tamanho das caixas e/ou itens podem ser heterogêneos. Pode-se classificar a dimensão do problema de acordo com o número de fatores que são levados em conta para a alocação dos itens nas caixas. *Bin packing* unidimensional corresponde a problemas onde somente um fator é utilizado para definir a alocação e assim por diante. Por exemplo, na alocação de contêineres em navios, o fator a ser analisado pode ser o

espaço físico que os contâineres ocupam e o espaço físico disponível nos navios. Esse caso seria unidimensional. Se se adicionar ao problema o fator peso dos contêineres e peso suportado pelos navios, tem-se um problema bidimensional.

O problema de *bin packing* é um problema NP-completo [4], ou seja, problema que não pode ser solucionado em tempo polinomial determinístico. Por isso, para esse tipo de problema, utiliza-se soluções alternativas que obtém uma solução aproximada da solução ótima para o problema em tempo polinomial determinístico, possibilitando resultados relativamente satisfatórios em um tempo possível. Entre esse tipo de solução que visa um aproximação da solução ótima, destaca-se o uso de heurísticas.

Heurísticas consistem em soluções algorítmicas exploratórias simplificadas para um determinado problema em que somente se pode mensurar sua eficiência de forma empírica, ou seja, apresentam soluções não-determinísticas. Para o problema de *bin packing* se utiliza de heurísticas que usam estratégias para decidir em qual caixa cada item será alocado. Essas estratégias podem ser *on-line* ou *off-line*. Apresenta-se as estratégias *on-line* que são aquelas que processam a alocação de cada item individualmente e sequencialmente, analisando uma caixa por vez. Não há conhecimento de todos os componentes do cenário do problema e, além disso, uma decisão, quando tomada, não é mais alterada. Essas estratégias são do tipo *any-fit*, pois consideram que um item pode ser alocado em qualquer recipiente que esteja aberto à alocação.

Seguem três estratégias que, de acordo com Coffman et al. [9] são as mais utilizadas, conhecidas e experimentadas entre as técnicas existentes:

a) **First-Fit (FF):** Aloca o item na primeira caixa aberta à alocação (da esquerda para direita) na qual este item couber.

b) **Best-Fit (BF):** Aloca o item na caixa onde houver o menor espaço vazio e o item couber, ou seja, onde o item melhor se encaixa, deixando o menor espaço restante possível.

c) **Worst-Fit (WF):** É o oposto do BF. Aloca o item na caixa aberta à alocação onde houver o maior espaço restante e o item couber, deixando o maior espaço vazio possível.

Ainda existe a variação na qual se ordena os itens em ordem decrescente de tamanho antes de se fazer a alocação desses, formando assim o *First-Fit-Decreasing (FFD), Best-Fit-Decreasing (BFD) e Worst-Fit-Decreasing (WFD)*.

Modelagem do Problema de Mapeamento de Tasks aos Recursos

Modelou-se o problema de mapeamento das tasks da aplicação aos recursos como um problema de *bin packing on-line*, onde, analogicamente, as tasks a serem mapeadas correspondem aos itens e os recursos computacionais correspondem às caixas do problema de *bin packing*. Os fatores a serem levados em conta para alocação das tarefas são: memória disponível no recurso em relação à

requisitada pela task e o número de unidades de processamento disponíveis no recurso em relação ao exigido pela task.

Como o cenário do problema do algoritmo de mapeamento difere do apresentado no problema original de *bin packing*, algumas questões obrigaram a realização de redefinições nas premissas dos algoritmos de *bin packing* originais para que se pudesse adaptar a solução à realidade do problema. Segue a discussão dessas questões e as modificações realizadas pelo algoritmo proposto em relação ao problema de *bin packing* tradicional:

a) **Heterogeneidade dos recursos:** O fato de os recursos (recipientes) serem heterogêneos, implica que esses podem ter diferentes capacidades de processamento. Assim sendo, pelo fato de as heurísticas apresentadas percorrerem os recursos (recipientes) da esquerda para a direita com a premissa de que esses são iguais, pode ocorrer que recursos de menor capacidade de processamento serem superutilizados em relação aos de maior capacidade, principalmente para FF. Essa caso acontece quando a maioria dos recursos de menor capacidade ficarem nas primeiras posições da lista de recursos.

   Para contornar essa situação, o algoritmo aqui proposto ordena os recursos em ordem decrescente, de acordo com as suas capacidades de processamento (valor obtido por *benchmark*). Esta medida faz com que os recursos de maior capacidade de processamento sejam preferidos em relação aos de menor. Além disso, na ordenação dos recursos, outro fator é levado em conta: a possibilidade de existência de recursos multiprocessados. Com isso, assume-se que a capacidade de um recurso computacional equivale à soma das capacidades de todas as suas unidades de processamento. Essa medida evita que recursos com um único processador com capacidade elevada sejam considerados mais capacitados do que recursos com mais de uma unidade de processamento os quais, embora tenham a capacidade individual de processamento dessas unidades menor, possuam uma capacidade agregada de processamento muito maior por possuírem diversas unidades de processamento. Essa medida adquire importância ainda maior quando as tasks a serem mapeadas permitem uma escalabilidade interna com muitos processos.

b) **Limitação do número de itens por recipiente:** Como a taxonomia de tarefas adotada neste trabalho define que uma task deve conter ao menos um processo e um processo deve ser mapeado exclusivamente a uma unidade de processamento, o número de tasks alocadas a cada recurso computacional (recipiente) não deve exceder o número de unidades de processamento existentes no recurso.

c) **Redefinição do critério de *best* e *worst-fit*:** Na definição formal de bin-packing o fator a ser levado em conta para definir o que seria a melhor ou a pior acomodação de um item (*best/worst-fit*) é o fator espaço disponível do recipiente em relação ao espaço necessitado pelo item. Se o algoritmo de mapeamento seguisse essa analogia, o fator a ser levado em conta para definir melhor/pior acomodação seria o fator memória. No entanto, como o objetivo do algoritmo é obter

aceleração da execução da aplicação, o fator memória disponível não é o principal responsável pelo atingimento desse objetivo. Um processo que ocupa no máximo 100MB de memória não terá sua execução mais rápida se, por exemplo, for disponibilizado 1000MB a esse. Entende-se que o fator *capacidade computacional* expressa melhor o papel de influente no tempo de execução de um processo. Um processo de computação intensiva tende a ser executado mais rapido à medida que a unidade de processamento onde esse está sendo executado aumentar sua capacidade de processamento.

Portanto, o fator que é levado em conta para definir qual é a melhor ou pior acomodação para uma task é a capacidade de processamento restante no recurso. Essa capacidade é expressa pelo somatório da capacidade de processamento de cada unidade de processamento do recurso que ainda não foi alocada a alguma task. Ainda para satisfazer a possibilidade de alocação de uma task, um recurso deve possuir ao menos uma unidade de processamento não alocada e memória suficiente ainda não alocada para ser disponibilizada à aplicação.

d) **Eliminação de recursos de baixa capacidade para best-fit:** Novamente, devido à heterogeneidade dos recursos, quer-se evitar que na aplicacão de BF, quando houver recursos mais que suficientes para se executar uma aplicação, elimine-se os recursos considerados com capacidade de processamento muito limitada. Essa medida é realizada para evitar que somente esses recursos sejam utilizados devido à propriedade de BF de escolher sempre os recursos de menor capacidade restante.

Para realizar essa eliminação, verifica-se, sobre o vetor de recursos já previamente ordenados pela ordem de capacidade de processamento, qual o conjunto mínimo de recursos necessários para a alocação. Esse conjunto, denominado A, irá corresponder aos recursos mais capacitados do total de recursos. Então, sobre os recursos restantes, representados pelo conjunto B, aplica-se a seguinte heurística sobre cada recurso B_i que irá definir se B_i deve ou não ser inserido em A: se a capacidade computacional total de B_i for até 30% menor que a capacidade computacional total do recurso menos capacitado de A, incluí B_i em C, caso contrário, descarta-se B_i. Após a aplicação da heurística em cada recurso, incluí-se C em A e define-se A como sendo o conjunto de recursos aptos a receber tasks.

e) **Número de recipientes disponíveis para alocação fixo:** No problema de *bin packing* se define que somente uma quantidade x constante do total k de recipientes pode estar aberta para alocação. Essa limitação é utilizada para evitar que a solução do problema se torne computacionalmente inviável à medida que o número de recipientes cresça infinitamente.

Como o presente algoritmo tem como objetivo ter todos os recursos disponíveis para utilização pelo usuário, define-se que a quantidade de recursos (recipientes) abertos à alocação dada por x, será sempre igual a quantidade total dos recursos disponíveis representada por k ($x = k$). Esse fato implica na limitação da quantidade de recursos que podem ser usadas pelo algoritmo. Nos

testes realizados, constatou-se que até 50000 recursos podem ser utilizados sem que a execução do algoritmo ultrapasse o tempo de dois minutos.

Aplicação das Heurísticas de Mapeamento

Definidas as modificações a serem realizadas para a aplicação das heurísticas de *bin packing*, finalmente se realiza a execução do algoritmo que objetiva mapear as tarefas de cada job aos recursos computacionais. O referido algoritmo é composto pelos seguintes passos: (a) Definição de visão de recursos e (b) mapeamento das tarefas.

O primeiro passo diz respeito à definição de como os recursos serão vistos para a alocação das tasks. Essa definição se dá dependendo da forma como os processos das tasks que compõem o job demandam a memória dos recursos. Caso os processos permitam memória distribuída, o algoritmo trata os clusters como um único recurso ao qual uma task pode alocar seus processos em mais de um nodo computacional. Caso contrário, quando os processos de uma task precisarem ser executados em memória compartilhada, o algoritmo trata cada nodo computacional de um cluster como um recurso separado.

Definida a forma como os recursos serão tratados, é aplicado, para cada task, o algoritmo baseado em *bin packing* com uma das heurísticas citadas (FF,BF ou WF indicada na entrada do algoritmo) com as modificações apresentadas na seção anterior. O resultado da aplicação desse passo é o mapeamento de cada task em um determinado recurso. A cada mapeamento realizado é subtraída da quantidade de memória do recurso o qual a task foi alocada a quantidade de memória necessitada pela task. Também se subtrai da quantidade de unidades de processamento desse recurso a quantidade mínima de processos demandado pela task (Ei_1). Assim, a cada task a ser mapeada, tem-se atualizados os valores de quantidade memória e unidades de processamento dos recursos, além disso, é recalculada a capacidade total de processamento desses recursos com base na quantidade de unidades de processamento livres.

Terminados os passos de mapeamento das tasks nos recursos, passa-se ao último passo do algoritmo aqui descrito: o dimensionamento do número de processos de cada task.

5.3.6 Dimensionamento do Número de Processos de Cada Task

Recapitulando, nesse ponto da execução do algoritmo já se tem a definição de quantos jobs a execução da aplicação será dividida e de quantas tasks cada job será composto. Também se tem a definição de a qual recurso cada task está mapeada. Logo, seguindo a linha de execução do algoritmo aqui proposto, o passo final desse consiste no dimensionamento automático da escalabilidade interna de cada task. Nesse passo, objetiva-se definir em quantos processos cada task será executada. Procura-se, sempre, alocar a cada task o maior número de unidades processamento possível de acordo o vetor de escalabilidade interna (Ei) fornecido na entrada do algoritmo e no total de unidades de processamento disponíveis no recurso onde a task será executada.

Dado esse objetivo, pode-se perceber que a expressão que representa a melhor distribuição da unidades de processamento de um recurso para as tasks nele alocadas é dada pela razão entre a quantidade de unidades de processamento desse recurso e o total de tasks nele alocadas. Além disso, leva-se em conta que cada task deve ter, nessa divisão, o maior número possível de unidades alocadas.

O procedimento para dimensionamento do número de processos de cada task é executado na seguinte sequência: primeiramente, reinicia-se ao valor original (sem alocações) a quantidade total de unidades de processamento de cada recurso. Então, para cada recurso R_i, encontra-se um valor q que representa a maior quantidade uniforme de unidades de processamento que todas as tasks alocadas no recurso podem receber, respeitando sua escalabilidade Ei. Após, é atribuída para cada task T_i o número q de processos e, em cada iteração, se subtrai do total de unidades de processamento de R_i o valor de q.

Se após esse processo ainda houver unidades de processamento não alocadas em R_i, tenta-se alocá-las, sempre respeitando a escalabilidade Ei, para o maior número possível de tasks alocadas em R_i. Essa redistribuição é realizada da seguinte maneira: para cada task T_i em R_i, até que não haja mais unidades de processamento não alocadas em R_i ou não seja possível realizar alguma alocação, verifica-se se há unidades de processamento livres que possam ser alocadas a T_i de forma a satisfazer a próxima possibilidade de alocação maior do que a quantidade de processos já alocados para T_i na primeira distribuição de processos. Caso sim, aloca-se a quantidade necessária de unidades de processamento a T_i de modo que se possa aumentar sua escalabilidade, subtraindo-se essa quantidade da quantidade de unidades sem alocação.

Com isso, divide-se o número de unidades de processamento de forma uniforme entre as tasks alocadas no recurso respeitando as diretivas de escalabilidade interna dessas tasks, obtendo-se a melhor utilização das unidades do recurso.

Quando a alocação é realizada em um cluster, procura-se alocar os processos de cada tarefa no menor número possível de nodos. Isso é feito para facilitar a comunicação entre esses processos, mantendo-os o mais próximos possível a fim de diminuir a latência inerente da comunicação.

Com a finalização desse processo, encerra-se a execução do algoritmo de mapeamento proposto, tendo como resultado o mapeamento e dimensionamento de jobs, tasks e processos que compõem uma execução de uma aplicação paralela.

Descrito algoritmo proposto, segue algumas considerações sobre as funcionalidades do mesmo.

