UMA PROPOSTA DE APLICAÇÃO PARA A SOLUÇÃO DO PROBLEMA DA ÁRVORE GERADORA DE CUSTO MÍNIMO COM GRUPAMENTOS UTILIZANDO *CLUSTER* EM LINUX

FABIANO VIEIRA DE ALVARENGA

2007

FABIANO VIEIRA DE ALVARENGA

UMA PROPOSTA DE APLICAÇÃO PARA A SOLUÇÃO DO PROBLEMA DA ÁRVORE GERADORA DE CUSTO MÍNIMO COM GRUPAMENTOS UTILIZANDO *CLUSTER* EM LINUX

Monografia de Pós-Graduação *"Lato Sensu"* apresentada ao Departamento de Ciência da Computação para obtenção do título de especialista em "Administração em Redes Linux".

Orientadora
Profª. Marluce Rodrigues Pereira

Lavras
Minas Gerais - Brasil
2007

FABIANO VIEIRA DE ALVARENGA

UMA PROPOSTA DE APLICAÇÃO PARA A SOLUÇÃO DO PROBLEMA DA ÁRVORE GERADORA DE CUSTO MÍNIMO COM GRUPAMENTOS UTILIZANDO *CLUSTER* EM LINUX

Monografia de Pós-Graduação *"Lato Sensu"* apresentada ao Departamento de Ciência da Computação para a obtenção do título de especialista em "Administração em Redes Linux".

Aprovada em **29** de **abril** de **2007**

Profª. Simone Markenson Pech

Prof. Sandro Pereira Melo

Profª. Marluce Rodrigues Pereira
(Orientadora)

Lavras
Minas Gerais - Brasil

Aos meus pais e meu querido filho pelo simples fato de existirem e fazerem parte de minha vida.

AGRADECIMENTOS

A Deus por ter me concedido força e perseverança para concluir este trabalho em meio às diversas dificuldades enfrentadas.

A profª. Marluce Rodrigues, pela oportunidade de fazer parte deste trabalho e pelos conhecimentos transmitidos com dedicação e disponibilidade; e principalmente por ter acreditado e investido seu tempo na minha orientação.

Ao prof. Marcelo Lisboa, pelo apoio e ajuda no desenvolvimento da aplicação aqui utilizada para testes; e principalmente pelo incentivo a pesquisa cientifica.

Ao diretor do Colégio Objetivo de Gurupi – TO, Reinaldo Ayres, por ter disponibilizado o laboratório de informática para realização deste trabalho.

Aos professores e amigos do ARL que auxiliaram de forma direta ou indireta na criação deste trabalho.

UMA PROPOSTA DE APLICAÇÃO PARA A SOLUÇÃO DO PROBLEMA DA ÁRVORE GERADORA DE CUSTO MÍNIMO COM GRUPAMENTOS UTILIZANDO *CLUSTER* EM LINUX

RESUMO

O presente trabalho tem como objetivo, desenvolver uma aplicação paralela para solução do problema da Árvore Geradora de custo Mínimo com Grupamentos utilizando-se de *clusters* de Alto Desempenho em Linux, bem como a configuração e testes do *cluster* com aplicações livres, sem nenhum custo financeiro.

SUMÁRIO

CAPÍTULO I 10

1 INTRODUÇÃO 10

1.1 OBJETIVO GERAL 11

1.2 OBJETIVOS ESPECÍFICOS 12

1.3 ESTRUTURA DO TRABALHO 12

CAPÍTULO II 14

2 REVISÃO DA LITERATURA 14

2.1 *CLUSTERS* 14

2.1.1 *Cluster* de Alta Disponibilidade 15

2.1.2 *Clusters* de Alto Desempenho 18

2.1.3 *Clusters Beowulf* 19

2.2 BIBLIOTECAS PARA TROCA DE MENSAGENS EM *CLUSTERS* 22

2.2.1 MPI – *Message Passing Interface* 23

2.2.2 Porque usar MPI? 25

2.3 SISTEMAS OPERACIONAIS 28

2.3.1 Linux 29

CAPÍTULO III 33

3 MATERIAIS E MÉTODOS 33

3.1 MATERIAIS 33

3.2 METODOLOGIA 34

3.3 INSTALAÇÕES E CONFIGURAÇÕES 35

3.3.1 Sistema Operacional Linux 35

3.3.2 Configuração da Rede 36

3.3.3 Configuração do *cluster* 37
3.3.4 Configuração do Nó Mestre 42
3.3.5 Configuração dos Nós Escravos 43
3.3.6 Configuração e instalação da biblioteca MPI 44

CAPÍTULO IV 48

4 TESTES E RESULTADOS COMPUTACIONAIS 48
4.1 APLICAÇÃO UTILIZADA PARA TESTES 48
4.1.1 Paralelização da aplicação 51
4.2 TESTES 53
4.2.1 Solução e tempo de execução da aplicação paralela 54
4.2.2 Analise e comparação dos resultados da aplicação paralela 55

CAPÍTULO V 58

5 CONCLUSÕES E TRABALHOS FUTUROS 58

REFERÊNCIAS BIBLIOGRÁFICAS 60

LISTA DE FIGURAS

2.1	Modelo de um sistema de Alta Disponibilidade.....	17
2.2	Arquitetura de um *cluster Beowulf*.....	21
3.1	Esquema do *cluster* configurado.....	36
3.2	Atributos da rede.....	36
3.3	Conteúdo do arquivo /etc/hosts.....	38
3.4	Gera chaves.....	39
3.5	Copia chave para authorized_keys.....	40
3.6	Altera permissões.....	40
3.7	Teste I.....	40
3.8	Copia chave.....	41
3.9	Teste II.....	41
3.10	Copia chave para authorized_keys.....	41
3.11	Altera permissões.....	42
3.12	Conteúdo do arquivo etc/exports.....	43
3.13	Conteúdo do arquivo etc/fstab.....	44
3.14	Descompactando o arquivo mpich.tar.....	44
3.15	Instalando MPI.....	45
3.16	Conteúdo do arquivo machines.LINUX.....	45
3.17	Conteúdo do arquivo *.bashrc*.....	46
3.18	Compilando o cpi.c.....	46
3.19	Executando o cpi.c.....	47
3.20	Resultado do cpi.c.....	47
4.1	Uma aplicação para o problema da AGMG.....	49
4.2	Solução para a Figura 4.1.....	50
4.3	Pseudocódigo do algoritmo paralelo.....	52
4.4	Tempo computacional das instâncias de 1 a 10.....	56
4.5	Tempo computacional das instâncias de 10 a 15.....	57
4.6	Tempo computacional das instâncias de 15 a 20.....	57

LISTA DE TABELAS

4.1 Instâncias utilizadas para testes..... 53
4.2 Desempenho do algoritmo paralelo (em segundos de CPU)…..... 55

CAPÍTULO I

1 INTRODUÇÃO

Atualmente, a grande busca por poder de processamento vem gerando a necessidade de computadores cada vez mais rápidos e eficientes. Diversas áreas requerem cada vez mais serviços de processamento rápido, resolução de problemas cada vez maiores em períodos de tempo cada vez mais curtos. Estas áreas são, por exemplo, multimídia; otimização combinatória; cálculos complexos; redes, no caso de servidores sobrecarregados seja web, banco de dados ou e-mail, ou seja, servidores com grandes volumes de dados, e entre outras. Desta forma, o Processamento Paralelo se torna uma ótima alternativa para se obter alto desempenho computacional. Em [SIMÕES *et al*., 2003] são apresentados alguns problemas comuns que podem ser resolvidos utilizando Processamento Paralelo. A previsão do tempo, exploração de petróleo, dinâmica dos fluidos, visão computacional, aerodinâmica e o mapeamento do genoma humano são exemplos deste tipo de problema.

Com o surgimento de um paradigma denominado *cluster* de computadores junto à evolução das redes de computadores, o acesso ao alto desempenho se tornou mais acessível. O termo *cluster*, ou aglomerado de computadores, que vem do inglês, pode ser formado por um conjunto de computadores convencionais (*Desktops*) agrupados fisicamente em um ambiente, ou simplesmente por computadores dedicados [*CLUSTER*, 2007].

Segundo [SIMÕES *et al*., 2003], a aquisição de supercomputadores paralelos para se obter acesso ao alto desempenho nem sempre constitui a melhor alternativa, por serem máquinas usualmente caras. Atualmente, a utilização de *clusters* de estações de trabalho tem permitido que muitas empresas

e/ou instituições tenham acesso a um alto poder computacional a um baixo custo. Devido a isso, os *cluster*s têm se tornado cada vez mais uma alternativa aos supercomputadores.

De forma geral, montar um *cluster* que tenha o mesmo desempenho que um supercomputador (multiprocessador) é muito mais barato do que montar um supercomputador. Isto porque os PCs que são utilizados nos nós do *cluster* estão cada vez mais rápidos e baratos, bem como os dispositivos de rede.

É indispensável lembrar que, os *clusters* podem ser usados em uma infinidade de aplicações, basicamente em qualquer uma que exija alta disponibilidade e grandes capacidades de processamento, solucionando problemas antigos da supercomputação, como o altíssimo custo de aquisição e manutenção dos equipamentos, uso de *softwares* proprietários e caros, total dependência dos fornecedores, bem como dificuldades de atualizações [PITANGA, 2004].

Visto que um *cluster* pode ser definido por um conjunto de nós processadores, ou seja, por estações de trabalho ou computadores pessoais, surgiram as motivações para realização deste trabalho. A seguir, são apresentados os objetivos gerais e específicos.