5.3.7 Considerações sobre o algoritmo

Quanto ao algoritmo aqui apresentado, pode-se destacar as seguintes funcionalidades:

a) Capacidade de gerir escalabilidades dinâmicas tanto no nível externo quanto interno, possibilitando a execução de uma série de tipos de aplicações comuns para a execução de mineração paralela, permitindo variações de escalabilidade dessas.

b) Capacidade de trabalhar tanto com BoT e troca de mensagens, tanto em clusters, quanto em nodos isolados, trazendo a possibilidade de aproveitar os recursos computacionais da forma mais adequada, de acordo com o tipo de aplicação que se deseja executar.

c) Adequação de heurísticas de *bin packing* para o uso em ambientes heterogêneos, possibilitando o uso de métodos reconhecidos de modo que se possa trabalhar com recursos heterogêneos, levando em conta, especialmente, a presença de recursos multinúcleos.

Como limitações, pode-se citar o fato de o uso de valor de memória requisitado pela aplicação ser sempre fixo, quando na verdade se sabe que este valor pode variar de acordo com a entrada dessa. Além disso, o método de composição do valor de desempenho de um recurso com várias unidades de processamento é adequado a aplicações com escalabilidade interna alta, podendo ser prejudicial a aplicações com escalabilidade interna baixa, alocando-as a recursos com unidades relativamente fracas computacionalmente somente por esse possuir várias unidades.

Finalizado a descrição de todos os componentes da solução proposta, segue a descrição dos experimentos realizados a fim de se avaliar as funcionalidades das ferramentas propostas.

6. Avaliação da Arquitetura e Algoritmo de Mapeamento Propostos

Este capítulo apresenta os experimentos realizados a fim de se avaliar a arquitetura proposta. Para isso, construiu-se uma implementação de um protótipo para teste, no qual se realizou a execução da paralelização realizada para o E-Motion e da aplicação k-Means testada. Sobre essas execuções se analisou o ganho que se pode obter com a utilização das ferramentas da arquitetura em relação ao *overhead* resultante de sua utilização. Faz-se também uma análise de viabilidade do gerenciador de dados, mostrando-se os ganhos que podem ser obtidos através do seu uso.

No que diz respeito à avaliação do algoritmo de mapeamento proposto, mostra-se resultados de experimentos que visam reproduzir a realidade das taxonomias de aplicações suportadas pela arquitetura, buscando-se analisar o impacto e a eficiência dos mecanismos de dimensionamento e mapeamento propostos.

Segue portanto os resultados obtidos pelos experimentos dessa avaliação, iniciando-se pela descrição dos cenários utilizados para os experimentos.

6.1 Descrição dos Cenários Utilizados

Construiu-se três cenários de recursos para servirem de plataforma para a execução dos experimentos de avaliação. Nesses cenários, têm-se cinco tipos de nodos computacionais. Esses nodos correspondem a nodos reais presentes no Laboratório de Alto Desempenho da PUCRS (LAD/PUCRS) [32]. Os atributos de cada um foram obtidos através da execução real do prospector de recursos implementado no protótipo da arquitetura, que é descrito na Seção 6.3.1.

Segue, na Tabela 6.1, [1] a descrição de cada um dos tipos de nodos computacionais utilizados, bem como os valores dos seus atributos que são considerados relevantes para as análises realizadas neste capítulo. Tendo-se esses recursos descritos, criou-se três cenários com a combinação dos

Tabela 6.1 – Descrição dos tipos de nodos computacionais utilizados nos experimentos.

Nome	Arq.	Capac. comp.	Memória	SO	Num. Cores
Cerrado	IA64	300	2002(MB)	Linux Debian	2
Amazônia	i386	250	256(MB)	Linux Debian	1
Pantanal	x64	1218	2014(MB)	Linux Ubuntu	2
Atlântica	x64	6984	15870(MB)	Linux Ubuntu	8
VM-Atlântica	x64	6224	7906(MB)	Linux Ubuntu	8

[1] Faz-se uma ressalva em relação ao resultado obtido pela execução do *benchmark* para o recurso Cerrado, que não correspondeu ao esperado e informado pelo fabricante. A possível explicação para tal é a arquitetura do processador do recurso (Itanium II, arquitetura ia64) que, por estar descontinuada, pode não conter suporte as instruções testadas pela aplicação de *benchmark*. Como os experimentos visam retratar a realidade da execução da arquitetura, se mantém o valor encontrado como referência de desempenho do recurso para o algoritmo de mapeamento tomar decisões, mesmo que essas possam surtir um impacto negativo.

mesmos. O objetivo dessas combinações é ter um cenário real (*Cenário A*), que é um subconjunto dos recursos presentes no LAD/PUCRS, e dois cenários artificiais que correspondem a um cenário relativamente pequeno (*Cenário B*) e a um cenário com uma quantidade significativa de recursos (*Cenário C*). Assume-se que, para cada cenário, existem cinco clusters de nome igual aos recursos apresentados, cada um contendo uma certa quantidade de nodos correspondentes aos recursos de mesmo nome. Esses clusters são todos gerenciados por uma única máquina com processador Dual Xeon 2.8Ghz e 2GB de memória, denominada *marfim*. A conexão de rede entre cada cluster e a *marfim* é uma conexão *fast ethernet* (100/10). A Tabela 6.2 mostra cada cenário e a respectiva quantidade de nodos que cada cluster têm. A execução do protótipo implementado e demais expe-

Tabela 6.2 – Quantidade de nodos computacionais presentes em cada cluster de cada cenário para experimentação

Cenário	Cerrado	Amazônia	Pantanal	Atlântica	VM-Atlântica
A	4	15	4	1	1
B	2	2	2	2	2
C	25	25	25	25	25

rimentos foram realizados em uma máquina idêntica à máquina Atlântica com sistema operacional Linux Ubuntu 9.10. Todas as implementações realizadas foram feitas utilizando a linguagem de programação Java 6. Assume-se que todos os recursos estão exclusivamente alocados para uso da arquitetura, sem outras aplicações sendo executadas e possuem acesso RSH (Remote Shell) habilitado ao usuário configurado na arquitetura.

A fim de se obter uma melhor acurácia nos resultados apresentados, cada experimento foi repetido cinco vezes. Dessas cinco execuções, eliminou-se as de maior e menor tempos e se compôs o resultado através da média das três execuções restantes.

Apresentados os cenários de execução dos experimentos, inicia-se a apresentação dos experimentos de avaliação primeiramente com a avaliação do algoritmo de mapeamento, seguida da avaliação da arquitetura como um todo e, por fim, é apresentada uma discussão sobre os resultados apresentados.

6.2 Experimentos para Avaliação do Algoritmo de Mapeamento da Arquitetura

Esta seção mostra os experimentos realizados para avaliar o desempenho do algoritmo de mapeamento proposto quando esse é utilizado pelo mapeador de recursos da arquitetura. Avalia-se qual a aceleração da execução da aplicação de mineração que se consegue obter utilizando cada estratégia de alocação de recursos (FF, BF e WF) nos cenários de recursos apresentados. Para complementar a avaliação, realiza-se o cálculo de uma função de custo de execução para cada combinação de técnica e cenário de recursos. Essa avaliação complementar visa obter uma avaliação da eficiência de utilização dos recursos e obter uma relação de custo/benefício de cada estratégia de mapeamento em cada cenário. Ainda, a fim de ilustrar a quantidade de recursos alocados para cada caso de teste,

mostra-se a quantidade de unidades computacionais de cada tipo de recurso que são alocadas para a execução.

Objetivando melhor comparar o desempenho de cada técnica de alocação utilizada implementou-se uma técnica de alocação aqui denominada Random. Essa técnica, como o próprio nome sugere, faz uma alocação aleatória das tasks nos recursos. Mesmo essa alocação sendo aleatória, continua respeitando as regras de alocação de uma unidade de processamento para cada processo da task. Do ponto de vista de *bin packing*, pode-se classificá-la como uma variação de First-Fit com embaralhamento de recipientes heterogêneos antes da alocação dos itens.

Para os estudos de caso, procurou-se utilizar aplicações que se classificam na taxonomia de aplicações suportada pela arquitetura.

A seguir, mostra-se as métricas de desempenho utilizadas e como se procedeu para avaliar o desempenho das aplicações usadas como casos de teste:

a) ***Aceleração do Makespan da Execução da Aplicação:*** Consiste na aceleração (Speed-Up) do tempo total de execução de uma aplicação paralela (todos os jobs e tasks que a compõe) sobre determinados recursos em relação ao tempo de execução da mesma aplicação realizada em um outro recurso referencial. Para os resultados apresentados, utilizou-se como referência para medição da aceleração do makespan dos estudos de caso a execução de cada aplicação em um processador de um nodo computacional do cluster Atlântica.

   A medição da aceleração é realizada visando combinar as acelerações que podem ser obtidas tanto para o nível de paralelização interna, quanto externa. A avaliação da aceleração com a paralelização interna é obtida da seguinte maneira: para cada aplicação a ser avaliada se tem uma tabela que indica a aceleração da aplicação em relação a execução de referência (na máquina Atlântica com um processo) de acordo com o número de processos que esta aplicação está executando em um determinado recurso.

   Para a avaliação da aceleração em virtude da paralelização externa, ou seja, execução paralela de diversas partições de dados em diversas instâncias individuais da aplicação, assume-se que a aceleração obtida é sempre linear e ótima. Dessa maneira, uma aplicação que é executada com o particionamento de seus dados em duas partições terá uma aceleração de duas vezes em relação a sua execução sequencial com a base de dados sem particionamento e assim por diante. Obviamente as execuções de cada instância da aplicação devem se dar em paralelo sem concorrência de processadores.

   Para ilustrar, pega-se como exemplo uma aplicação que utilizando um processador de um nodo da Atlântica e, não utilizando paralelismo externo, leva 10 minutos para ser executada. Essa mesma aplicação, quando executada com uma task com quatro processos num nodo VM-Atlântica, tem seu tempo de execução diminuído em duas vezes, ou seja, tem aceleração interna de ordem dois em relação à Atlântica. Já, quando se realiza essa execução em duas instâncias externas paralelas, cada uma com 50% dos dados da base de dados, essa execução demanda a metade do tempo que

demandaria na execução sem o particionamento da base, ou seja, teve uma aceleração externa na ordem de duas vezes em relação a sua versão sem particionamento externo. Assim sendo, se essa aplicação fosse executada com particionamento externo de ordem dois e executada em duas instâncias, cada uma em um nodo VM-Atlântica, com tarefas com quatro processos para cada, teria uma aceleração de quatro vezes em relação a sua versão sequencial sem particionamento (2 vezes devido ao particionamento externo $\times$ 2 vezes devido ao particionamento externo), tendo seu tempo de execução diminuído para 2.5 minutos, pois cada execução, de cada instância, acabaria ao mesmo tempo. Portanto, a fórmula para se calcular a aceleração de uma task em determinado recurso é dada por $AceleraçãoInterna \times EscalabilidadeExterna$.

Como por diversas vezes pode ocorrer de existirem tasks heterogêneas em relação as suas quantidades de processos, e ainda ocorrer de uma execução ser dividida em diversos jobs, faz-se o cálculo explicado acima para cada task que compõem um job. Esse cálculo é feito com o fim de se obter o tempo que cada task leva para ser executada. Para definir o tempo de execução de um job, assume-se que esse tempo será o tempo de execução da tarefa mais demorada entre aquelas que o compõem. Para se obter o tempo total de uma execução que demanda diversos jobs soma-se os tempos de execução de todos jobs que a compõem. Por fim, o cálculo da aceleração do makespan da execução é dado por $\frac{tempoReferência}{SomaDoTempoDosJobsDaExecução}$ e é este o valor apresentado nos resultados.

b) ***Eficiência da Alocação de Recursos:*** Conforme visto na Seção 2.1, o cálculo de eficiência para recursos homogêneos é bastante simples. Porém, esse cálculo não é adequado quando se trata de recursos heterogêneos. Em recursos heterogêneos se tem unidades de processamento com desempenhos diferentes entre si, fazendo com que o speed-up de uma aplicação executada num recurso x seja diferente do speed-up da mesma aplicação quando executada num recurso y. Assim sendo, em aplicações nas quais se utilizam processadores heterogêneos para suas execuções, caracteriza-se um erro utilizar um valor de eficiência único, pois as variáveis que o compõem derivam de cenários de execução diferentes.

Diante desse problema, propõe-se avaliar a eficiência da execução das aplicações utilizando-se uma fórmula que intenta obter um preço que seria pago para executar a aplicação, caso essa execução fosse tarifada. Para isso, define-se tarifas de alocação para cada nodo computacional. Essas tarifas variam de acordo com o tempo de alocação do nodo e de acordo com o número de unidades de processamento alocadas no recurso.

Como referência para os experimentos realizados se utilizou as tarifas expostas na Tabela 6.3. Cada valor corresponde ao preço de alocação de uma unidade de processamento de um recurso pelo tempo de um minuto. O preço é dado em uma unidade monetária fictícia. Os valores atribuídos procuram refletir a capacidade computacional de cada recurso (de acordo com o afirmado pelo fabricante), atribuindo preços mais altos a recursos considerados mais nobres.

Tabela 6.3 – Custo de alocação de uma unidade de processamento pelo tempo de um minuto.

Recurso	**Atlântica**	**VM-Atlântica**	**Pantanal**	**Cerrado**	**Amazônia**
Custo	\$2,00	\$1,50	\$1,00	\$0,30	\$0,10

Definidas as tarifas de cada recurso, o cálculo do custo de alocação da aplicação em determinado recurso é baseado na multiplicação do tempo em minutos alocados no nodo pelo valor da multiplicação do número de unidades de processamento utilizadas pelo preço de alocação de cada unidade. O custo total da execução da aplicação é dado pela soma dos custos de cada alocação realizada em um determinado recurso.

A fim de testar todos os cenários de aplicações descritos na taxonomia da arquitetura, utilizou-se, além das aplicações E-Motion e K-Means, cenários com aplicações sintéticas. Como nesse tipo de cenário, por não se poder fazer nenhum experimento baseado em execuções reais não se tem conhecimento da aceleração interna das aplicações em cada recurso, utilizou-se valores sintéticos para representar essa aceleração. Os valores definidos procuram expressar a realidade da maioria das aplicações paralelas, representando uma aceleração linear, porém não ótima.. A Tabela 6.4 mostra a aceleração interna utilizada como referência para os cálculos dos resultados que envolvem aplicações sintéticas para até 5 processos, nos casos que demandam mais processos, a aceleração é expandida, seguindo o padrão apresentado na tabela. Conforme se pode observar, essa aceleração tem como referência a execução com um processo num nodo Atlântica (aceleração 1). Novamente, se seguiu como orientação o desempenho alegado pelos fabricantes dos processadores dos recursos para se definir a aceleração em cada recurso.

Tabela 6.4 – Aceleração interna utilizada como referência para a apresentação de resultados de aplicações sintéticas.

$\frac{Recurso}{NúmeroDeProcessos}$	**Atlântica**	**VM-Atlântica**	**Pantanal**	**Cerrado**	**Amazônia**
1	1	0,9	0,7	0,3	0,2
2	1,85	1,67	1,3	0,56	0,37
3	2,70	2,43	1,89	0,81	0,54
4	3,55	3,20	2,49	1,07	1
5	4,40	3,96	3,08	1,32	0,88

Definidos os parâmetros de custo e aceleração das aplicações, apresenta-se os resultados das simulações realizadas para se avaliar o desempenho do algoritmo de mapeamento proposto neste trabalho. Os casos de teste são divididos em dois tipos, de acordo com a sua escalabilidade externa: os que tem essa escalabilidade fixa e os que tem essa escalabilidade variável. Para cada caso de teste mostra-se as entradas fornecidas para simulação dos resultados e os gráficos para análise de desempenho. Ao fim da apresentação dos casos de teste se apresenta uma discussão sobre os resultados obtidos.

Para ampliar o número de testes, em todos os casos que envolvem aplicações sintéticas, fez-se

duas simulações, cada uma com um requisito de memória diferente: uma com 1000MB e outra com 4000MB de memória requisitados. Essas quantidades de memória são representadas, respectivamente, pelos símbolos 1K e 4K.

Os resultados são apresentados da seguinte forma: mostra-se uma tabela com os valores de entrada do algoritmo e os valores de referência utilizados para calcular os valores apresentados nos resultados. Nessa tabela o requisito de memória apresentado diz respeito à execução de uma única task da aplicação. O fator compartilhamento de memória indica se os processos da task permitem uso de memória compartilhada (como em casos implementados com threads) ou distribuída (como em casos implementados com MPI para clusters). Após, apresenta-se gráficos que mostram, de acordo com a técnica de alocação de tasks utilizadas, quantos processadores foram alocados para a execução da aplicação, incluindo a proporção de cada recurso utilizado, qual o speed-up do makespan da aplicação e, por fim, qual o custo da execução da aplicação. Cada gráfico contém os dados citados acima dispostos para cada cenário de recurso utilizado, incluindo a quantidade de memória demandada pela aplicação. Essa informação (cenário de recursos e memória) é disposta da seguinte maneira $Cenário(A|B|C) : Memória$. No fim de cada estudo de caso faz-se uma exposição da quantidade total de jobs e tasks que foram criados para cada execução e que são invariáveis para todos os métodos de alocação. O número de processos total no entanto, é variável para cada processo, pois depende de onde as tasks estão alocadas. Essa dimensão do mapeamento pode ser analisada através do gráfico de alocação apresentado em cada caso. A distribuição de tasks por job pode ser obtida com $\frac{TotalDeTasks}{TotalDeJobs}$.

Seguem, nas seções seguintes, os resultados dos experimentos realizados.

6.2.1 Caso de Teste I: Resultado do Teste com Escalonamento Externo Dinâmico

Esta seção apresenta um caso de teste que possui escalabilidade externa dinâmica, ou seja, o qual pode variar o número de partições de dados que podem ser criadas para serem analisadas paralelamente e distribuidamente. Um exemplo de aplicação desse caso é a classificação de uma base de dados não classificada sobre um classificador já construído.

A Tabela 6.5 mostra as informações referentes a aplicação de escalonamento dinâmico. Essa aplicação é uma aplicação sintética.

Tabela 6.5 – Dados de entrada para o Caso de Teste I.

Escalabilidade externa	[1 a 20]
Escalabilidade interna	[1 a 10]
Compartilhamento de memória	Memória compartilhada (threads)
Tempo da aplicação sequencial	10000 minutos
Custo da aplicação sequencial	$20000,00

A Figura 6.1 mostra os resultados dos experimentos referentes ao Caso de Teste I.

A Tabela 6.6 exibe o dimensionamento criado automaticamente em cada cenário no Caso de Teste I.

Tabela 6.6 – Dimensionamento de jobs e tasks para o Caso de Teste I.

Dimensionamentos / *Cenário*	**Quantidade de jobs**	**Quantidade de tasks**
A:1K	1	20
A:4K	1	4
B:1K	1	20
B:4K	1	8
C:1K	1	20
C:4K	1	20

A seção seguinte apresenta os resultados dos testes com escalonamento externo estático.

6.2.2 Resultados dos Testes com Escalonamento Externo Estático

Nesta seção são mostrados os resultados de casos de testes os quais se tem a escalabilidade externa fixa, ou seja, tem-se sempre o mesmo número de instâncias de execuções e se tem um valor fixo para particionamento de dados. Nesses casos se incluem todas as aplicações que realizam *cross-fold-validation* ou *bagging*.

A fim de avaliar o comportamento do mapeamento com essa classe de aplicação, testou-se as aplicações E-Motion paralela com utilização de *10-cross-fold-validation* e a implementação do algoritmo k-Means usando OpenMP. Também se procurou avaliar a execução de aplicações de troca de mensagens utilizando a visão de recursos como clusters para alocação dos processos internos. Para isso, modelou-se duas aplicações sintéticas de troca de mensagens para serem executadas em clusters. Como o principal objetivo era avaliar o comportamento do mapeador com clusters, utilizou-se como parâmetro somente a técnica Worst-Fit para comparação de desempenho com a técnica aleatória (Random).

Os valores de aceleração interna utilizados para simulação das aplicações reais foram conseguidos através de resultados obtidos pela execução real dessas aplicações em cada nodo computacional utilizado. As aplicações sintéticas para cluster seguem o padrão de aceleração definido para aplicações sintéticas.

Seguem os resultados de cada Caso de Teste realizado.

Caso de Teste II: E-Motion Paralelo

Aqui são exibidos os testes realizados com a implementação paralela do E-Motion desenvolvida no decorrer do trabalho. Como a aplicação consome em média 5GB de memória, não se varia o valor *memória* no cenários. A escalabilidade interna foi delimitada em seis, pois acima dessa escalabilidade a aplicação tem sua aceleração comprometida, não fazendo sentido se alocar mais processos para

execução. A aplicação faz a execução de 10-cross-fold-validation, por isso a escalabilidade externa é fixada em 10.

A Tabela 6.7 mostra as informações referentes a aplicação E-Motion (Caso de Teste II).

Tabela 6.7 – Dados de entrada para o Caso de Teste II.

Escalabilidade externa	[10]
Escalabilidade interna	[1 a 6]
Compartilhamento de memória	Memória compartilhada (threads)
Tempo da aplicação sequencial	7500 minutos
Custo da aplicação sequencial	$15000,00

A Figura 6.2 mostra os resultados dos experimentos referentes ao Caso de Teste II.

A Tabela 6.8 exibe o dimensionamento criado automaticamente em cada cenário no Caso de Teste II.

Tabela 6.8 – Dimensionamento de jobs e tasks para o Caso de Teste II.

Dimensionamentos / *Cenário*	**Quantidade de jobs**	**Quantidade de tasks**
A:5K	3	10
B:5K	2	10
C:5K	1	10

Caso de Teste III: K-Means Paralelo (OpenMP)

Para a avaliação com este caso de teste se utilizou como referência a aplicação k-Means testada no Capítulo 3 implementada por meio de threads OpenMP. Como essa execução não varia o uso de memória, utiliza-se esse valor sempre fixo. A aplicação não realiza *cross-fold*, por isso usa-se somente uma instância de execução, ou seja, escalonamento externo sempre igual a um.

A Tabela 6.9 mostra as informações referentes a aplicação do Caso de Teste III).

Tabela 6.9 – Dados de entrada para o Caso de Teste III.

Escalabilidade externa	[1]
Escalabilidade interna	[1 a 6]
Compartilhamento de memória	Memória compartilhada (threads)
Tempo da aplicação sequencial	180 minutos
Custo da aplicação sequencial	$360,00

A Figura 6.3 mostra os resultados dos experimentos referentes ao Caso de Teste III.

A Tabela 6.10 exibe o dimensionamento criado automaticamente em cada cenário no Caso de Teste III.

Tabela 6.10 – Dimensionamento de jobs e tasks para o Caso de Teste III.

Dimensionamentos / *Cenário*	**Quantidade de jobs**	**Quantidade de tasks**
A:200MB	1	1
B:200MB	1	1
C:200MB	1	1

Caso de Teste IV: Aplicação de Troca de Mensagens

Neste caso de teste, se avalia o desempenho de uma aplicação fictícia que utiliza troca de mensagens em um ambiente de cluster, podendo ser executada em memória compartilhada. Assim sendo, o algoritmo de mapeamento irá ter a visão dos recursos cadastrados como clusters, permitindo que uma task tenha processos alocados em mais de um nodo computacional. Essa aplicação utiliza somente uma instância de execução. Utilizou-se uma escalabilidade interna alta propositalmente para avaliar a capacidade de utilização de mais de um nodo para alocar processos de uma mesma task.

A Tabela 6.11 mostra as informações referentes a aplicação do Caso de Teste IV.

Tabela 6.11 – Dados de entrada para o Caso de Teste IV.

Escalabilidade externa	[1]
Escalabilidade interna	[1 a 20]
Compartilhamento de memória	Memória distribuída (MPI)
Tempo da aplicação sequencial	10000 minutos
Custo da aplicação sequencial	$20000,00

A Figura 6.4 mostra os resultados dos experimentos referentes ao Caso de Teste IV.

A Tabela 6.12 exibe o dimensionamento criado automaticamente em cada cenário no Caso de Teste IV.

Tabela 6.12 – Dimensionamento de jobs e tasks para o Caso de Teste IV.

Dimensionamentos / *Cenário*	**Quantidade de jobs**	**Quantidade de tasks**
A:1K	1	1
A:4K	1	1
B:1K	1	1
B:4K	1	1
C:1K	1	1
A:4K	1	1

Caso de Teste V: Aplicação de Troca de Mensagens com Replicação de Execução

Este caso é idêntico ao Caso de Teste IV exceto pela diferença que este utiliza replicação de 10 execuções por meio de *cross-fold-validation*.

Tabela 6.13 – Dados de entrada para o Caso de Teste V.

Escalabilidade externa	[10]
Escalabilidade interna	[1 a 20]
Compartilhamento de memória	Memória distribuída (MPI)
Tempo da aplicação sequencial	10000 minutos
Custo da aplicação sequencial	$20000,00

A Tabela 6.13 mostra as informações referentes a aplicação do Caso de Teste V.

A Figura 6.5 mostra os resultados dos experimentos referentes ao Caso de Teste V.

A Tabela 6.14 exibe o dimensionamento criado automaticamente em cada cenário no Caso de Teste V.

Tabela 6.14 – Dimensionamento de jobs e tasks para o Caso de Teste V.

Dimensionamentos / *Cenário*	**Quantidade de jobs**	**Quantidade de tasks**
A:1K	1	10
A:4K	3	10
B:1K	1	10
B:4K	2	10
C:1K	1	10
A:4K	1	10

6.2.3 Comparação de Execução Real com Simulação

Para se obter uma medida de confiabilidade nos resultados que foram obtidos através de simulação, executou-se efetivamente dois cenários dos casos de testes do E-Motion Paralelo e do K-Means OpenMP no grupo de recursos A. Nessas execuções se utilizou os mesmos mapeamentos utilizados para a simulação dos resultados. Dessa maneira, se compara os resultados obtidos com a execução real com os resultados encontrados nas simulações realizadas.

A Figura 6.6 mostra os resultados da aceleração do *makespan* da execução das aplicações obtidos através de simulação e através da execução real. Em ambos os casos se utilizou a técnica de mapeamento Worst-Fit. Esses valores estão multiplicados 100 vezes para que se possa analisar o contraste da diferença entre ambos com maior clareza na figura.

Conforme pode ser observado, a diferença entre um resultado e outro é relativamente pequena, tendo-se um valor de aceleração um pouco menor para a execução real. Isso provavelmente se deve ao *overhead* que a execução de diversas instâncias quando executadas num mesmo nodo pode acarretar. Assim sendo, devido a essa baixa diferença, considera-se os resultados obtidos por meio de simulação suficientemente realísticos para serem utilizados para avaliação do algoritmo de mapeamento.

Apresentados os resultados para avaliação do algoritmo de mapeamento, a seção a seguir apresenta os experimentos relativos à avaliação da arquitetura proposta.

6.3 Experimentos para Avaliação da Arquitetura

Nesta seção são discutidos e apresentados os experimentos realizados a fim de se avaliar as funcionalidades das ferramentas da arquitetura. Inicia-se com uma breve descrição do protótipo implementado para a execução dos experimentos.