1.1 OBJETIVO GERAL

Este trabalho tem como objetivo geral propor uma implementação paralela para solução do problema da Árvore Geradora de custo Mínimo com Grupamentos utilizando um cluster em Linux. Para isso, será necessária a configuração e teste de um cluster para computação de alto desempenho, utilizando 8 máquinas homogêneas. O sistema operacional GNU/Linux e a biblioteca de passagem de mensagem, MPI, para realizar computação paralela fazem parte desta

configuração. Desta forma, serão utilizadas somente aplicações livres, sem nenhum custo.

1.2 OBJETIVOS ESPECÍFICOS

- Configuração do *cluster* em oito estações de trabalho homogêneas, utilizando GNU/Linux;
- Instalação e configuração da biblioteca MPI;
- Paralelização da aplicação utilizada no trabalho "Melhorando o Desempenho da Metaheurística GRASP Utilizando a Técnica Path-Relinking: Uma Aplicação para o Problema da Árvore Geradora de Custo Mínimo com Grupamentos" por [ALVARENGA e ROCHA, 2006] . O objetivo da paralelização desta aplicação é reduzir o tempo computacional da mesma;
- Realizar testes no *cluster* com a aplicação paralela citada acima e colher os resultados da execução da aplicação com um, dois, quatro e oito processadores;
- Comparar e analisar os resultados obtidos, de forma a mostrar que a execução utilizando mais de uma máquina (*cluster)* é mais rápida do que utilizando somente uma máquina.

1.3 ESTRUTURA DO TRABALHO

O presente trabalho está dividido em cinco capítulos, incluindo esta introdução. O restante está organizado como segue.

No segundo capítulo é apresentada a revisão bibliográfica, iniciando-se pelos *clusters* de Alta Disponibilidade e Alto Desempenho de Computação.

Posteriormente, são apresentadas algumas informações das principais bibliotecas de troca de mensagens em *clusters*: PVM e MPI, com ênfase em MPI. Por fim, este capítulo apresenta uma abordagem sobre sistemas operacionais com foco no Sistema Operacional GNU/Linux.

O terceiro capítulo apresenta os materiais e métodos utilizados no desenvolvimento do trabalho bem como a metodologia adotada e as instalações e configurações do *cluster*.

O quarto capítulo, apresenta os testes e resultados computacionais realizados sobre o *cluster*, bem como a aplicação utilizada para teste, e sua paralelização; as instâncias utilizadas para teste; o desempenho do algoritmo da aplicação paralela; e por fim os resultados obtidos tanto sobre a qualidade das soluções fornecidas quanto pelo tempo de execução. Também são fornecidas comparações com os resultados obtidos.

Finalmente, o trabalho é concluído no quinto capítulo, apresentando uma análise final sobre os resultados obtidos com este trabalho e também são apresentadas as sugestões e recomendações para trabalhos futuros.

CAPÍTULO II

2 REVISÃO DA LITERATURA

2.1 *CLUSTERS*

O termo *cluster*, ou aglomerado de computadores, que vem do inglês, pode ser formado por um conjunto de computadores convencionais (*Desktops*) agrupados fisicamente em um ambiente, ou simplesmente por computadores dedicados. De forma geral [MARTINS *et al*. 2005] define um *cluster* como um sistema onde dois ou mais computadores independentes trabalham de maneira conjunta, coordenados por um sistema operacional paralelo, para realizar processamento pesado. Para atingir este objetivo, os computadores dividem as tarefas de processamento através da troca de mensagens, fazendo com que elas sejam realizadas paralelamente. Desta forma, os processadores trabalham como se fossem um único computador.

Os *clusters* trazem consigo inúmeras vantagens em sua utilização. Em [JÚNIOR e FREITAS, 2005] algumas destas vantagens são destacadas:

- Alto Desempenho - possibilidade de se resolver problemas muito complexos através do processamento paralelo, diminuindo o tempo de resolução do mesmo;
- Escalabilidade - possibilidade de que novos componentes sejam adicionados à medida que cresce a carga de trabalho. O único limite é capacidade da rede;
- Tolerância a Falhas - o aumento de confiabilidade do sistema como um todo, caso alguma parte falhe;

- Baixo Custo - a redução de custo para se obter processamento de alto desempenho utilizando-se simples PCs;
- Independência de fornecedores - utilização de *hardware* aberto, software de uso livre e independência de fabricantes e licença de uso.

É importante colocar que os *clusters* podem ser configurados de acordo com sua finalidade, que podem ser divididas em: *clusters* de Alta Disponibilidade, *clusters* de Alto Desempenho, como no caso do *clusters Beowulf,* e *cluster* de Balanceamento de Carga, que como o próprio nome diz, este tem como finalidade distribuir carga entre servidores de modo que um único servidor não fique sobrecarregado e outros sem carga. Também pode ser implementado para fornecer Alto Desempenho e possivelmente Alta Disponibilidade. O *cluster* de Alta Disponibilidade tem a finalidade de manter um determinado serviço de forma segura o maior tempo possível. O *cluster* de Alto Desempenho é uma configuração designada a prover grande poder computacional, maior que somente um único computador poderia oferecer em capacidade de processamento. Adiante são apresentadas as principais características dos *clusters* de Alto Desempenho e Alta Disponibilidade.

2.1.1 *Cluster* de Alta Disponibilidade

Os *clusters* de Alta Disponibilidade são implementados para fornecer uma disponibilidade de serviços de forma ininterrupta, através de operações redundantes nos nós, ou seja, se um deles falhar, então um outro nó iria executar sua tarefa, de forma que o *cluster* não se desliga por inteiro. Tais *clusters* são essenciais para executar missões críticas. Cabe aqui lembrar que existem classes de disponibilidade que se dividem em: disponibilidade básica, alta disponibilidade e disponibilidade contínua. É importante mencionar que a

disponibilidade encontrada em computadores comuns de mercado é a básica, que apresentam uma disponibilidade de 99% a 99,9% [PITANGA, 2002].

Segundo [JÚNIOR e FREITAS, 2005] um *cluster* de alta disponibilidade tem como função essencial deixar um sistema no ar vinte e quatro horas por dia, sete dias por semana, ou que não suporte paradas de meia hora ou alguns minutos. São estas paradas não planejadas que influenciam diretamente na qualidade do serviço e nos prejuízos financeiros que estas podem acarretar. Nos *clusters* de alta disponibilidade, os equipamentos são usados em conjunto para manter um serviço ou equipamento com total disponibilidade, replicando serviços e servidores, o que evita máquinas paradas, ociosas, esperando apenas o outro equipamento ou serviço paralisar, passando as demais a responder por elas normalmente. É claro que com isso pode-se ter perda de desempenho e poder de processamento, mas o principal objetivo será alcançado, ou seja, não paralisar o serviço. A disponibilidade dos sistemas torna-se então uma questão vital de sobrevivência empresarial, ou seja, se o sistema parar a empresa e ou serviço também pára.

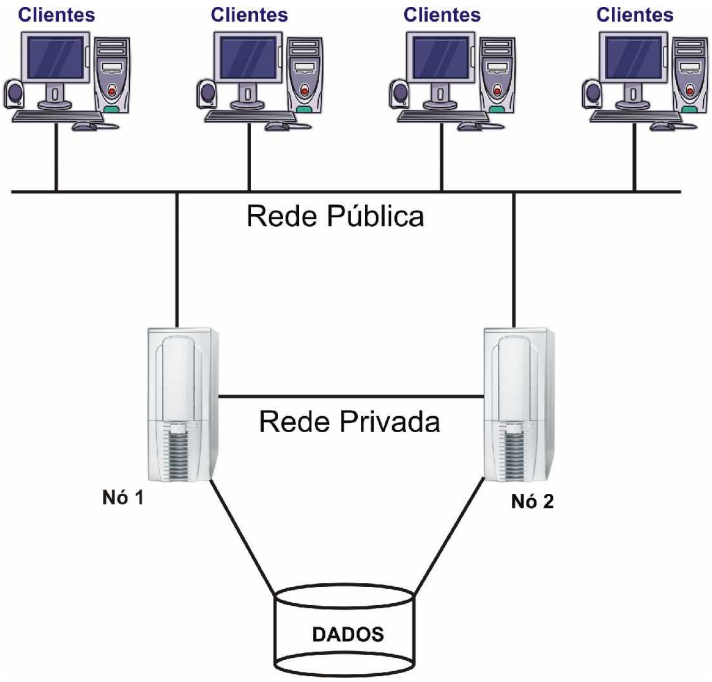

Figura 2.1 - Modelo de um sistema de Alta Disponibilidade

Pode se perceber que uma pequena falha ou interrupção no funcionamento deste tipo de *cluster*, pode causar sérios prejuízos, dependendo das aplicações que possam estar sendo executas no mesmo. Desta forma, é necessário que os equipamentos sejam utilizados em conjunto, com o objetivo de manter os serviços com total disponibilidade.

Vale lembrar que *clusters* de Alta Disponibilidade são caracterizados como *clusters* dedicados e heterogêneos. A Figura 2.1, mostra um exemplo simples de uma configuração de *clusters* de Alta Disponibilidade. Observando a

Figura 2.1 podemos notar que todos os clientes da rede pública podem ter acesso aos dados através de dois nós interligados por uma rede privada, garantindo assim o acesso aos dados pela rede pública, caso um dos nós venha a parar.