6.3.1 Descrição do Protótipo Implementado

Para a execução dos experimentos para avaliação da arquitetura se construiu um protótipo funcional com o intuito de realizar a execução de forma automatizada nos recursos da arquitetura do E-Motion Paralelo e do K-Means OpenMP. Para tal, implementou-se o gerenciador de execuções e o gerenciador de tarefas com proxy de escalonamento à *marfim* e ao *script* de execução remota. Do ponto de vista da aplicação implementou-se as rotinas de pré e pós-processamento e os algoritmos em si. Além disso, criou-se um modelo de aplicação que visa atender à integração das aplicações ao protótipo. Nessa prototipação, assume-se que todos os recursos, bases de dados e aplicações já estão devidamente configurados e prontos para uso. Entende-se que a demonstração desse tipo de configuração é uma questão meramente técnica e que não cabe ao escopo do propósito de avaliação discutir esses arquivos de configuração.

Segue a descrição de cada módulo implementado:

- **Gerenciador de Execuções:** Implementou-se um gerenciador de execuções funcional contendo implementações do algoritmo de mapeamento e prospector de recursos, ambos interligados com as aplicações e os recursos disponíveis.

  O algoritmo de mapeamento tem sua entrada equivalente a exposta na Seção 5.3 e tem essa alimentada pela descrição dos recursos e da aplicação previamente configurados, o que inclui a escalabilidade da aplicação. Sua saída é utilizada pelo gerenciador de execuções para a criação dos jobs e tasks a serem submetidos ao gerenciador de tarefas.

  O prospector de recursos foi implementado como uma aplicação que quando executada em um nodo, faz um diagnóstico através da execução de comandos presentes no Unix para listagem de configuração de hardware. Também faz a execução da aplicação de *benchmark* Java Scimark 2 [31], que objetiva obter um perfil da arquitetura de processamento da máquina. Ainda, executa uma série de operações comuns em aplicações científicas e numéricas fazendo uma composição desses resultados, apresentando essa composição como um valor em MFlops (milhões de operações em ponto flutuante por segundo). Esse valor é usado como referência para o desempenho computacional de um núcleo do processador da máquina e é valor utilizado pelo mapeador da arquitetura para mensurar a capacidade de uma máquina. A escolha dessa aplicação de benchmark se deu pelo fato de essa ser implementada em Java o que possibilita que execute em arquitetura de processadores diferentes e por apresentar resultados

que coincidem com o desempenho alegado pelos fabricantes dos processadores dos recursos testados.

- **Gerenciador de Tarefas**: A prototipação do gerenciador de tarefas consiste na implementação de rotinas básicas para a execução de comandos remotos nos nodos computacionais. Utilizou-se somente um proxy para acesso a máquina marfim para que se pudesse indiretamente acessar os demais nodos dos clusters e também para que fosse possível executar aplicações de troca de mensagens. Também se implementou um mecanismo de geração do script de execução da aplicação remota, que garante a transferência dos arquivos demandados pela aplicação, gera a estrutura de diretório para a execução remota, adquire os dados, executa a aplicação e, por fim, grava os resultados na área de armazenamento. A fim de simplificar a implementação, assume-se que a área de armazenamento é um ponto de montagem a um compartilhamento de rede via NFS (*Network File System*).

- **Modelo de Aplicação:** O modelo de aplicação criado para comportar a execução das aplicações suportadas pela arquitetura consiste em um conjunto de três diretórios, cada um para armazenar os scripts de pré e pós processamento e para o armazenamento da aplicação em si. Em cada um desses diretórios se criou um adaptador de chamada de comando que, ao receber os parâmetros básicos para a execução de cada função correspondente ao diretório, cria uma linha de comando de execução real, adaptando os parâmetros básicos passados pelo gerenciador de execuções para a entrada requisitada pela implementação da função de cada aplicação. Esses adaptadores utilizam das informações da descrição da aplicação para requisitar, se necessário, que o usuário preencha as entradas para os parâmetros de execução da aplicação, repassando esses de forma adequada para a linha de execução da aplicação.

  Definiu-se que os parâmetros básicos para chamada da execução da rotina de pré-processamento devem ser a base que se deseja particionar, o número de partições e o padrão de nome que será dado às partições resultantes do processo (e.g. part-emotion-1, part-emotion-2...). Para a chamada da execução da aplicação, considera-se como padrão os parâmetros que indicam o número de processos internos, qual(is) a(s) partição(ões) de dados e qual o diretório onde os resultados serão armazenados. Por fim, os parâmetros padrão da rotina de pós-processamento ficaram definidos como sendo as partições que devem ser consolidadas e o arquivo de saída contendo os resultados.

Descrito o protótipo utilizado para os experimentos, as próximas seções apresentam os resultados dos experimentos realizados a fim de se avaliar o desempenho da solução proposta, iniciando-se pela avaliação do gerenciador de dados.

6.3.2 Impacto do Gerenciador de Dados

O objetivo dos experimentos desta seção é avaliar qual o desempenho que o uso de um gerenciador de dados com as funções descritas na arquitetura pode agregar em termos de desempenho na execução da mineração no que diz respeito aos passos em que ocorrem aquisição, preparação e distribuição de dados.

Segue a descrição dos experimentos realizados para avaliação da viabilidade do gerenciador de dados.

Experimento de Aquisição e Conversão de Dados

O experimento realizado buscou obter o tempo que é necessário para transferir os dados conseguidos através de uma consulta SQL a uma base de dados e convertê-los para um arquivo de dados no formato ARFF, o mesmo utilizado pelo E-Motion e por uma série de aplicações de mineração encontradas na literatura. Para aquisição dos dados, utilizou-se a API padrão de acesso a SGBD da plataforma Java (JDBC). Para se realizar a conversão dos dados adquiridos, usou-se a API do *framework* Weka para conversão de consultas SQL para arquivos do tipo ARFF.

Utilizou-se como software SGBD a versão 8.3.9-1 do Postgres instalado em uma máquina equivalente à Atlântica e acessado de outra máquina idêntica através de uma conexão Fast-Ethernet. Utilizou-se a configuração padrão do SGBD, somente se alterando as permissões de acesso remoto de dados. O banco de dados criado consiste em dados sintéticos armazenados numa única tabela com 32 colunas do tipo inteiro na qual se variou a quantidade de entradas a fim de se obter consultas de 10, 100, 500 e 1000 megabytes (MB). Os dados replicados na tabela foram obtidos do repositório de dados de testes da Universidade de Sevilha [2]. A consulta SQL utilizada corresponde a uma consulta onde se seleciona todo o conteúdo da tabela que contém os dados (*select * from tabela*).

A Figura 6.7 mostra o tempo de execução dos experimentos de acordo com o tamanho da base utilizada.

Experimentos de Particionamento de Dados

Neste experimento se procurou obter o tempo necessário para realizar o particionamento das bases convertidas no experimento anterior utilizando a ferramenta disponibilizada pela API do Weka para se executar procedimentos de *cross-fold-validation* para geração de 10 *folds* em uma base no formato ARFF.

A Figura 6.8 mostra os tempos que cada experimento necessitou para a realização de *10-cross-fold-validation* de acordo com o tamanho da base ARFF.

Experimentos de Distribuição de Dados

Este experimento teve como objetivo verificar o tempo que é necessário para a distribuição das partições de dados aos nodos computacionais para a mineração ser realizada de acordo com o tipo de transferência que é realizada (cópia remota, acesso local e transferência Web). Os recursos e metodologias utilizados foram os mesmos usados no experimento de aquisição de dados, com excessão que neste caso os dados estão armazenados em um arquivo e não por meio de um SGBD e se utilizou a implementação de um Web Service em Java hospedado em um servidor Tomcat 5.

A Figura 6.9 mostra o gráfico com os tempos obtidos para transferências das partições com tamanhos exibidos no eixo x.

6.3.3 Impacto e Overhead da Arquitetura na Execução do Processo de Mineração

Tendo-se os valores para cada etapa dos processos de manipulação de dados e os tempos de execução da mineração efetiva, parte-se para a avaliação do impacto que as ferramentas da arquitetura podem trazer à execução de todo o processo de mineração.

Esta seção mostra o ganho que uma execução de mineração pode obter com a sua execução através das ferramentas da arquitetura. Para tal, faz-se uma comparação do tempo que as execuções de todo o processo de mineração das aplicações E-Motion paralela e da implementação do algoritmo K-Means testada (Seção 3.1.1) levam para serem executadas em diferente cenários. Compara-se as execuções sendo realizadas de três modos: executadas sequencialmente, executadas com a arquitetura sem os mecanismos do gerenciador de dados e executadas com a arquitetura utilizando as ferramentas do gerenciador de dados. Também se apresenta o *overhead* que a utilização da arquitetura implica.

A Figura 6.10 mostra o tempo total de execução das aplicações nos cenários mencionados. Esse tempo é composto pela execução de todos as etapas do processo e o *overhead* devido ao uso da arquitetura. Salienta-se que o tempo de aquisição de dados inclui o tempo de conversão dos dados para o formato ARFF.

A política de mapeamento utilizada foi a *Worst Fit* por ter o melhor resultado entre as políticas testadas. Para a execução da versão sequencial se utilizou uma máquina equivalente à Atlântica. Analisando-se os resultados, pode-se perceber que a utilização da arquitetura se torna viável a partir do momento em que o tempo de execução da mineração de dados é relativamente extenso em relação ao tempo do *overhead* necessitado pela arquitetura. Quanto às ferramentas de gerência de dados, pode-se dizer que essas se tornam úteis quando conseguem reutilizar particionamentos e conversões de dados de execuções anteriores em cenários os quais se utiliza bases de dados de no mínimo 100MB.

Analisando as aplicações testadas se verifica que no caso do E-Motion o maior impacto na aceleração da execução foi atingido por meio do mapeamento utilizando o algoritmo de mapeamento proposto. Como a aplicação utiliza bases de dados pequenas o uso das ferramentas de gerência de

dados se torna pouco relevante para a aceleração de todo o processo.

Para o caso da implementação do K-Means, observa-se que o que causa maior impacto no tempo do processo de execução são as ferramentas de gerência de dados. Isso ocorre pois, no exemplo testado, utilizou-se uma base de dados relativamente grande (500MB) a qual demanda tempo considerável para ser preparada para a aplicação. Quanto ao ganho com o uso das ferramentas de mapeamento automático, pelo fato de a aceleração da versão paralela da aplicação ser baixa, o *overhead* acarretado pelo uso dessas ferramentas acaba por anular o ganho de desempenho obtido com a paralelização.

Análise do *Overhead* da Arquitetura

O *Overhead* da arquitetura nos casos executados foi de aproximadamente 65 segundos. Desse tempo total, constatou-se que 45 segundos são referentes à fase de prospecção de recursos, a qual demanda mais tempo em virtude da execução do *benchmark* nos nodos computacionais. Os tempo restante (20 segundos) é demandado pelo mapeador de tarefas e para a criação das tarefas concretas para execução e envio dos arquivos de execução aos nodos computacionais.

Terminada a apresentação dos resultados obtidos nos experimentos, segue, na próxima seção, uma discussão sobre a avaliação da arquitetura.

6.4 Discussão sobre os Resultados Obtidos

Nesta seção se apresenta uma discussão dos resultados obtidos nos experimentos. Avalia-se esses resultados procurando fazer relação com o atingimento dos objetivos propostos na solução apresentada.

Inicia-se essa discussão avaliando-se o algoritmo de mapeamento proposto. Segue os pontos da discussão:

a) **Corretude dos mapeamentos:** o primeiro ponto a ser analisado diz respeito à corretude do mapeamento realizado pelo algoritmo. Avaliando-se se não houve mal dimensionamento de tasks e/ou alocações de um único processador para mais de uma task e vice-versa.

   Nesse ponto, se pode afirmar que o algoritmo executou corretamente a alocação de tasks e processos. Não foi encontrado em nenhum dos casos de teste evidências de erro de alocação, nem houve caso onde foram elegidos recursos que não suportavam a a aplicação a ser executada. Sob o ponto de vista do compartilamento de memória, também se respeitou a divisão da memória do recurso de forma que uma task não compartilhasse a memória alocada com outra. Analisando a funcionalidade de dimensionamento automático da escalabilidade das aplicações, pôde-se constatar que em todos os casos testados o dimensionamento sempre conseguiu atingir a maior dimensão que os recursos a serem escalonados poderiam suportar. Além disso, também se observou que o balanceamento de carga entre os jobs sempre resultou no mais balanceado

possível e não houve casos onde foram criados jobs desnecessários, podendo-se afirmar que o algoritmo sempre utilizou as possibilidades de escalonamento de maneira a minimizar o número de jobs para execução, sempre procurando diminuir o tempo de execução das aplicações.

Ainda sobre a escolha dos recursos, também se observou que a solução soube chavear de forma correta a maneira de como os recursos poderiam ser vistos: como clusters ou nodos isolados. Isso permitiu um melhor aproveitamento de recursos em aplicações de memória distribuída, permitindo que uma única task possa utilizar unidades de processamento de mais de um nodo, aumentando seu *makespan*.

b) **Comparação das Técnicas de Mapeamento Utilizadas:** as heurísticas utilizadas para alocação de tasks se mostraram eficientes para diminuir o makespan das execuções, pois no pior dos casos tiveram resultado igual a alocação aleatória e sempre obtiveram a aceleração do makespan quando comparadas à versão sequencial da aplicação. Os resultados em que houve empate de desempenho das heurísticas com a técnica aleatória ocorreram em conjuntos pequenos de recursos, onde as possibilidades de alocação são bastante reduzidas, diminuindo o campo de exploração de possibilidades das heurísticas.

   Outro ponto a ser discutido são as modificações realizadas nas heurísticas para se adequarem ao cenário da arquitetura, que possui tanto recursos quanto aplicações heterogêneas. De acordo com os resultados obtidos, pode-se afirmar que essas modificações obtiveram boa resposta, fazendo com que as características funcionais de cada técnica não fossem drasticamente alteradas e conseguissem obter resultados compatíveis com suas características originais.

   Para complementar essa etapa da discussão, segue uma breve descrição dos resultados obtidos com cada técnica.

   - *First-Fit:* foi a técnica que, na média, obteve a menor aceleração em relação a alocação randômica. Porém, em contrapartida, foi a técnica que melhor minimizou o uso de recursos, o que influi diretamente na diminuição dos custos de execução de uma aplicação. Portanto, deve ser utilizada quando se pretende minimizar a quantidade de recursos utilizados e o custo da alocação.
   - *Worst-Fit:* esta técnica foi a que no geral apresentou a melhor resultado em termos de aceleração do makespan das aplicações. Seus resultados se tornam mais evidentes em cenários onde há uma grande quantidade de recursos. Por a heurística sempre buscar o recurso com maior capacidade disponível, aumenta-se o número de nodos utilizados, consequentemente possibilitando o aumento da quantidade de processos que cada task pode ter. Em contrapartida, essa abordagem, por maximizar a utilização dos recursos, pode acarretar em uma elevação do custo de execução de uma aplicação, principalmente se essa não tiver uma aceleração interna com boa eficiência. Considera-se recomendável o uso dessa

heurísticas em casos onde se quer priorizar a aceleração da aplicação independentemente do custo de execução que se terá.

- *Best-Fit:* é, entre as técnicas testadas, foi aquela que obteve a melhor relação entre custo de execução e a aceleração da execução. Isso ocorreu graças a modificação realizada a fim de se evitar o uso de recursos com desempenho relativamente baixo, o que fez com que se aumentasse os resultados de aceleração sem, no entanto, comprometer o custo da execução.

c) **Aprimoramentos possíveis:** observando-se os resultados, foi constatado que possíveis aprimoramentos na maneira de escalonamento do algoritmo podem melhorar o desempenho do mesmo. A implementação de uma técnica de escalonamento dinâmico é a principal modificação em se tratando de agregar desempenho ao processo. Essa técnica pode fazer com que à medida que uma task de um job terminar possa-se dinamicamente alocar uma task de outro job, e, até mesmo, pode-se recalcular as alocações de task a recursos quando um recurso antes ocupado vagar. Acredita-se que isso possa minimizar o tempo ocioso de utilização de recursos que pode ocorrer quando se espera todas as tasks de um job terminar para se iniciar a execução de outro job.

Para terminar a avaliação da solução apresentada no trabalho, discute-se alguns pontos sobre os resultados de avaliação da arquitetura:

a) *Suporte a recursos heterogêneos:* nesse ponto, pode-se dizer que o trabalho proposto atingiu o seu objetivo, pois mostrou que os resultados das execuções realizadas com ajuda do mapeador de recursos tiveram um resultado considerável em relação ao escalonamento aleatório. Também se considera que os métodos de prospecção de recursos utilizados apresentaram, na maioria dos casos, resultados compatíveis com o esperado. Possibilitando que o mapeador de recursos trabalhe adequadamente com recursos os quais os usuários não tem informações completas.

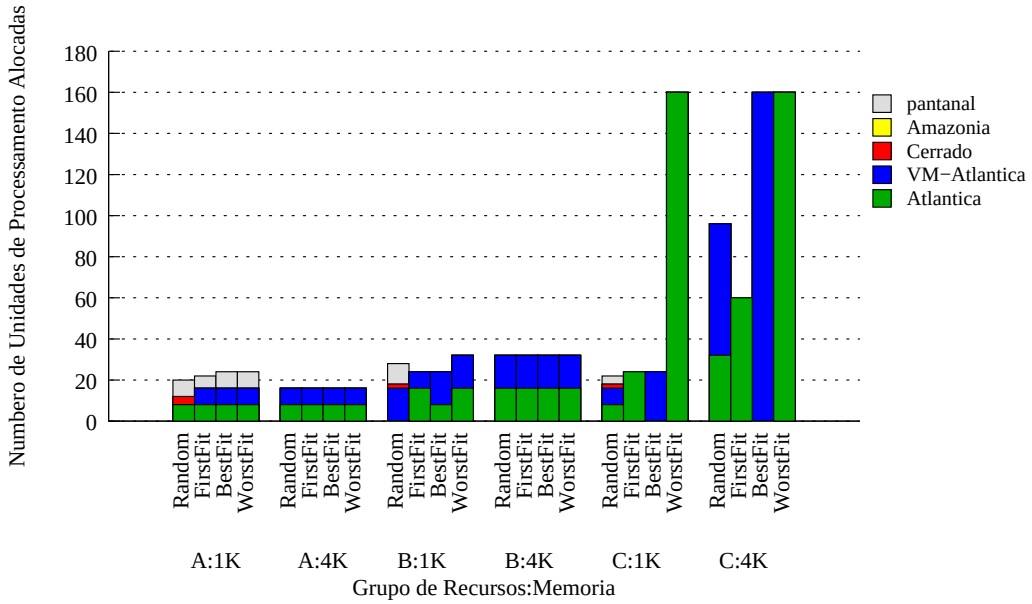
b) *Suporte a diferentes classes de aplicações:* considera-se que arquitetura apresenta soluções que permitem a execução de uma gama de aplicações de mineração que se enquadram na taxonomia de aplicações suportada. Por outro lado, o suporte de aplicações com diferentes formas de implementação ou modos de utilização se mostrou ser uma tarefa bastante difícil. Embora se tenha conseguido adaptar duas aplicações completamente diferentes em termos de linguagem de programação e parâmetros de execução bastantes distintos (E-MOtion implementado em Java e k-Means implementado em C), sabe-se que há casos em que talvez a execução de certas implementações de algoritmos de mineração possa ser dificilmente realizada na arquitetura, sem que se tenha que modificar a forma de execução dessas aplicações.

c) *Ferramentas de gerência de dados:* sobre estas ferramentas, a conclusão que se pode chegar através dos experimentos é a de que a execução de tarefas de mineração pode ganhar um aumento

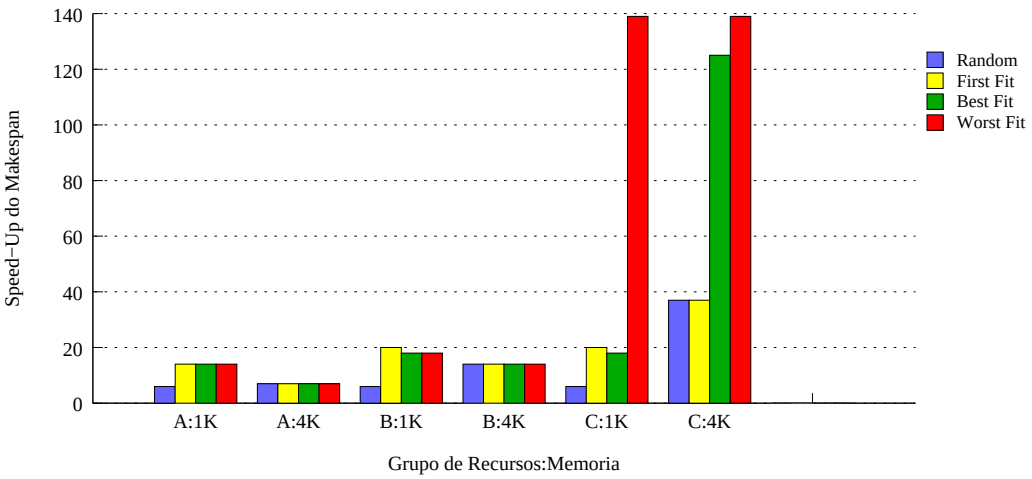
significativo em performance quando há ferramentas que auxiliem a manipulação de dados. Embora os testes realizados tenham sido executados em ferramentas não especializadas para mineração, fica claro que, mesmo soluções relativamente simples, se comparadas com soluções comerciais de grande porte, podem trazer benefícios para a organização e execução de tarefas de mineração.

d) *Facilidade de utilização:* entende-se que a implementação de uma interface de usuário adequada para interagir com os componentes da arquitetura pode possibilitar que a execução automática de aplicações de mineração de dados paralela possa ser realizada de forma intuitiva e direta, mesmo quando efetuadas por usuários não experientes em computação de alto desempenho. Já quando o assunto é o cadastramento de recursos e de aplicações, embora esses procedimentos possam ser triviais a especialistas em computação de alto desempenho, são procedimentos que podem apresentar dificuldades para serem realizados pelo usuário final.

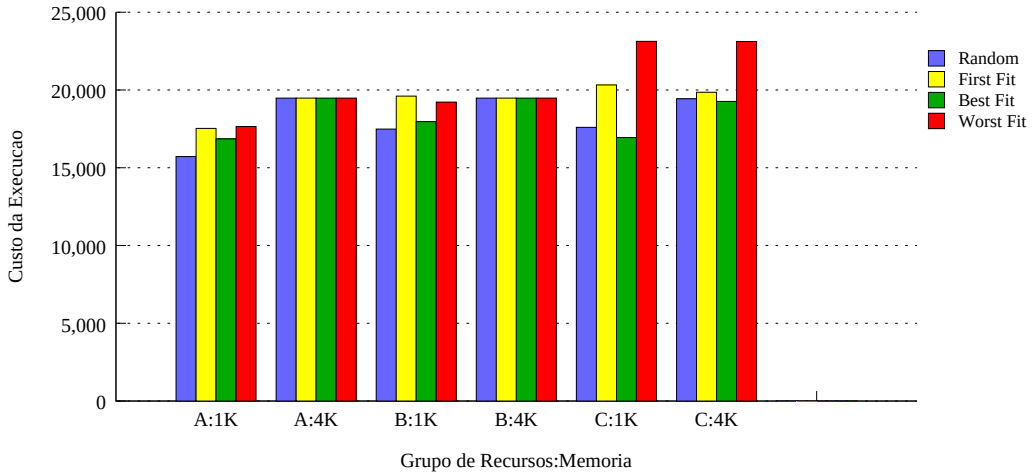
Dadas as considerações acima, dá-se por encerrado a avaliação do trabalho proposto. Segue para finalizar o trabalho, as considerações finais.



(a) Unidades de processamento alocadas

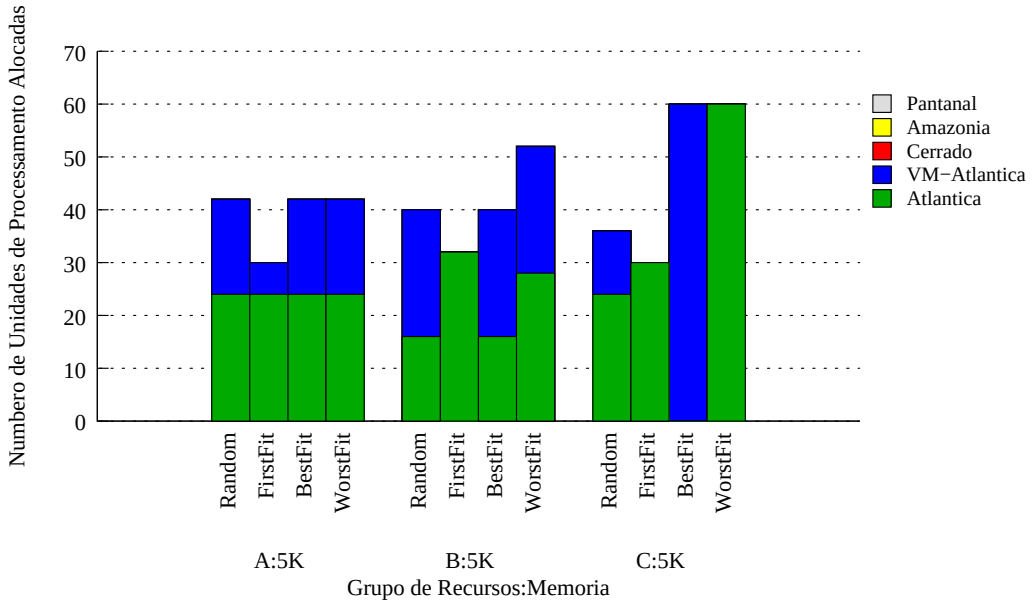


(b) Aceleração do makespan

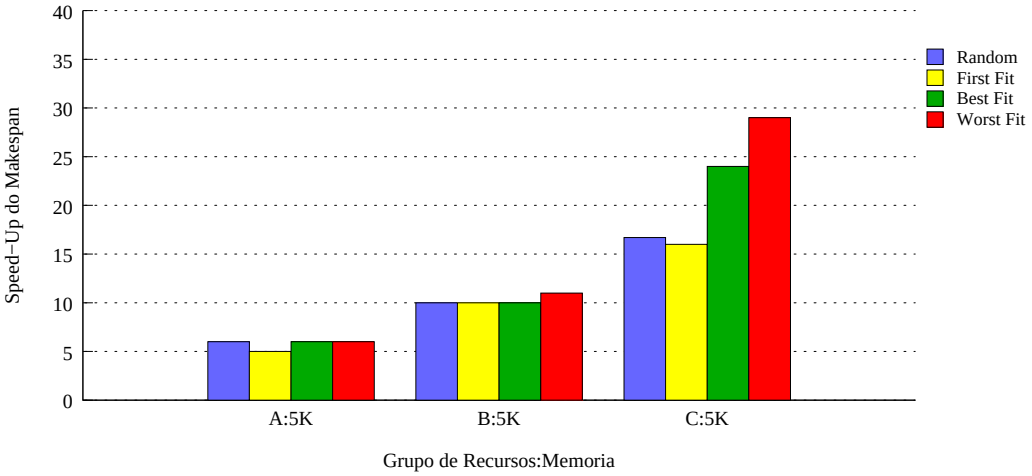


(c) Custo da alocação

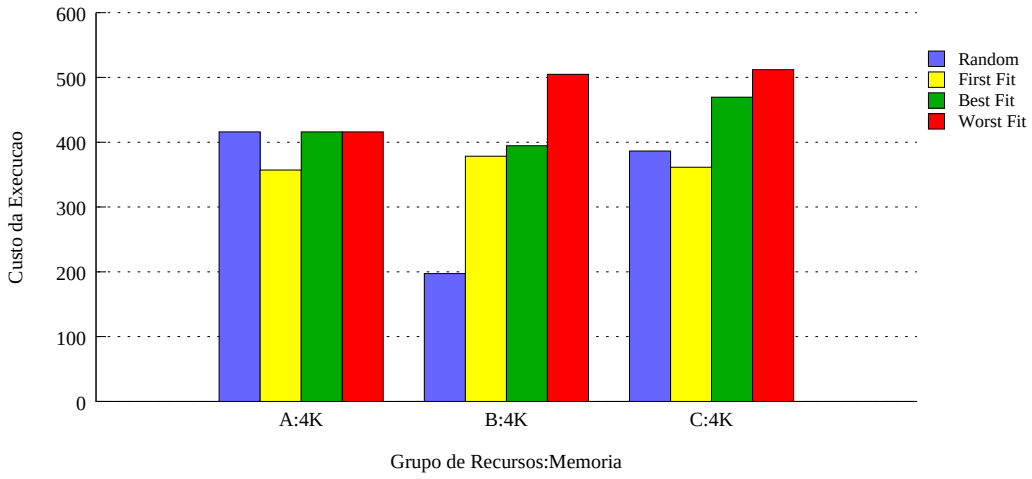
Figura 6.1 – Resultados obtidos para o Caso de Teste I.