2.1.2 *Clusters* de Alto Desempenho

Essa classe de *cluster* tem como foco o desenvolvimento de máquinas paralelas com alto poder computacional, que podem ser utilizadas em estudo de algoritmos de processamento paralelos e construção de aplicações paralelas e distribuídas [PITANGA, 2004]. Desta forma, entende-se que esta classe de *clusters* tem como objetivo diminuir o tempo para a resolução de problemas computacionais. Sistemas de *clusters* de Alto Desempenho podem haver em Computação Paralela e Computação Distribuída. Onde se entende que Computação Paralela se refere à submissão de processos para mais de um processador distribuindo assim o esforço computacional para resolução do problema e Computação Distribuída tem por objetivo compartilhar recursos de computadores de maneira a otimizá-los. Pode-se citar como exemplo, um *cluster* de Alto Desempenho, os *clusters* desenvolvidos para programação paralela que são os *cluster Beowulf*, que serão apresentados na Seção 2.1.3.

Quando se fala em alto desempenho, imagina–se um supercomputador dedicado, com custo elevadíssimo e de difícil operação, mas isso pode ser traduzido em processamento paralelo e processamento distribuído, realizado sobre *clusters* de computadores [PITANGA, 2004]. Os *clusters* utilizados para esse fim, são os *clusters* de Alto Desempenho que, entre outras vantagens, oferecem um custo bem reduzido frente aos supercomputadores, o que viabiliza o uso do processamento de alto desempenho na solução de problemas em diversas áreas.

De acordo com [JÚNIOR e FREITAS, 2005], *cluster* de Alto Desempenho é um tipo de sistema para processamento paralelo e distribuído que consiste de uma coleção de computadores interconectados, trabalhando juntos como um recurso de computação simples e integrado. Um nó do *cluster* pode ser um simples sistema multiprocessado (PCs, estações de trabalho ou Multiprocessadores Simétricos – SMPs[1]) com memória, dispositivo de entrada/saída de dados de um sistema operacional. No entanto, esse sistema pode fornecer características e benefícios (serviços rápidos e confiáveis) encontrados somente em sistemas com memória compartilhada (SMP), como os supercomputadores.

2.1.3 *Clusters Beowulf*

O *cluster Beowulf* surgiu com o objetivo de suprir a crescente necessidade de elevada capacidade de processamento em diversas áreas científicas, possibilitando a construção de sistemas computacionais que apresentassem um custo viável e um alto poder de processamento.

Em 1994, a NASA (agência espacial norte-americana) necessitava de um equipamento com poder de processamento da ordem de um gigaflop. Como uma máquina com esse desempenho custava em torno de um milhão de dólares, foi considerado um investimento muito alto para ser aproveitado apenas por um grupo de pesquisadores. Buscando uma solução mais viável, financeiramente, os pesquisadores Thomas Sterling e Donald J. Becker interligaram 16 computadores pessoais, cada um com um microprocessador 486, usando Linux e uma rede *Ethernet*, visando alcançar o desempenho desejado com um custo menor [MERKEY, 2007].

[1]http://www.tldp.org/HOWTO/SMP-HOWTO.html.

Este *cluster* atingiu a marca de 70 megaflops, que era uma velocidade comparável à de pequenos supercomputadores comerciais, e foi utilizado para manipular informações recebidas pelo satélite. O custo total do sistema foi dez por cento menor que o de uma máquina com processamento equivalente [OLIVEIRA, 2004]. Desde então, diversas instituições têm construído os seus próprios *clusters Beowulf*. Atualmente existem *clusters* deste tipo com mais de milhares de nós.

Assim começaram os projetos com os *clusters Beowulf* que, além das características comuns a um *cluster*, devem possuir as seguintes características, conforme as informações contidas em [BEOWULF, 2004] e [PITANGA 2004]:

- Nenhum componente feito sob encomenda;
- Independência de fornecedores de *hardware* e *software*;
- Periféricos escaláveis;
- *Software* livre de código aberto;
- Uso de ferramentas de computação distribuída disponível livremente com alterações mínimas;
- Retorno à comunidade do projeto e melhorias.

A arquitetura do *cluster Beowulf* é colocada por [ROCHA, 2003] da seguinte forma: o *clusters Beowulf* é uma arquitetura de computador que pode ser usada para computação paralela. É um sistema que normalmente consiste em nós clientes (são computadores, porém não são equipados com monitor de vídeo, teclado e mouse, possuindo somente placa-mãe, microprocessador, memória, disco rígido e placa de rede) e um nó servidor (também é um computador, mas o mesmo funciona como a *interface* para o mundo exterior, por este motivo ele é equipado com: monitor de vídeo, placa de vídeo, teclado e *mouse*), conectados

por uma rede *Ethernet* ou algum outro tipo de rede local. Na Figura 2.2 pode-se observar a arquitetura de um *cluster Beowulf*.

O sistema é construído usando-se componentes de *hardware* largamente comercializados, placas de rede e concentradores (*Hubs* ou *swicthes*). Não contém nenhum componente de *hardware* desenvolvido especialmente para um propósito especifico e é trivialmente reproduzível. *Beowulf* também usa *software* livre como o sistema operacional GNU/Linux e bibliotecas para comunicação em rede, como *Parallel Virtual Machine* (PVM) e *Message Passing Interface* (MPI). O nó servidor controla o *cluster* inteiro e distribui os processos aos nós clientes. Estas bibliotecas MPI e PVM serão apresentadas na próxima Seção e posteriormente os sistemas operacionais incluindo o GNU/Linux.

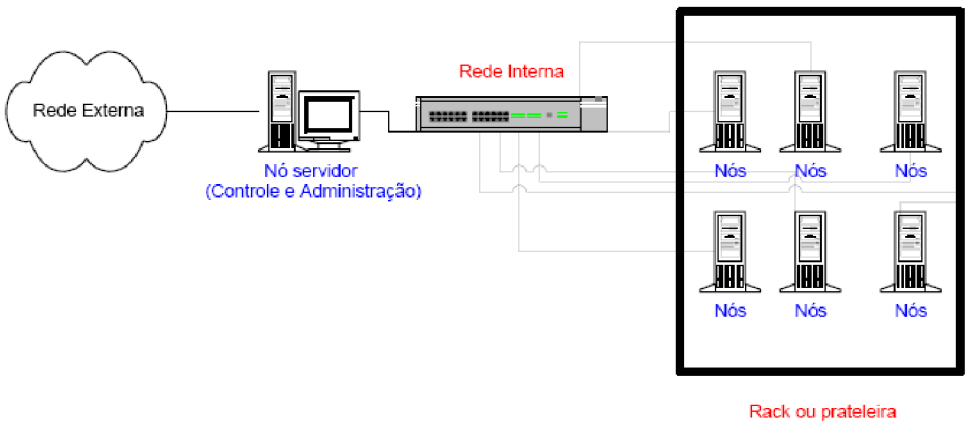

Figura 2.2 - Arquitetura de um *cluster Beowulf*
Fonte: Rocha, 2003, p. 27.

Uma colocação muito importante quanto a utilização de aplicações livres em *cluster Beowulf* colocado por [ROCHA, 2003] que pode-se concordar, é que mesmo utilizando-se de *software* disponível livremente a arquitetura *Beowulf* é

tão sofisticada, robusta e eficiente quanto qualquer outra comercial. Uma característica chave do *Beowulf* é o uso do sistema operacional Linux, assim como de bibliotecas para troca de mensagens (PVM e MPI) de livre distribuição. Isto permite fazer alterações no Linux para dotá-lo de novas características que facilitam a implementação de aplicações paralelas.

A área de aplicação dos *clusters Beowulf* é muito diversificada. São geralmente as aplicações seqüenciais muito grandes, que necessitam de enorme quantidade de memória e tempo de processamento. Segundo [ROCHA, 2003], com um *cluster* paralelo de Linux é possível construir um servidor de arquivos que é mais rápido que a maioria dos servidores comerciais. Isso pode ser feito por causa dos sistemas múltiplos e conexões múltiplas de rede inerente nestes *cluster*s, sendo útil para aplicações como sistemas de banco de dados, onde os principais gargalos de desempenho acontecem quando não podem ser movidos dados efetivamente do disco para memória, ou em aplicações de *web*, onde a velocidade de transação é crítica para o sucesso de uma aplicação.

2.2 BIBLIOTECAS PARA TROCA DE MENSAGENS EM *CLUSTERS*

Quando se fala em bibliotecas para troca de mensagens em *clusters* o *Parallel Virtual Machine* (PVM) e o *Message Passing Interface* (MPI), merecem destaque.

PVM (Parallel Virtual Machine) [GEIST, 1994] é uma biblioteca de comunicação que emula computação concorrente heterogênea de propósitos gerais em computadores interconectados de variadas arquiteturas. A idéia do PVM é montar uma máquina virtual de *n* processadores e usá-los para enviar tarefas e receber os resultados.

MPI (*Message Passing Interface*) [MPI FORUM, 1994], normalmente referida como MPI *standard*, é um esforço para melhorar a disponibilidade de eficiência e portabilidade do software, garantindo as necessidades de aplicações paralelas e distribuídas. O MPI *standard* define um padrão de troca de mensagens onde cada fabricante é livre para implementar as rotinas, já com sintaxe definida, utilizando características exclusivas de sua arquitetura.

Este trabalho não tem como objetivo fazer uma comparação entre MPI e PVM. Desta forma serão apresentadas nas próximas Seções apenas algumas das principais características referentes à MPI. . Alguns trabalhos como [LINHALIS e FIATS, 1998] e [FREITAS, 2006], podem apresentar um estudo comparativo entre MPI e PVM.

2.2.1 MPI – *Message Passing Interface*

O MPI é uma biblioteca de passagem de mensagem, desenvolvida para ser padrão em ambientes de memória distribuída, em *Message-Passing* (passagem de mensagem) e em computação paralela. O MPI é portável para qualquer arquitetura, tem aproximadamente 125 funções para programação e ferramentas para se analisar a performance. A plataforma alvo para o MPI, são ambientes de memória distribuída, máquinas paralelas massivas, *clusters* de estações de trabalho e redes heterogêneas.