(a) Unidades de processamento alocadas

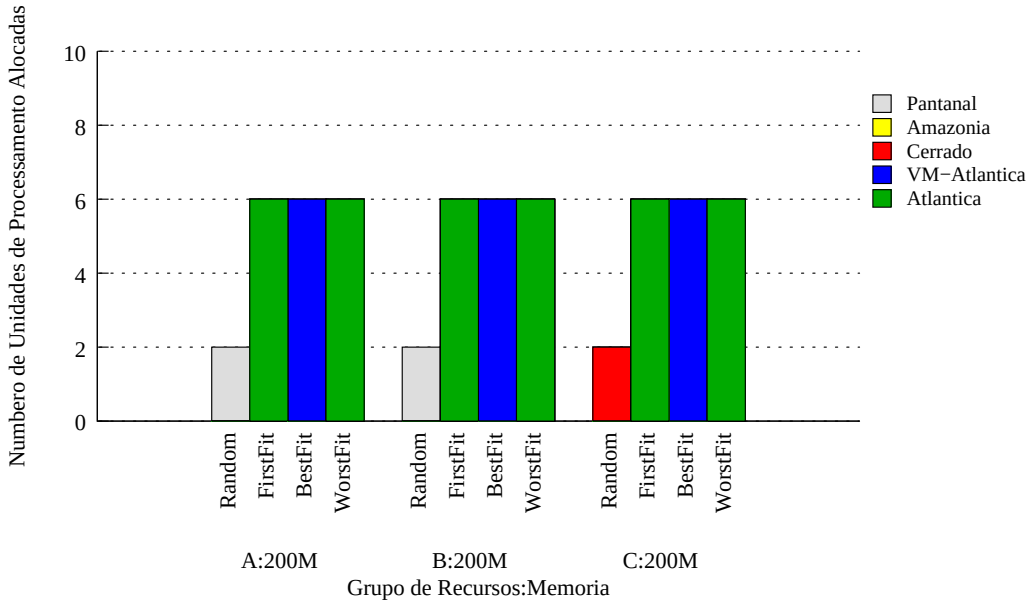


(b) Aceleração do makespan

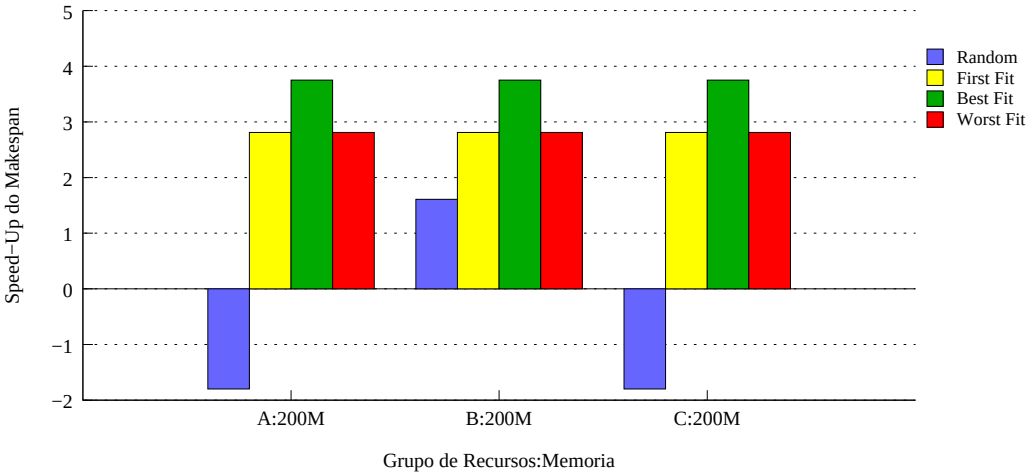


(c) Custo da alocação

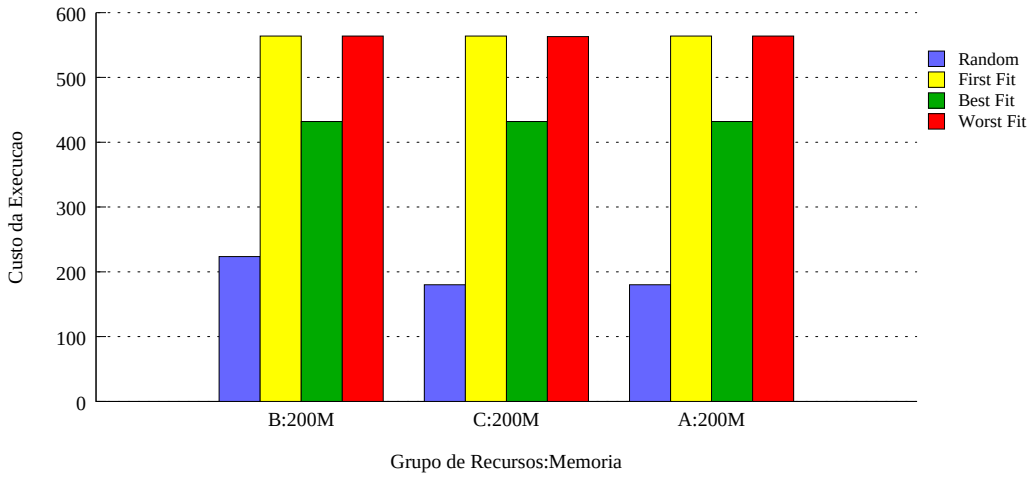
Figura 6.2 – Resultados obtidos para o Caso de Teste II.



(a) Unidades de processamento alocadas.

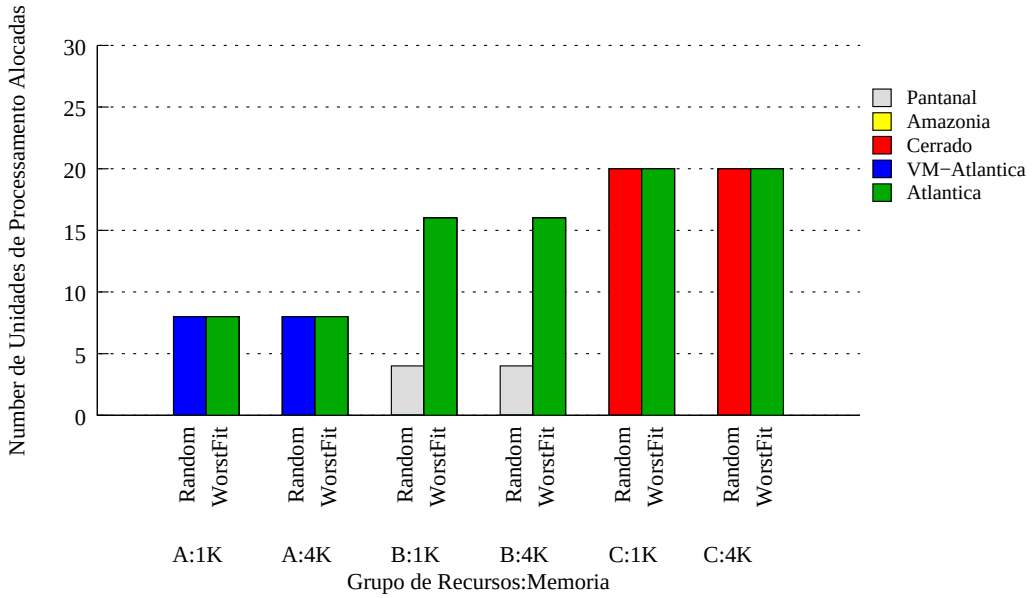


(b) Aceleração do makespan.

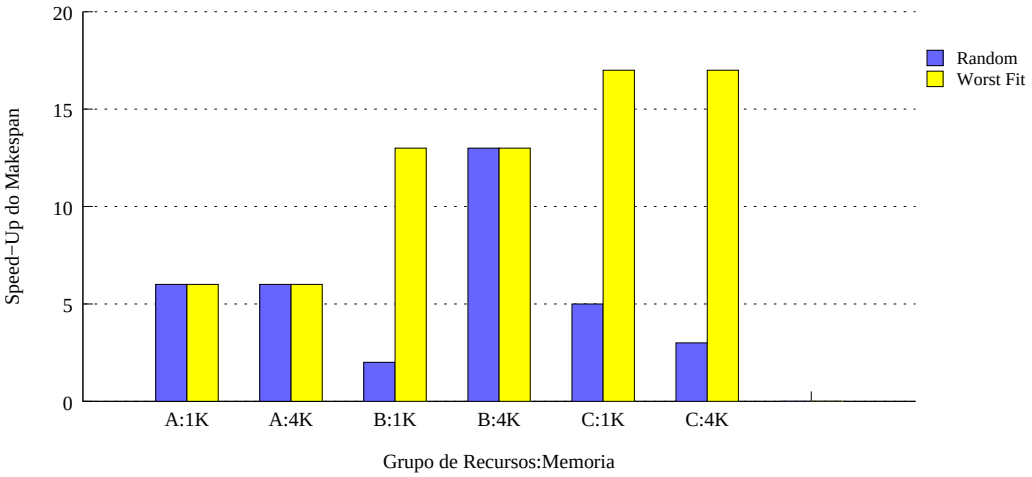


(c) Custo da alocação.

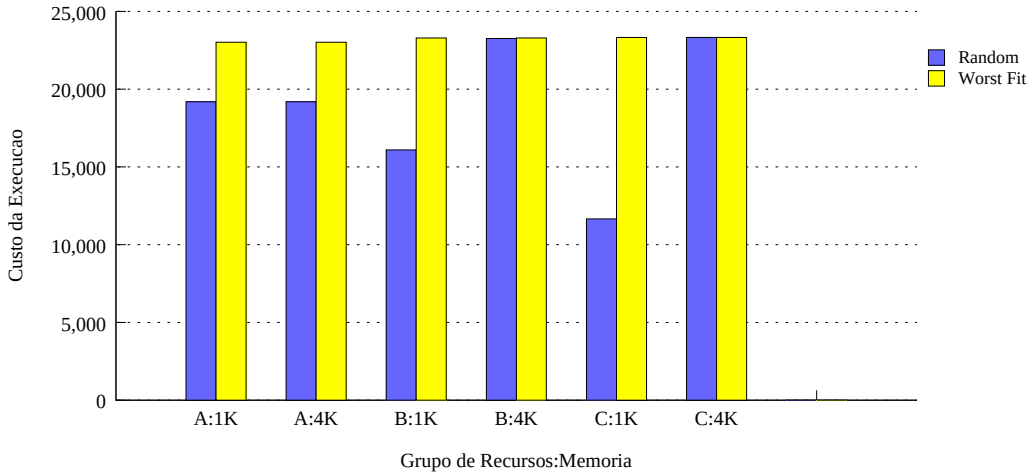
Figura 6.3 – Resultados obtidos para o Caso de Teste III.