O MPI define um conjunto de rotinas para facilitar a comunicação entre os processos. Segundo [BRAUN, 2006], o MPI possui rotinas para programação em linguagens distintas como: Fortran 77/90[2], C[3] / C++[4]. Logo, os programas são escritos e compilados em uma determinada linguagem e ligados à biblioteca

[2] http://pt.wikipedia.org/wiki/Fortran.
[3] http://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_C
[4] http://pt.wikipedia.org/wiki/C++

MPI. Informações detalhadas sobre MPI podem ser encontradas no Fórum de MPI [MPI, 2006]. No entanto, vale salientar que existem tantas implementações diferentes de MPI quanto existem distribuições Linux. Algumas implementações do MPI existentes são:

- MPI-F: IBM *Research*
- UNIFY: *Missipi State University*
- CHIMP: *Edinburg Parallel Computing Center*
- LAM: *Ohio Supercomputer Center*
- MPICH: ANL/MSU – *Argone National Laboratory* e *Missipi State University*

Dentre tais implementações a LAM (*Local Area Multicomputer*) e MPICH (*MPIChameleon*), merecem destaque, pois além de seguirem o padrão MPI, as mesmas incorporam uma série de funcionalidades adicionais, no caso da MPICH tem suporte a máquinas SMP e interoperabilidade entre máquinas heterogêneas. Já a implementação LAM é a única distribuída através de pacotes RPM (*Red Hat Packages Management*) o que a torna muito popular nas distribuições GNU/Linux que utilizam essa forma de empacotamento.

De acordo com [JUNIOR e FREITAS, 2005], o padrão para *Message Passing*, denominado MPI (*Message Passing Interface*), foi projetado em um fórum aberto constituído de pesquisadores, acadêmicos, programadores, usuários e vendedores, representando 40 organizações ao todo. As principais características que compõem o MPI são:

- Eficiência - Foi cuidadosamente projetado para executar eficientemente em máquinas diferentes. Especifica somente o funcionamento lógico das

operações. Deixa em aberto a implementação. Os desenvolvedores otimizam o código usando características específicas de cada máquina;

- Facilidade - Define uma interface não muito diferente de outros padrões, como PVM. e acrescenta algumas extensões que permitem maior flexibilidade;
- Portabilidade - É compatível para sistemas de memória distribuída, NOWs (*network of workstations*) e uma combinação deles;
- Transparência - Permite que um programa seja executado em sistemas heterogêneos sem mudanças significativas;
- Segurança - Provê uma interface de comunicação confiável. O usuário não precisa preocupar-se com falhas na comunicação;
- Escalabilidade - O MPI suporta escalabilidade sob diversas formas, por exemplo, uma aplicação pode criar subgrupos de processos que permitem operações de comunicação coletiva para melhorar o alcance dos processos.

Diante da extensão da documentação relacionada à biblioteca de troca de mensagem MPI, este trabalho recomenda a leitura de [LINHALIS e FIATS, 1998], [FREITAS, 2006], [ROCHA, 2003], [PAIVA e JUSTO, 2004], [MPI FORUM, 1994], [MPI, 2006], [LONDEROS, 2006] e [SANCHES, 2003] para maiores informações, tais como: arquitetura, protocolos de comunicação, implementações, padrões, histórico, entre outras.

2.2.2 Porque usar MPI?

Como foi colocado anteriormente, este trabalho não tem como objetivo fazer uma comparação ente MPI e PVM. Desta forma esta Seção apresenta apenas as

razões pela quais o MPI foi escolhido para a elaboração deste trabalho. Até mesmo porque deve-se ter muita cautela para afirmar que uma biblioteca seja, no geral, melhor que a outra. Diante disto segue algumas das inúmeras razões colocadas por [RIGONI *et al*., 1999], [PITANGA, 2004], [LINHALIS e FIATS, 1998], [LONDEROS, 2006], [FREITAS, 2006], [ROCHA, 2003] e [PAIVA e JUSTO, 2004] para se usar MPI ao invés de PVM.

- Facilidade na programação: MPI é mais simples e mais legível que PVM. O simples fato de no PVM existirem *n* códigos diferentes de uma mesma aplicação e no MPI haver apenas um, já o torna mais atraente.
- Portabilidade e independência de plataforma computacional do MPI. "Por exemplo, um código MPI que foi escrito para uma arquitetura IBM RS-600 utilizando o sistema operacional AIX pode ser portado para uma arquitetura SPARC (*Scalable Processar ARChitecture* - Arquitetura de Processadores Escaláveis) com SO Solaris ou PC com Linux com quase nenhuma modificação no código fonte da aplicação" [PITANGA, 2004];
- MPI tem mais de um sistema livre disponível como ferramenta de desenvolvimento. As principais são: LAM e MPICH. As ferramentas de desenvolvimento não se unem a interface de rede, pois operam no nível do sistema operacional;
- MPI define uma terceira pessoa, que executa o mecanismo. Um construtor da ferramenta pode extrair a informação do perfil e usos do MPI provendo a interface padrão do perfil do MPI em uma biblioteca separada, sem ter acesso ao código fonte durante a etapa de desenvolvimento, na prática, evita a modificação da biblioteca MPI;

- MPI tem comunicação assíncrona completa. De maneira imediata envia e recebe as operações, onde o cálculo pode ser sobreposto completamente;
- Os grupos de MPI são sólidos, eficientes e deterministas. A quantidade de membros do grupo é estática. Não condiciona aos usuários do grupo ao qual pertencem nem ao processos que executam que se mantenham de maneira independente ao grupo. A nova formação de um grupo é coletiva e se distribui, não se centraliza a informação a cerca dos membros do grupo;
- MPI gerencia de maneira eficiente armazenadores intermediários de mensagens. As mensagens são enviadas e recebidas das estruturas de dados do usuário, não de armazenadores intermediários, se não de maneira direta desde dentro da biblioteca de comunicação. O buffering pode, em alguns casos, ser evitado totalmente.
- A sincronização do MPI protege os processos de terceiros (outros usuários). Toda a comunicação dentro de um grupo particular de processos está marcada com uma variável adicional de sincronização, determinada pelo sistema. Os produtos de software independentes dentro do mesmo processo não têm que se preocupar em determinar etiquetas às mensagens.
- MPI pode programar eficientemente para MPP (Máquinas Massivamente Paralelas) e *clusters*. Uma topologia virtual que reflita o padrão de comunicação de uso se pode associar a um grupo de processos. Uma utilização de MPP com MPI poderia utilizar essa informação para fornecer processos aos processadores de uma maneira que otimiza as trajetórias de comunicação.

- MPI é totalmente portável: pode ser recompilado e funcionar em qualquer ambiente. Com topologias virtuais e o gerenciamento intermediário eficiente, por exemplo, um processo que se move desde um *cluster* a um MPP pode contar com bom funcionamento.
- MPI se especifica formalmente como um sistema portável. Os processos têm um tempo determinado de vida. Ao executar um processo este tem um tempo de vida em caso de não retornar a fonte;
- MPI é um padrão. Suas características e comportamento foram estabelecidos em um conselho, em um fórum aberto, podendo ser trocado somente pelo mesmo processo.

2.3 SISTEMAS OPERACIONAIS

Inicialmente vale lembra que este trabalho tem como um dos seus objetivos utilizar somente aplicações livres para o desenvolvimento do *clusters*, de forma que o mesmo seja finalizado com o menor custo possível. Assim, os Sistemas Operacionais proprietários como, por exemplo, Solaris[5], AIX[6], True64[7], HPUX[8], entre outros que além de caros, a maioria requerem *hardware* também proprietário, tornando-os assim dispensáveis a este trabalho. Ao contrário aos sistemas operacionais proprietários, o Linux é um *software* livre e suporta completamente as mais variadas arquiteturas e pode ser adquirido de forma gratuita.

Segundo [ROCHA, 2003], que segue as finalidades do projeto de *clusters* tipo *Beowulf*, o sistema operacional teria que atender as seguintes características: ser estável, seguro, ter capacidade de trabalhar em sistemas

[5] http://www.sun.com/software/solaris/
[6] http://www-03.ibm.com/servers/aix/
[7] http://h30097.www3.hp.com/
[8] www.hp.com/go/hpux/

multiusuário e multitarefa, poder operar sobre as mais diversas arquiteturas de computadores, ser gratuito e ter seu código fonte disponível. Atendendo estas características aparecem duas versões do Sistema Operacional Unix conhecidas como Linux e FreeBSD[9].

Qualquer um dos dois (Linux ou FreeBSD) seria uma ótima escolha, mas devido à sua grande popularidade e crescente uso na comunidade optou-se por usar o Linux como sistema operacional no ambiente de *cluster*, pois ele se adequa muito bem. Além do preço, que às vezes determina a realização ou não de um projeto, especialmente num ambiente acadêmico ou em empresas pequenas, ainda se tem a vantagem do código fonte aberto [ROCHA, 2003]. Além do mais, no ano de 2002 ouve um grande crescimento da utilização de *clusters* Linux no mundo. Isso levou a um grande investimento nas empresas que desenvolvem essas tecnologias. Isso tudo se deve ao fato de que hoje em dia a tecnologia de *clusters* está estável e disponível neste sistema, afirma [PAIVA e Justo, 2004].

2.3.1 Linux

O Linux é um sistema operacional Unix multitarefa, multiusuário e multiprocessado, desenvolvido há poucos anos graças aos esforços coletivos da comunidade tecnológica e, em especial, de seu idealizador, Linus Torvalds. Em agosto de 1991, na Finlândia, esse jovem de 21 anos de idade chamado Linus Torvalds iniciou o projeto do Linux. O estudante universitário desejava desenvolver uma versão do Unix que rodasse em micros PC AT e compatíveis, mas que fosse diferente dos sistemas Unix já existentes, cujo preço era exorbitante para o usuário comum. Linus chegou a divulgar a idéia num *newsgroup* de que participava e embalado pelo projeto, programou sozinho a

[9] http://www.freebsd.org/

primeira versão do *kernel* do Linux (núcleo do sistema operacional). Linus Torvalds se inspirou em Andy Tanembaum, criador do Minix, outro sistema operacional Unix, do qual Linus era usuário. A nova criação foi batizada com o nome de Linux, vocábulo que resultou da fusão de Unix com o primeiro nome de seu criador, Linus.

Depois de finalizar o *kernel*, Linus deu ao seu projeto o rumo que desencadeou seu grande sucesso: passou a distribuir o código fonte do *kernel* pela Internet, para que outros programadores pudessem aprimorar o sistema. Assim, várias empresas e programadores de todo o planeta contribuíram com seus conhecimentos para melhorar e fazer do Linux o sistema operacional potente e diversificado que é hoje. Esse foi o segredo: trabalho cooperativo e voluntário. Linus distribuiu seu trabalho sem cobrar nada e em troca, exigiu que os outros programadores envolvidos no projeto fizessem o mesmo. Por isso o Linux é gratuito.

Por causa da abertura do código fonte ao mundo, não existe uma, mas muitas distribuições do Linux no mercado. Todas possuem características especiais que as diferenciam entre si. Na verdade, não existe "o Linux", existem "os Linux". Mas apesar de singulares, todas essas versões são compatíveis, porque utilizam o mesmo *kernel*. Entre inúmeras distribuições Linux, algumas merecem destaque devido sua popularidade entre elas estão: Slackware[10], Fedora Core[11], Mandriva[12], Ubuntu[13], Debian[14], Gentoo[15] , entre outras.

O Linux está sob a licença GPL (*General Public License*), permite que qualquer um possa usar os programas que estão sob ela, com o compromisso de

[10] http://www.slackware.com/
[11] http://fedoraproject.org/wiki/
[12] http://www.mandriva.com/
[13] http://www.ubuntu.com/
[14] http://www.debian.org/
[15] http://www.gentoo.org/

não tornar os programas fechados e comercializados. Ou seja, pode-se alterar qualquer parte do GNU/Linux, modificá-lo e até comercializá-lo, mas não se pode fechá-lo.

Além dos pontos positivos já citados, o GNU/Linux possui todas as características que se pode esperar de um Unix moderno, dentre elas estão[ROCHA, 2003]:

- Multitarefa real;
- Memória virtual;
- Modularização - O Linux somente carrega para a memória o que é usado durante o processamento, liberando totalmente a memória assim que o programa/dispositivo é finalizado.
- Suporte a diversos dispositivos e periféricos disponíveis no mercado, tanto os novos como obsoletos.
- Suporte a nomes extensos de arquivos e diretórios (255 caracteres).
- Pode ser executado em arquiteturas diferentes.
- Suporte a até 16 processadores no mesmo sistema com versão do *Kernel* 2.4;
- Biblioteca compartilhada;
- Gerenciamento de memória próprio;
- Rede TCP/IP (incluindo SLIP/PPP/ISDN) e
- X Windows.
- Suporte a diversos sistemas de arquivos.

Quanto a *clusters* em Linux pode-se afirma que as vantagens são diversas, entre elas destacam-se o fato do Linux ser um sistema robusto, que dá suporte desde aplicações simples até aplicações extremamente complexas e por ser possível dotá-lo de novas características que facilitam a implementação para aplicações paralelas. Isso é possível principalmente pelo fato do mesmo estar

disponível na forma de código objeto, bem como em código fonte [OLIVEIRA, 2004].

CAPÍTULO III

3 MATERIAIS E MÉTODOS

Nesta Seção são apresentados os materiais e métodos utilizados no desenvolvimento deste trabalho, bem como a metodologia, configuração do *cluster* e configuração da biblioteca de troca de mensagens MPI.

3.1 MATERIAIS

Parte da pesquisa bibliográfica foi realizada na biblioteca da Fundação UNIRG[16] (Universidade Regional de Gurupi) de Gurupi - TO. Também foram utilizados os recursos da Internet como meio de obtenção de material de pesquisa e principalmente de artigos científicos. Todo o material bibliográfico utilizado para o desenvolvimento deste trabalho está citado e referenciado no texto e nas referências bibliográficas.

Todos os recursos computacionais necessários para o desenvolvimento deste trabalho, foram disponibilizados pelo laboratório de informática (LABIN) do Colégio Objetivo[17] de Gurupi – TO. Segue as especificações de todos os equipamentos utilizados para a realização deste trabalho:

- 1 Hub 16 portas 100Mpbs
- 8 computadores com a seguinte configuração:
    - Processador: Pentium III de 1 Ghz
    - Memória: 512 MB de Memória
    - HD: Sansung 40 GB

[16] http://www.unirg.edu.br
[17] http://www.objetivogurupi.com.br

- Placa de Rede: SIS 900 PCI Fast Ethernet Adapter

3.2 METODOLOGIA

Inicialmente a metodologia aplicada no desenvolvimento deste trabalho consiste em uma revisão da literatura a respeito de *clusters*, bibliotecas de troca de mensagens em *clusters* e sistemas operacionais.

Com base na revisão bibliográfica levantada no decorrer do trabalho, foi possível identificar as principais características que se fazem necessárias para configuração de *cluster* de Alto Desempenho. Desta forma, este trabalho adotou como sistema operacional o Linux e a biblioteca de troca de mensagens MPI.

A distribuição escolhida para a realização deste trabalho, foi a Fedora Core 5. Esta distribuição foi instalada no laboratório de informática do Colégio Objetivo por ser uma distribuição estável, por possuir uma interface bastante amigável é principalmente pela facilidade de instalação/remoção de aplicativos através *yum* (*Yellow dog Updater*, *Modified*, trata-se de um *software* desenvolvido pela *Duke University* para ser um instalador, atualizador e removedor de pacotes RPM, semelhante ao apt-get do Debian.). Desta forma o *cluster* se utilizará deste laboratório com o Sistema Operacional Linux Fedora Core 5.

Já a opção pela biblioteca de troca de mensagens MPI, se deu devido a todas as especificações colocadas na Seção 2.2.2. Em síntese a esta Seção, pode-se colocar como um dos principais motivos a esta escolha, a facilidade de programa utilizando MPI, ou seja, a familiaridade com a mesma. Entre outros motivos pode-se colocar também a portabilidade e independência de plataforma computacional do MPI.

3.3 INSTALAÇÕES E CONFIGURAÇÕES

Para as configurações e instalações do *cluster* de Alto Desempenho, foram utilizadas oito máquinas, interligadas por uma rede *Fast-Ethernet* (100Mpbs), conforme demonstrado na Figura 3.1. Onde a estrutura do *cluster* desenvolvido é formada por um nó controlador (nó mestre), e sete nós escravos. É importante colocar que não há uma limitação para a quantidade de nós escravos, desta forma pode-se inserir, substituir ou remover nós do *cluster* quando necessário.

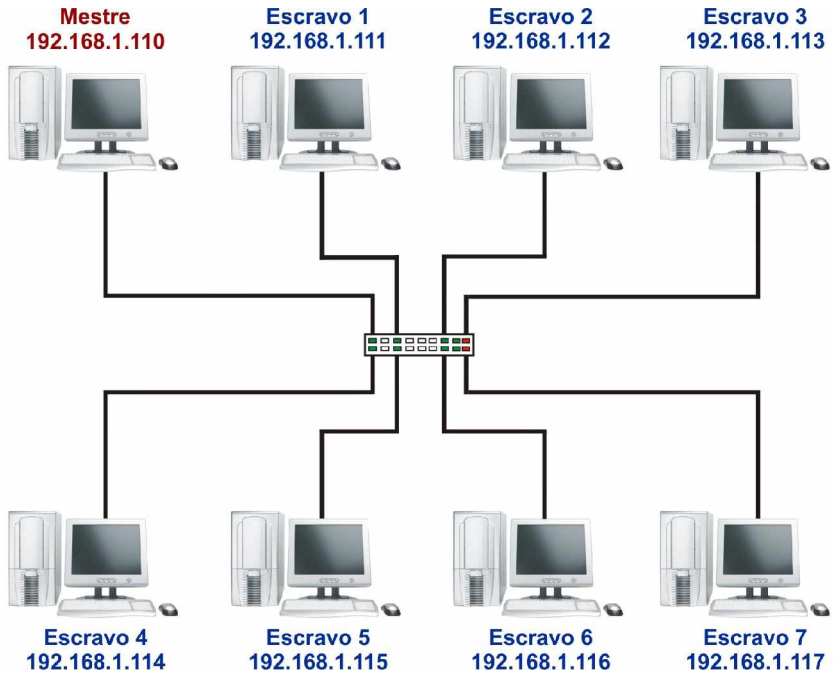

Figura 3.1 – Esquema do *cluster* configurado

3.3.1 Sistema Operacional Linux

Como já colocado anteriormente o *cluster* foi configurado no laboratório de informática do Colégio Objetivo, onde o sistema operacional instalado em todas as máquinas é a distribuição GNU/Linux Fedora Core 5. Porém, pode ser

utilizado outras distribuições Linux disponíveis, como Mandrake, Ubuntu, Suse, entre outras, por exemplo, que podem ser baixadas em [http://www.openaddict.com/download.php].