(a) Unidades de processamento alocadas

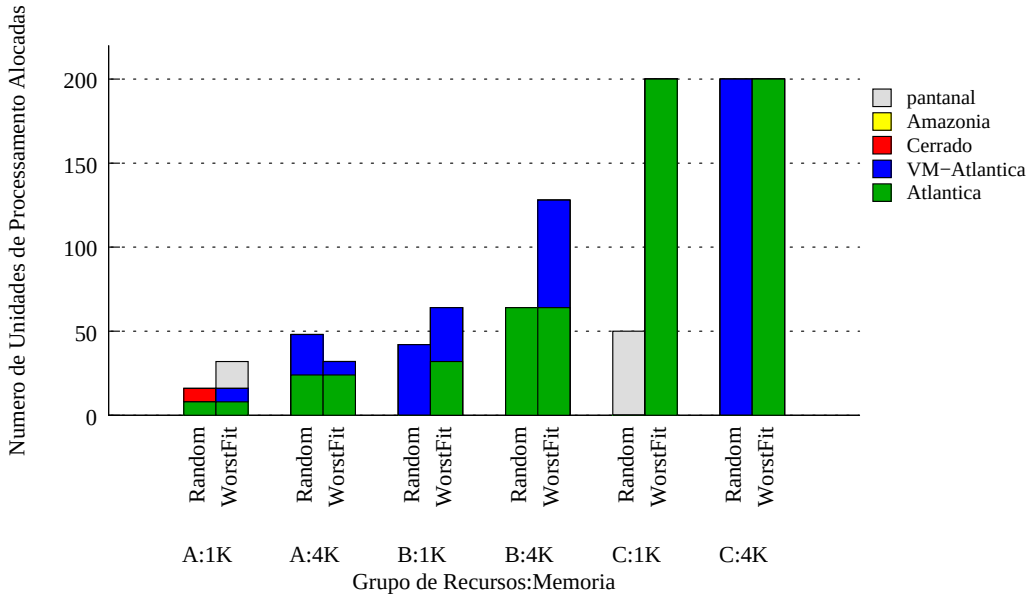


(b) Aceleração do makespan

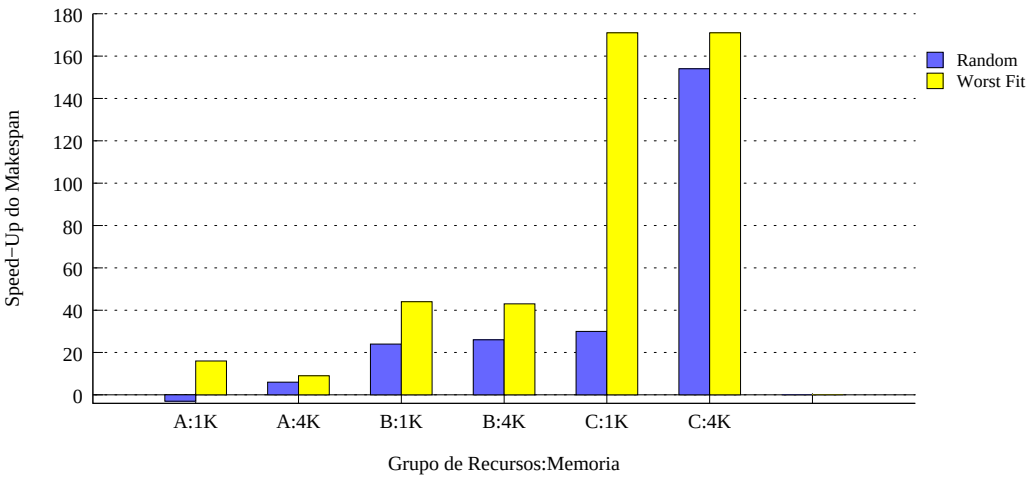


(c) Custo da alocação

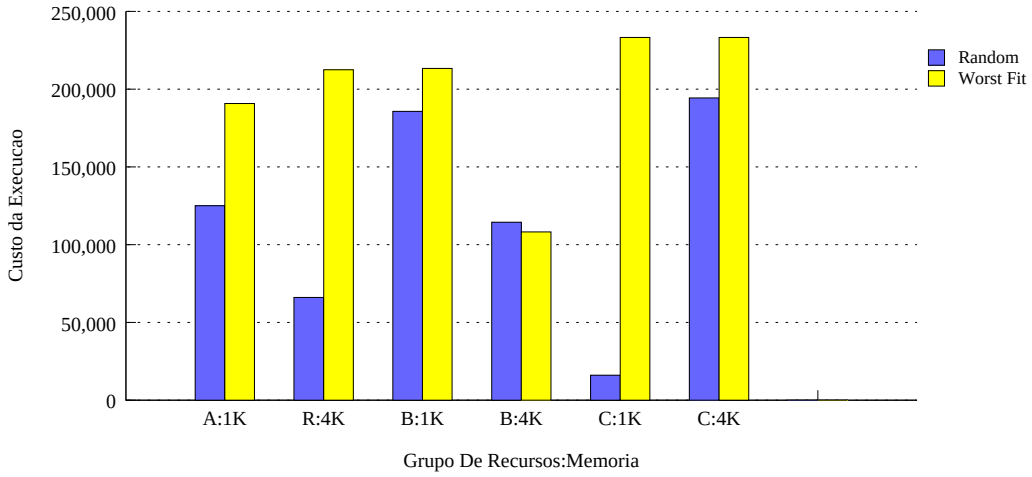
Figura 6.4 – Resultados obtidos para o Caso de Teste IV.



(a) Unidades de processamento alocadas



(b) Aceleração do makespan



(c) Custo da alocação

Figura 6.5 – Resultados obtidos para o Caso de Teste V.

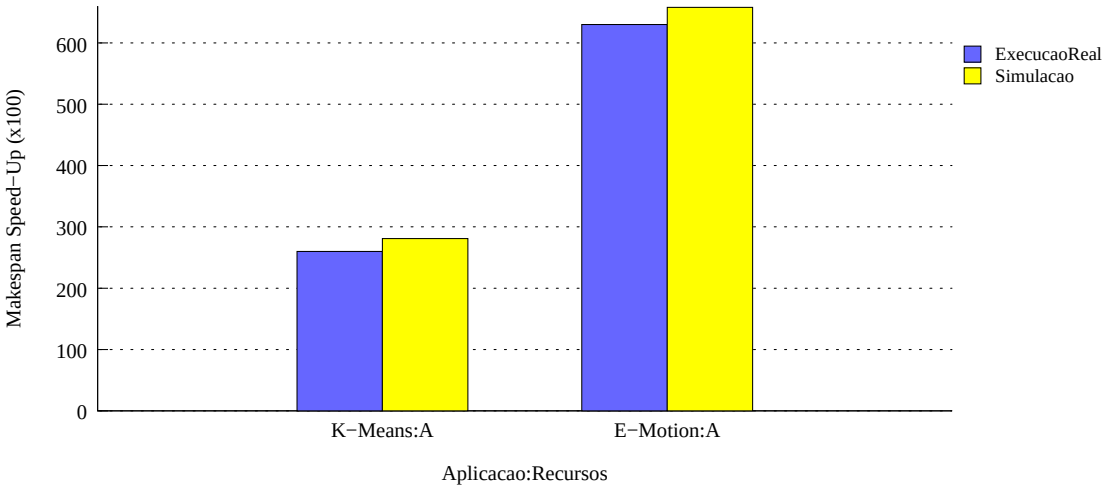


Figura 6.6 – Comparação do Speed-Up uma Execução Real com a Simulada.

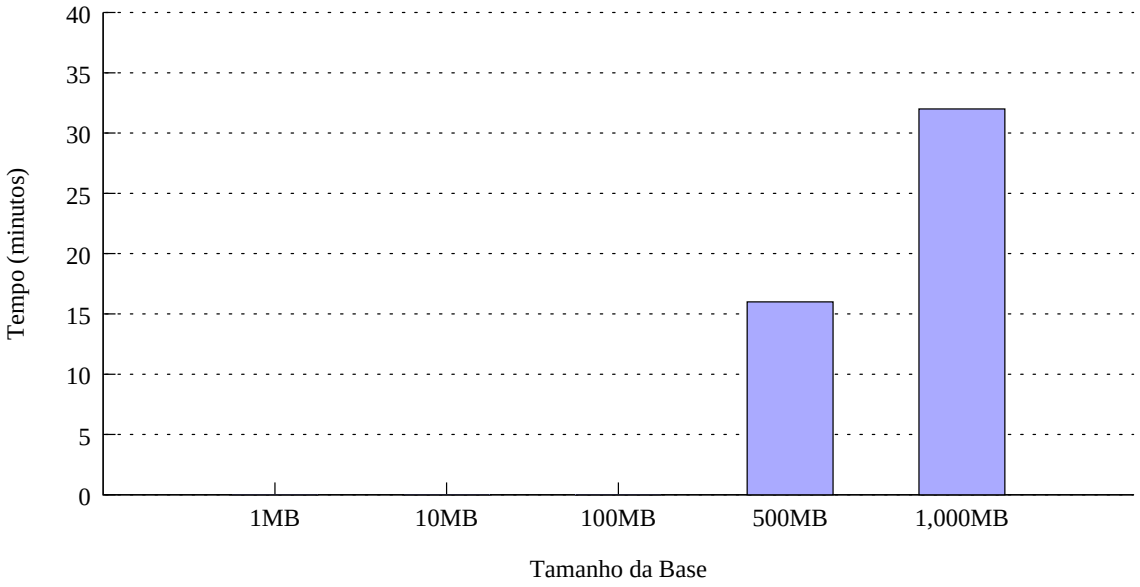


Figura 6.7 – Tempos de Aquisição de Dados do SGBD e Conversão para o formato ARFF.

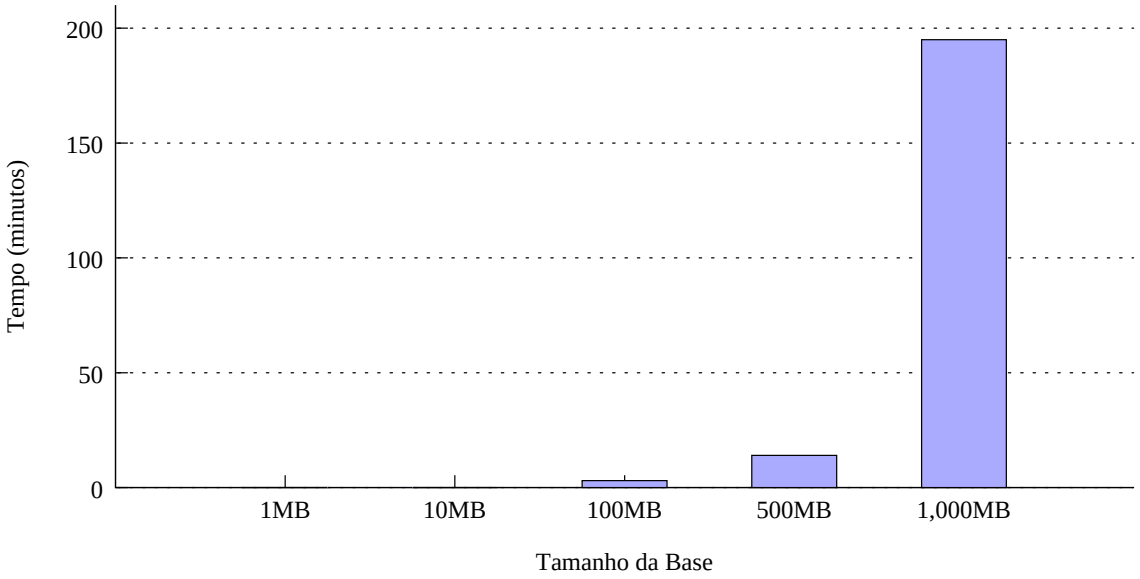


Figura 6.8 – Tempos para Particionamento com 10-Cross-Fold-Validation do Weka.

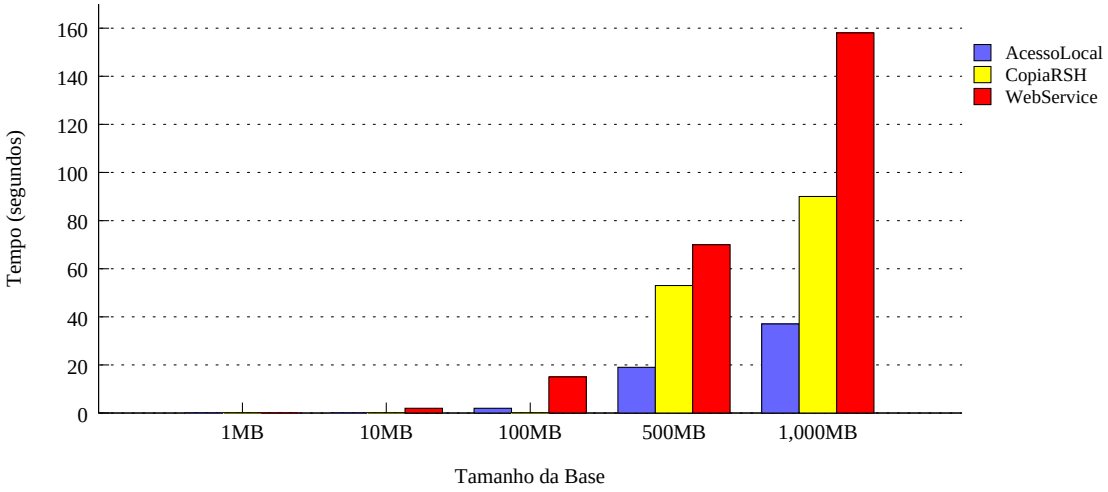
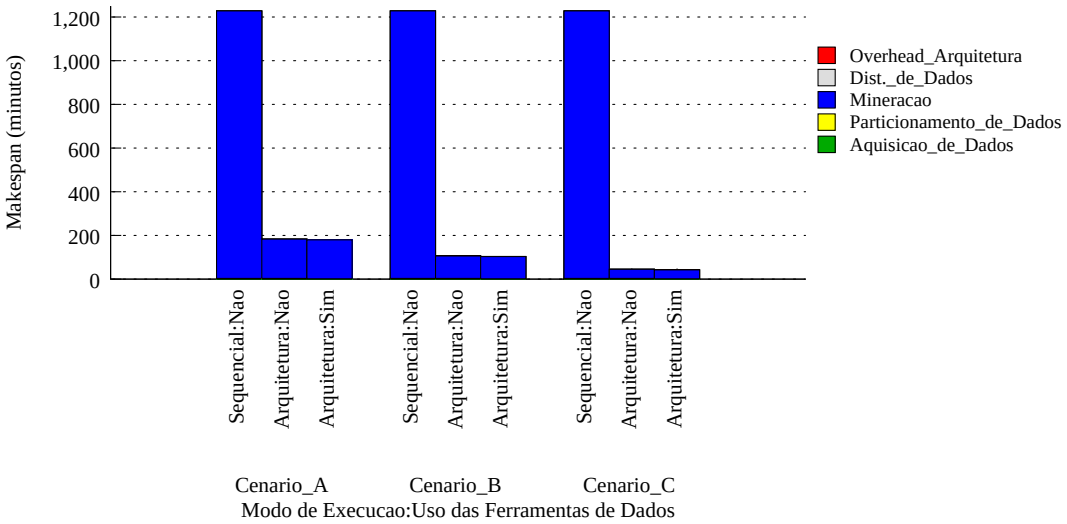
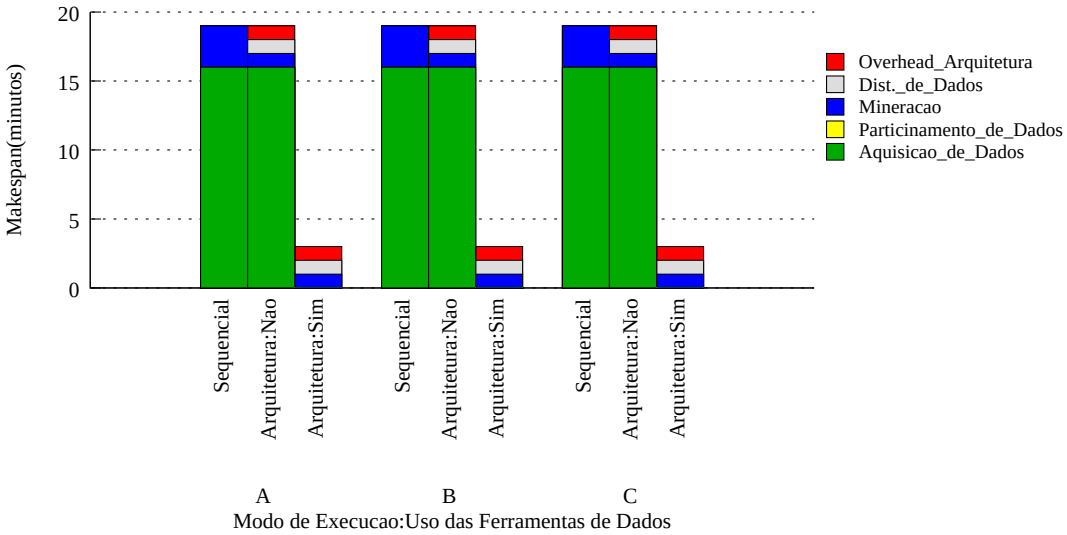