É importante colocar que para se obter um maior desempenho, o ideal é que sejam instalados somente os serviços necessários para o funcionamento do *cluster* e do sistema operacional, a fim de operar com o mínimo de recursos necessários possível, assim aumentando o seu desempenho.

O processo de instalação do sistema operacional adotado foi por meio de uma unidade de DVD-ROM, através de um modo gráfico, para facilitar a interatividade entre o usuário e o sistema. Detalhes quanto à instalação do Sistema Operacional GNU/Linux Fedora Core 5 podem ser encontrados em [http://jasonnfedora.no-ip.org/manuais/Instalando_Fedora_Core_5.pdf].

3.3.2 Configuração da Rede

Esta Seção apresenta todos os atributos necessários da rede com oito computadores para a configuração do *cluster*, conforme Figura 3.2.

MAQUINAS	IP	HOSTS	DOMAIN
Computador 1	*192.168.1.110*	*Mestre*	*mestre.cluster.br*
Computador 2	*192.168.1.111*	*escravo1*	*escravo1.cluster.br*
Computador 3	*192.168.1.112*	*escravo2*	*escravo2.cluster.br*
Computador 4	*192.168.1.113*	*escravo3*	*escravo3.cluster.br*
Computador 5	*192.168.1.114*	*escravo4*	*escravo4.cluster.br*
Computador 6	*192.168.1.115*	*escravo5*	*escravo5.cluster.br*
Computador 7	*192.168.1.116*	*escravo6*	*escravo6.cluster.br*
Computador 8	*192.168.1.117*	*escravo7*	*escravo7.cluster.br*

Figura 3.2 – Atributos da rede

Vale lembrar que a atribuição desses endereços pode ser feita de forma estática ou dinâmica, via protocolo DHCP[18] (*Dynamic Host Configuration Protocol*), que é um protocolo que evita a necessidade de atribuir manualmente endereços IP. Por se tratar de uma rede de pequeno porte, a atribuição foi realizada de forma estática.

3.3.3 Configuração do *cluster*

As configurações contidas nesta Seção foram feitas em todos nós (mestre e escravos) pertencentes ao *cluster*. Inicialmente deve se configurar a resolução de nomes dos computadores, ou seja, conversão do nome da máquina em um endereço lógico da rede. Como estamos utilizando poucas máquinas na rede vamos utilizar o arquivo *hosts*, que fica localizado no diretório /etc. É importante lembrar que em uma rede *TCP/IP* esta configuração pode ser feita de várias formas, pode-se utilizar o arquivo *hosts* como é o nosso caso, o serviço de resolução de nomes de domínio (DNS[19] - *D*omain *N*ame *S*ystem*)* ou ainda o serviço de NIS[20] *(Network Information Service).* Atenção este arquivo(*hosts*) deve estar presente em todos os computadores que formam o *cluster* e devem ser idênticos. A figura 3.3 apresenta o conteúdo do arquivo:

[18] http://pt.wikipedia.org/wiki/DHCP
[19] http://pt.wikipedia.org/wiki/Dns
[20] http://pt.wikipedia.org/wiki/NIS

127.0.0.1	*Localhost*	*localhost.localdomain*
192.168.1.110	*Mestre*	*mestre.cluster.br*
192.168.1.111	*escravo1*	*escravo1.cluster.br*
192.168.1.112	*escravo2*	*escravo2.cluster.br*
192.168.1.113	*escravo3*	*escravo3.cluster.br*
192.168.1.114	*escravo4*	*escravo4.cluster.br*
192.168.1.115	*escravo5*	*escravo5.cluster.br*
192.168.1.116	*escravo6*	*escravo6.cluster.br*
192.168.1.117	*escravo7*	*escravo7.cluster.br*

Figura 3.3 – Conteúdo do arquivo `/etc/hosts`

Dando continuidade na configuração do *cluster*, deve-se configurar a relação de confiança entre os nós do *cluster*. Para o funcionamento do *cluster* é necessário que exista uma relação de confiança entre seus membros (nós), de forma que os nós possam se comunicar sem o uso de senhas. Isso pode ser feito através de equivalência, não havendo necessidade de autenticação por senha. Do ponto de vista da segurança isto significa um grande risco, mas é extremamente necessário para o funcionamento do RSH[21] (*Remote Shell*), o protocolo de acesso remoto que permite a comunicação dos nós do *cluster*.

Como podemos ver o *cluster* precisa de uma relação de confiança para poder efetuar uma comunicação entre seus nós. Na grande maioria dos *clusters* esta comunicação é feita através do RSH. Mas além do RSH, o SSH[22] (*Secure Shel)l* também permite uma comunicação entre os nós de um *cluster*. Aqui neste trabalho esta comunicação será feita através do protocolo SSH, onde o mesmo

[21] http://en.wikipedia.org/wiki/RSH
[22] http://pt.wikipedia.org/wiki/Ssh

será configurado de forma que os nós do *cluster* possam se comunicar sem o uso de senhas.

Esta configuração será feita conforme as instruções de [MPICH, 2007]. Inicialmente, tem-se duas situações em que o SSH não deve fazer a solicitação de senha para o usuário do *cluster*.

1. Quando abrir conexão de uma máquina para ela mesma;
2. Quando abrir conexão de uma máquina para uma outra (no caso do mestre para o escravo e vice-versa).

Desta forma é necessário gerar um par de chaves (pública e privada) para o usuário que terá permissão de efetuar SSH sem a digitação de senha. Assim será utilizado o comando `ssh-keygen`, conforme mostrado na Figura 3.4. Quando for solicitada a digitação de senha, apenas pressione a tecla ENTER. Observe que na pasta `/home/usermpi/.ssh` surgirá dois arquivos: `id_rsa` (chave privada) e `id_rsa.pub` (chave pública).

```
#cd /home/usermpi

#ssh-keygen -b 1024 -t rsa
```

Figura 3.4 – Gera chaves

Resolvendo a situação 1

Para resolver a situação 1 descrita, é necessário digitar o comando apresentado na Figura 3.5.

```
#cat /home/usermpi/.ssh/id_rsa.pub >>
/home/usermpi/.ssh/authorized_keys
```

Figura 3.5 – Copia chave para `authorized_keys`

O comando da Figura 3.5 adiciona o conteúdo da chave pública (`id_rsa.pub`) no arquivo de chaves autorizadas (`authorized_keys`). Isso permite que aquele usuário (`usermpi`) possa abrir conexão SSH sem ter que informar sua senha.

Agora, altere as permissões sobre o arquivo `authorized_keys`, conforme mostrado na Figura 3.6.

```
#chmod 0600 /home/usermpi/.ssh/authorized_keys
```

Figura 3.6 – Altera permissões

para testar se funcionou, execute o comando da apresentado na Figura 3.7.

```
#ssh usermpi@mestre.cluster.br
```

Figura 3.7 – Teste I

Resolvendo a situação 2

Para resolver a situação 2, é preciso copiar a chave pública para os outros nós do *cluster*, conforme demonstrado na Figura 3.8.

```
#scp /home/usermpi/.ssh/id_rsa.pub
usermpi@escravo1.cluster.br:/tmp/id_rsa.pub-origem-mestre
```

Figura 3.8 – Copia chave

Neste caso, será copiado o arquivo `/home/usermpi/.ssh/id_rsa.pub` para o diretório `/tmp` na *`usermpi@escravo1.cluster.br`* é acrescentando o sufixo origem-mestre para a identificação a origem do arquivo.

Agora, efetue um `ssh` para a máquina de destino, conforme apresentado na Figura 3.9. Certamente, ainda será solicitada uma senha neste momento.

```
#ssh usermpi@escravo1.cluster.br
```

Figura 3.9 – Teste II

Então, execute o comando da Figura 3.10.

```
#cat /tmp/id_rsa.pub-origem-mestre >>
/home/usermpi/.ssh/authorized_keys
```

Figura 3.10 – Copia chave para `authorized_keys`

Isso fará com que a chave pública do usuário, que foi gerada na máquina de origem, seja adicionada ao arquivo `authorized_keys` daquele usuário na máquina de destino.

Agora, altere as permissões sobre o arquivo `authorized_keys`, conforme mostrado na Figura 3.11.

```
#chmod 0600 /home/usuario/.ssh/authorized_keys
```

Figura 3.11 – Altera permissões

Saia da máquina com o comando `exit`, e acesse novamente para testar. Se tudo tiver sido efetuado corretamente, não será mais solicitada a digitação de senha entre o usuário `usermpi` das máquinas `mestre.cluster.br` e `escravo1.cluster.br`.

Uma observação importante, os passos acima devem ser feitos em todas as máquinas do *cluster*, ou seja, cada nó do *cluster* deverá permitir abertura de conexão SSH a partir dele mesmo e de outros nós sem solicitar a digitação de senha. No nosso caso o mestre começa a execução das aplicações, assim o mestre acessa todos os nós escravos sem senha e todos os nós escravos acessam o mestre da mesma forma, portanto aqui um nó escravo não acessa o outra sem a digitação de senha.