Figura 6.9 – Tempos para Transferência de Dados da Área de Armazenamento à Máquina Remota.



(a) E-Motion com Base de 1MB



(b) K-Means com Base de 500MB

Figura 6.10 – Impacto da Arquitetura e Overhead de Utilização das suas Ferramentas.

7. Considerações Finais

Este trabalho apresentou uma arquitetura para suportar o dimensionamento e execução de processos de mineração de dados paralelos e/ou distribuída em ambientes de computação de alto desempenho. As ferramentas propostas procuraram oferecer suporte a todas as etapas que compreendem a execução de um processo de mineração.

As principais contribuições deste trabalho foram:

- Criação de uma arquitetura que apoia a execução de aplicações de mineração em ambientes de alto desempenho. Destaca-se a possibilidade de execução de diversos tipos de aplicações de mineração, bastantes comuns no Estado da Arte.

- Apresentou-se uma abordagem para prospecção de recursos que permitiu que usuários pudessem utilizar recursos que tem acesso sem ter profundo conhecimento do funcionamento desses. Acrescenta-se o fato de que essas informações sobre os recursos puderam ser utilizadas para mapeamento de aplicações sem prejudicar a acurácia de mapeamento da maioria dos cenários testados.

- Paralelização de um algoritmo para a tarefa de regressão. A paralelização do E-Motion [1] possibilitou ganho de performance tanto com o paralelismo de dados quanto com o de instruções, que aliados mostraram ser possível uma aceleração do tempo de execução da aplicação de até 28 vezes, possibilitando que um problema que demorava dias para ser solucionado, possa agora ser resolvido na escala de horas.

- O ponto de maior destaque dessa arquitetura é a funcionalidade de dimensionamento e mapeamento automático de aplicações de mineração paralela em recursos heterogêneos, o que diferencia o solução apresentada dos demais trabalhos estudados (Tabela 7.1). O algoritmo proposto obteve sucesso em adaptar as heurísticas de *bin packing* para servirem ao propósito de mapear tasks heterogêneas em recursos computacionais igualmente heterogêneos. Possibilitando que se realizasse esse tipo de tarefa em um tempo computacional viável.

Como trabalhos futuros se considera as seguintes opções como sendo tópicos passíveis de investigação:

- Inclusão do custo de transferência de dados para o recurso remoto como um fator a ser levado em conta no mapeamento de aplicações.

- Desenvolvimentos de algoritmos de escalonamento que apliquem técnicas de escalonamento dinâmico; o que pode minimizar o tempo total de uma execução, diminuindo o tempo ocioso dos recursos.

Tabela 7.1 – Suporte às funcionalidades da arquitetura proposta nos trabalhos relacionados.

Funcionalidade / Ferramenta	**Weka4WS**	**Tamanduá**	**Anthill**	**Jin's Framework**
Recursos heterogêneos	*Sim*	*Não*	*Não*	*Não*
Execução Distribuída	*Sim*	*Sim*	*Sim*	*Sim*
Multiplataforma	*Não*	*Não*	*Não*	*Não*
Aplicações em diversas linguagens	*Não*	*Não*	*Não*	*Não*
Ferramentas de acesso a dados	*Sim*	*Sim*	*Não*	*Não*
Execução Paralela	*Não*	*Sim*	*Sim*	*Sim*

- Mineração de dados em ambientes de *Cloud Computing*, estudando-se a viabilidade da execução de aplicações de mineração nesses ambientes e quais os desafios a serem vencidos para possibilitar essa integração.

Com essas considerações, dá-se por encerrado o trabalho apresentado neste documento.

Referências Bibliográficas

[1] Barros, R.C. "Evolutionary Model Trees Induction". Dissertação de Mestrado, Programa de Pós Graduação em Ciência da Computação, PUCRS, 2009, 72p.

[2] Bioinformatics Group Sevilha. "Dataset Repository in ARFF (Weka)". Capturado em: http://www.upo.es/eps/bigs/datasets.html, Janeiro 2010.

[3] Bohn, C.A.; Lamont, G.B. "Load balancing for heterogeneous clusters of PCs". *Future Generation Computer Systems*, vol. 18-3, 2002, pp. 389-400.

[4] Brassard, G.; Bratley, P. "Fundamentals of Algorithmics", Prentice-Hall, Inc., 1996.

[5] Chawla, N.V; Hall, L. O.; Bowyer, K.W.; Kegelmeyer, W.P. "Learning Ensembles from Bites: A Scalable and Accurate Approach". *Journal of Machine Learning Research*, vol. 5, 2004, pp. 421-451.

[6] Chetty, M.; Buyya, R. "Weaving Computational Grids: How Analogous Are They with Electrical Grids?". *Computing in Science and Engineering*, vol. 4-4, Jul 2002, pp. 61-71.

[7] Chu, X.; Nadiminti, K.; Jin, C.; Venugopal, S.; Buyya, R. "Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications". *e-Science and Grid Computing, IEEE International Conference on*, Dec. 2007, pp. 151-159.

[8] Cirne, W.; Brasileiro, F.; Andrade, N.; Costa, L.; Andrade, A.; Novaes, R.; Mowbray, M. "Labs of the World, Unite!!!". *Journal of Grid Computing*, Vol. 4-3, 2006, pp. 225-246.

[9] Coffman, E.; Galambos, J.; Martello, S.; Vigo, D. "Handbook of Combinatorial Optimization", D.-Z. Du and P. M. Pardalos Eds. Kluwer Academic Publishers, 1998.

[10] Colorado, S. G.; Gordon, V. S.; Whitley, D. "Serial and Parallel Genetic Algorithms as Function Optimizers". In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993, pp. 177-183.

[11] De Rose , C. A. F.; Navaux, P. O. A. "Fundamentos de Processamento de Alto Desempenho". In *Anais:* 2^a *Escola Regional de Alto Desempenho*, 2002, pp. 3-29.

[12] De Rose , C. A. F.; Navaux, P. O. A. "Arquiteturas Paralelas", Editora Sagra Luzzatto, 2003.

[13] Dhillon, I. S.; Modha, D. S. "A Data-Clustering Algorithm on Distributed Memory Multiprocessors". In: In Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence, 2008, pp. 245-260.

[14] Dowdy, L. W.; Rosti, E.; Serazzi, G.; Smirni, E. "Scheduling issues in high-performance computing". *SIGMETRICS Perform. Eval. Rev.*, Vol. 26-4, 1999, pp. 60-69.

[15] Foster, I. "The Open Grid Services Architecture 1.0". Capturado em: http://www.ogf.org/documents/GFD.30.pdf, Junho 2009.

[16] Foster, I. "end-to-End Quality of Service for High-end Applications". *Computer Communications*, Vol. 27-14, Set. 2004, pp. 1375-1388.

[17] Foster, I.; Kesselman, C.; Nick, J. M.; Tuecke, S. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations". *The International Journal of High Performance Computing Applications*, Vol 15-3, Ago 2001, pp. 200–222.

[18] Foster, I.; Kesselman, C.; Nick, J. M.; Tuecke, S. "Grid Computing: Making the Global Infrastructure a Reality", John Wiley and Sons, 2003, 433p.

[19] Foster, I.; Kesselman, C. "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, 1999, 558p.

[20] Gepner, P.; Kowalik, M. F. "Multi-Core Processors: New Way to Achieve High System Performance. In *International Symposium on Parallel Computing in Electrical Engineering*, 2006, pp. 9-13.

[21] Halil, B. "Parallel Clustering Algorithms with Application to Climatology". Dissertação de Mestrado, Informatics Institute, Istanbul Technical University, Turkey, 2008, 83p.

[22] Hamscher, V.; Schwiegelshohn, U.; Streit, A.; Yahyapour, R. "Evaluation of Job-Scheduling Strategies for Grid Computing". In *GRID '00: Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, 2000, pp. 191–202.

[23] Hwang, K.; Xu, Z. "Scalable Parallel Computing: Technology, Architecture, Programming, McGraw-Hill, Inc., 1998, 452p.

[24] Jin, R.; Agrawal, G. "A Middleware for Developing Parallel Data Mining Applications". In *Proceedings of the First SIAM International Conference on Data Mining*, 2001, 8p.

[25] Larose, D. "Discovering knowledge in data : an introduction to data mining", John Wiley and Sons, Inc., 2005, 723p.

[26] Li, J.; Liu, Y.; Liao, W.; Choudhary, A. "Parallel Data Mining Algorithms for Association Rules and Clustering". In *Handbook of Parallel Computing: Models, Algorithms and Applications*, 2006, 8p.

[27] Luther, A.; Buyya, R.; Ranjan, R.; Venugopal, S. "Alchemi: A .NET-based Enterprise Grid Computing System". In *International Conference on Internet Computing*, 2005, pp. 269-278.

[28] Maheswaran, M.; Ali, S.; Siegel, H. J.; Hensgen, D.; Freund, R. F. "Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems". In *Eight Heterogeneous Computing Workshop*, 1999, pp. 30–44.

[29] Meligy, A.; Al-Khatib, M. "A Grid-Based Distributed SVM Data Mining Algorithm". *European Journal of Scientific Research*, Vol. 27-3, 2009, pp. 313–321.

[30] Nemeth, Z.; Vaidy, S. "Characterizing Grids: Attributes, Definitions, and Formalisms". Technical report, Journal of Grid Computing, 2003, 12p.

[31] Nist's Mathematical Institute. Java scimark 2. Capturado em: http://math.nist.gov/scimark2/, Janeiro 2010.

[32] Pontifícia Universidade Católica do Rio Grande do Sul. "Laboratório de Alto Desempenho". Capturado em em: www.lad.pucrs.br, Janeiro 2010.

[33] Qiu, X.; Fox, G.; Yuan, H.; Bae, S.; Chrysanthakopoulos, G.; Nielsen, H. "Parallel Data Mining on Multicore Clusters". In *GCC '08: Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing*, 2008, pp. 41–49.

[34] Talia, D.; Trunfio, P.; Verta, O. "The Weka4WS framework for distributed data mining in service-oriented Grids". *Concurrency and Computation: Practice and Experience*, Vol. 20-16, Nov 2008 pp. 1933–1951.

[35] Tan, P.; Steinbach, M.; Kumar, V. "Introduction to Data Mining", Addison-Wesley Longman Publishing Co., Inc., 2006, 476p.

[36] Teodoro, G.; Fireman, D.; Guedes, D.; Meira Jr., W.; Ferreira, R. "Achieving Multi-Level Parallelism in the Filter-Labeled Stream Programming Model". In *ICPP '08: Proceedings of the 2008 37th International Conference on Parallel Processing*, 2008, pp. 287–294.

[37] The Gartner Group. "The Gartner Group. Capturado em: www.gartner.com, Outubro 2009.

[38] Universidade Federal de Minas Gerais. "Projeto Tamanduá". Capturado em: http://tamandua.speed.dcc.ufmg.br, Setembro 2009.

[39] Witten, I. H.; Frank, E. "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, 2005, 563p.

[40] Zaki, J. M.; Ho, C. T.; Agrawal, R. "Parallel Classification for Data Mining on Shared-Memory Multiprocessors". In *ICDE '99: Proceedings of the 15th International Conference on Data Engineering*, 1999, pp. 198-206.

[41] Zhang, Y; Xiong, Z.; Mao, J.; Ou, L. "The Study of Parallel K-Means Algorithm". In *Proceedings of the 2006 6th World Congress on Intelligent Control and Automation*, 2006, pp. 5868–5871.