3.3.4 Configuração do Nó Mestre

Para que os nós escravos do *cluster* possam executar os programas distribuídos pelo controlador (nó mestre), é necessário que eles consigam acessar o sistema de arquivos do nó mestre de forma transparente, para isso, é necessário

configurar o servidor do sistema de arquivos *NFS*[23] (*Network File System*) e compartilhar o diretório `/home/usermpi`. A configuração do NFS é feita a partir do arquivo "`exports`" localizado no diretório `/etc`, onde o conteúdo é apresentado na Figura 3.12.

```
/home/usermpi *(rw,no_root_squash)
```

Figura 3.12 – Conteúdo do arquivo `etc/exports`

Para cada nó escravo do *cluster* é necessária uma entrada no arquivo `exports`, para o diretório `/home/usermpi`. O parâmetro `rw` significa que os nós podem ler e escrever no sistema de arquivos e, o parâmetro `no_root_squash` habilita privilégios de *superusuário* (`root`). Nos nós escravos devem ser configurados os clientes *NFS*, apresentados na próxima Seção.

3.3.5 Configuração dos Nós Escravos

Nos nós escravos bastar alterar o arquivo `/etc/fstab`, para que no momento da inicialização de cada nó escravo seja montado o diretório `/home/usermpi` compartilhado do nó controlador (nó mestre). Para configurar os clientes, em cada nó escravo é necessário editar o arquivo `/etc/fstab` para montar automaticamente o sistema de arquivos remoto. Adicione as linhas da Figura 3.13 ao final do arquivo `etc/fstab`.

[23] http://pt.wikipedia.org/wiki/Nfs

```
Mestre:/home/usermpi /home/usermpi nfs exec,dev,suid,rw 11
```

Figura 3.13 – Conteúdo do arquivo `etc/fstab`

3.3.6 Configuração e instalação da biblioteca MPI

Depois de concluída as configurações acima deve-se instalar e configurar a biblioteca de troca de mensagens MPI para finalizar a configuração do *cluster*.

Neste trabalho será instalada a implementação do padrão MPI 1 denominada MPICH versão 1. O *download* do arquivo pode ser feito no site: [http://www-unix.mcs.anl.gov/mpi/mpich1/download.html]. Após baixar o arquivo `mpich.tar`, descompacte-o no diretório `/tmp`, com o comando descrito na Figura 3.14.

```
#tar -zxvf mpich.tar
```

Figura 3.14 – Descompactando o arquivo `mpich.tar`

Após a descompactação do arquivo `mpich.tar` , é necessário executar a seqüência de comandos da Figura 3.15 para configuração e instalação da MPICH. Para e execução dos comandos é necessário estar dentro do diretório `/tmp/mpich-1.2.7`. É importante destacar que por `default`, o MPICH versão 1 efetua comunicação por meio do uso de RSH. Aqui neste trabalho esta comunicação será feita através do SSH. Para alterar de RSH para SSH, é preciso informar o parâmetro `-rsh=ssh` na hora de sua configuração, como apresentado na Figura 3.15.

```
#./configure --prefix=/home/usermpi/mpich-1.2.7 -rsh=ssh
#make
#make install
```

Figura 3.15 – Instalando MPI

O MPICH será instalado no diretório `/home/usermpi/mpich-1.2.7` conforme especificado no `--prefix` da Figura 3.15. Com um editor de texto, abra o arquivo `machines.LINUX`, localizado no diretório `/home/usermpi/mpich-1.2.7/share/machines/`, e acrescente ao final, os `hosts` de todos os computadores pertencentes ao *cluster*, conforme a Figura 3.16.

```
mestre
escravo1
escravo2
escravo3
escravo4
escravo5
escravo6
escravo7
```

Figura 3.16 – Conteúdo do arquivo `machines.LINUX`

Finalizando, para que possamos executar os comandos da MPICH a partir de qualquer diretório da máquina, é necessário adicionar o caminho para o mesmo na variável de ambiente PATH. Abra o arquivo `.bashrc` (observe o "." no início do nome do arquivo) localizado no diretório do usuário `home/usermpi` e acrescente a linha descrita na Figura 3.17.

```
Export MPICH=/home/usuario/mpich-1.2.7
```

Figura 3.17 – Conteúdo do arquivo *.bashrc*

Para certificar-se de que o *cluster* está funcionando corretamente é necessário fazer um teste, o MPICH trás consigo alguns exemplos que podem ser localizados no diretório `/home/usermpil/mpich-1.2.7/examples`. Neste trabalho foi escolhido um programa para cálculo de PI paralelizado, denominado CPI, que será utilizado para o nosso primeiro teste básico de passagem de mensagens. Cada computador calcula o valor de pi e compara-o com o valor 3.14159265358979. O programa CPI pode ser encontrado em `/home/usermpi/mpich-1.2.7/examples/.` Compile o programa a linha de comando apresentado na Figra 3.18.

```
#mpicc cpi.c –o cpi
```

Figura 3.18 – Compilando o cpi.c

Após compilar o `cpi.c`, execute-o com todos os nós do *cluster*, ou seja, um para cada nó do *cluster*, conforme descrição da Figura 3.19

```
#mpirun -np 8 cpi
```

Figura 3.19 – executando o `cpi.c`

A saída do programa `cpi.c` deverá ser similar à apresentada na Figura 3.20, logo pode-se concluir que o *cluster* está funcionando perfeitamente.

```
Process 0 on mestre
Process 1 on escravo1
Process 2 on escravo2
Process 3 on escravo3
Process 4 on escravo4
Process 5 on escravo5
Process 6 on escravo6
Process 7 on escravo7
pi is approximately 3.1416009869231241, Error is 0.0000083333333309
wall clock time =0.004784
```

Figura 3.20 – Resultado do `cpi.c`

CAPÍTULO IV

4 TESTES E RESULTADOS COMPUTACIONAIS

Nesta Seção, serão apresentados os testes e resultados computacionais realizados sobre o *cluster*. Primeiramente, será apresentada a aplicação que foi utilizada para teste, bem como sua paralelização e as instâncias utilizadas para teste. Em seguida, o desempenho do algoritmo da aplicação paralela e por fim os resultados obtidos tanto sobre a qualidade das soluções fornecida quanto pelo tempo de execução. Também serão fornecidas comparações com os resultados obtidos.

4.1 APLICAÇÃO UTILIZADA PARA TESTES

A aplicação utilizada foi a paralelização da Metaheurística GRASP utilizando a Técnica *Path-Relinking*, cuja implemetação seqüencial foi realizada no trabalho de Alvarenga e Rocha intitulado de "Melhorando o Desempenho da Metaheurística GRASP Utilizando a Técnica *Path-Relinking:* Uma Aplicação para o Problema da Árvore Geradora de Custo Mínimo com Grupamentos" disponível em [ALVARENGA e ROCHA, 2006]. Esta aplicação foi desenvolvida de forma seqüencial, é um dos objetivos deste trabalho paralelizar esta aplicação com o objetivo de reduzido o seu tempo computacional. Adiante será apresentada as principais características desta aplicação de forma a facilitar o entendimento do leitor.

O problema da Árvore Geradora de custo Mínimo com Grupamento (AGMG) se dá através de um conjunto de vértices do grafo associado particionado em subconjuntos disjuntos, onde a árvore geradora resultante

não deve ligar necessariamente todos os pontos (nós) e sim todos os conjuntos.

A AGMG consiste em um grafo $G = (V, E)$ onde V representa um conjunto de vértices e E é o conjunto de arestas que ligam os vértices de V. O conjunto de vértices V é particionado em k grupamentos, onde $V = V1 \cup V2 \cup ... \cup Vk$. O objetivo da AGMG é ligar pelo menos um nó (vértice) de cada grupamento (conjunto), de modo que a árvore geradora possa ligar todos os grupamentos do grafo com custo mínimo, onde custo significa a soma dos valores de todas as arestas que formam a AGMG.

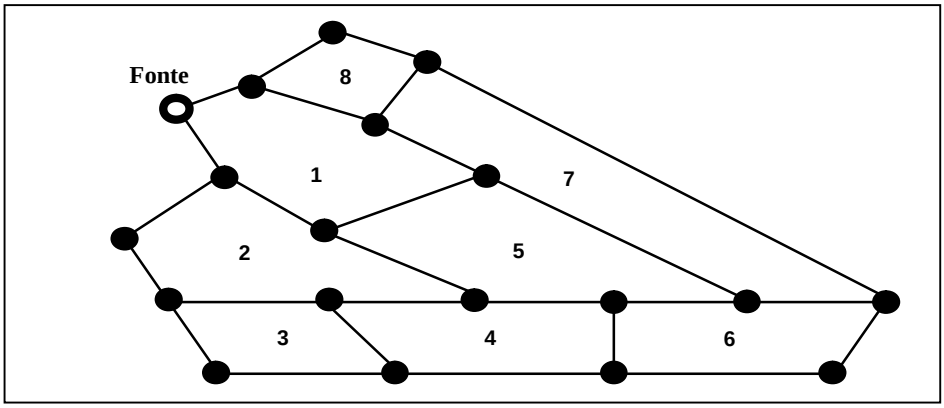

Figura 4.1 - Uma aplicação para o problema da AGMG.

Na Figura 4.1 tem-se um exemplo de uma aplicação para sistemas de irrigação em áreas desérticas, onde tem-se uma área (representada por um grafo) particionada em 8 (oito) regiões (grupamentos), necessitando de irrigação. O problema consiste em criar uma rede de irrigação de menor caminho que possa tocar no mínimo um vértice (reservatório) de cada região, onde o nó fonte é o principal, pois é nele que se localiza a fonte de água. Esta rede [OCHI *et al*., 2003] não pode cruzar as regiões, podendo apenas passar pelas suas fronteiras (arestas). Assim, este problema da irrigação pode ser

modelado com uma AGMG. Considerando um Grafo $G=(V, E)$, os vértices de V representam os vértices das fronteiras das regiões, incluindo a fonte e, o peso de cada aresta de E é associado à distância entre os seus vértices.

Na Figura 4.2 é mostrada uma solução para o sistema de irrigação da Figura 4.1, onde se observa que a Árvore Geradora de custo Mínimo com Grupamentos está representada por linhas mais grossas. Além disso, está partindo do nó fonte e percorrendo todos os grupamentos (regiões) do grafo, ligando pelo menos um vértice de cada grupamento, sem cruzar as fronteiras (arestas) ou formar ciclos.

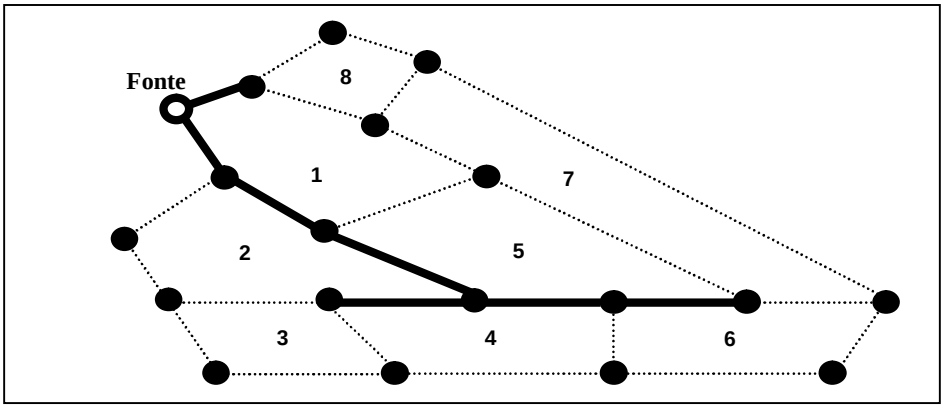

Figura - 4.2 Solução para a Figura 4.1.

Visto o exemplo das Figuras 4.1 e 4.2 do sistema de irrigação, pode-se imaginar a complexidade deste problema, tente imaginar uma área com 50 regiões (grupamentos), 300 reservatórios (vértices) e 5000 caminhos (arestas). Dada esta complexidade, foi necessário utilizar técnicas com Metaheurísticas como a GRASP (*Greedy Random Adaptive Search Procedures*) e técnicas como Path-Relinking, para se obter soluções de qualidade.

Maiores detalhes e informações sobre o problema da Arvore Geradora de Custo Mínimo com Grupamentos, Metaheurísticas, Path-Relinking, GRASP ou qualquer outra informação relacionada a esta aplicação podem ser encontrados em [ALVARENGA e ROCHA, 2006].

4.1.1 Paralelização da aplicação

A complexidade da aplicação e a utilização de instâncias de elevadas dimensões, faz com que o tempo computacional se torne muito grande. Assim faz-se necessária a paralelização de aplicação com intuito de reduzir o tempo computacional.

A paralelização da aplicação foi implementada seguindo o modelo mestre-escravo, utilizando os recursos da biblioteca de troca de mensagens MPI, de acordo com a Figura 4.3 conforme os seguintes passos:

1. O processador mestre recebe o número de iterações passadas pelo usuário, para serem executadas pelo algoritmo.
2. O processador mestre divide o número de iterações pelo número de processadores escravos mais um.
3. O processador mestre e os escravos executam o mesmo trecho de código da heurística GRASP seqüencial, com o número de iterações passado a cada um deles.
4. O processador mestre é então o responsável por receber as soluções dos escravos, o mestre compara todas as soluções incluindo a sua, e posteriormente seleciona-se a melhor entre todas.

Em geral, cada processador executa *MaxIter / p* iterações, onde *MaxIter* e *p* são, respectivamente, o número total de iterações do procedimento GRASP e o número de processadores. Cada processador possui uma cópia do algoritmo

seqüencial (uma descrição detalhada sobre todo o algoritmo seqüencial pode ser encontrada em [ALVARENGA e ROCHA, 2006]), uma cópia dos dados do problema e uma semente independente para gerar sua própria seqüência de números aleatórios. Para evitar que os processadores encontrem a mesma solução, cada um deles usa uma seqüência diferente de números aleatórios. Uma única variável global é necessária para armazenar a melhor solução dentre aquelas encontradas por todos os processadores.

```
Procedimento paralelo
1  f(solucao) ← ∞;
2  inic_paralelismo
3     proc_mestre
4        tarefa ← iteracoes/n_precessador
5        gera_semente;
6        ditribui_dados_escravos(tarefa)
7        execute(GRASP-PR)
8        recebi_dados_escravos(solucao')
9           Se f (solucao') < f (solucao) então
10             solucao ←solucao' ;
11          Fim Se
12    Fim proc_mestre
13    proc_escravo
14       recebe_dados_mestre(tarefa)
15       gera_semente;
16       execute(GRASP-PR)
17       retorne solucao';
18    Fim proc_escravo
19 final_paralelismo
Fim Procedimento
```

Figura - 4.3 Pseudocódigo paralelo

Um dos processadores atua como controlador (nó mestre), lendo e distribuindo os dados do problema, distribuindo as iterações pelos processadores e coletando a melhor solução obtida por cada um deles.

4.2 TESTES

O algoritmo paralelo da aplicação foi submetido às instâncias disponibilizadas por (DROR et al., 2000) para o problema da Árvore Geradora de custo Mínimo com Grupamentos (AGMG).

Tabela 4.1. Instâncias utilizadas para testes.

Instancias	**Nº. vértices**	**Nº. arestas**	**Nº. grupamentos**
Gmst1	25	50	4
Gmst2	25	100	8
Gmst3	25	150	10
Gmst4	50	150	5
Gmst5	50	300	10
Gmst6	75	200	8
Gmst7	75	300	10
Gmst8	75	400	15
Gmst9	100	300	7
Gmst10	100	500	10
Gmst11	150	300	8
Gmst12	150	500	12
Gmst13	200	500	10
Gmst14	200	1000	20
Gmst15	250	500	10
Gmst16	250	1000	25
Gmst17	300	1000	20
Gmst18	300	2000	30
Gmst19	300	3000	40
Gmst20	300	5000	50

Cada uma das 20 instâncias é um grafo conexo e particionado em vários grupamentos, onde: o número de grupamentos varia de 4 a 50; o número de nós varia de 25 a 500; e por fim o número de arestas varia de 50 a 5000. Cada grupamento possui inúmeros vértices conectados entre si, e a cada aresta é atribuído um peso (custo, valor ou distância). Na Tabela 4.1 são apresentadas todas as instâncias e seus parâmetros.

Para a realização dos testes da aplicação no *cluster*, foram utilizados um, dois, quatro e oito processadores para cada uma das 20 instâncias. Os resultados obtidos estão apresentados na próxima Seção.

4.2.1 Solução e tempo de execução da aplicação paralela

O algoritmo paralelo da aplicação foi implementado utilizando a linguagem C e a biblioteca *Message passing Interface* (MPI) de troca de mensagens. Para a realização dos testes, foram utilizados 1, 2, 4 e 8 processadores para cada uma das 20 instâncias. Os resultados obtidos estão apresentados na Tabela 4.5. A utilização de um recurso tão poderoso se deve a complexidade do problema e a utilização de instâncias de elevadas dimensões, fazendo com que o tempo computacional se torne muito grande.

Na execução das 20 instâncias (Gmst1, Gmst2,..., Gmst20) utilizadas para teste, o algoritmo paralelo foi executado utilizando os seguintes parâmetros:

- Alfa: sorteado aleatoriamente entre 0 e 1 a cada iteração do GRASP;
- Critério de para do GRASP: 5000 iterações;
- O *path-relinking* foi aplicado a cada: 200 iterações do GRASP;
- Tamanho do conjunto de elite: 5.

Análise e comparações mais aprofundadas dos resultados obtidos pelo algoritmo paralelo apresentados na Tabela 4.2, serão analisados na próxima Seção.

Tabela 4.2. Desempenho do algoritmo paralelo (em segundos de CPU).

	1 processador		2 processadores		4 processadores		8 processadores	
Instancias	**Solução**	**Tempo**	**Solução**	**Tempo**	**Solução**	**Tempo**	**Solução**	**Tempo**
Gmst1	23	0.61	23	0.32	23	0.16	23	0.11
Gmst2	41	0.70	41	0.39	41	0.25	41	0.13
Gmst3	36	0.77	36	0.41	36	0.26	36	0.12
Gmst4	18	2.07	18	1.03	18	0.60	18	0.34
Gmst5	27	3.17	27	1.53	27	0.88	27	0.41
Gmst6	55	5.86	55	3.03	55	1.51	55	0.79
Gmst7	67	6.03	67	3.11	67	1.53	67	0.81
Gmst8	53	8.01	53	4.05	53	2.01	53	1.11
Gmst9	37	9.11	37	4.76	37	2.51	37	1.43
Gmst10	48	14.21	48	7.19	48	3.67	48	1.97
Gmst11	50	22.53	50	11.28	50	5.69	50	3.17
Gmst12	75	25.89	75	12.79	75	6.53	75	3.67
Gmst13	44	49.11	44	24.47	44	12.69	44	6.35
Gmst14	58	53.07	58	28.64	58	13.93	58	7.19
Gmst15	60	74.84	60	37.40	60	18.43	60	9.79
Gmst16	124	133.71	124	67.99	124	33.99	124	17.32
Gmst17	91	171.62	91	86.87	91	43.76	91	21.98
Gmst18	91	224.59	91	112.99	91	56.40	91	28.63
Gmst19	92	297.46	92	149.61	92	74.49	92	37.88
Gmst20	123	1003.97	123	490.21	123	244.63	123	127.91

4.2.2 Analise e comparação dos resultados da aplicação paralela

Nos gráficos das Figuras 4.4 a 4.5, pode-se observar a redução do tempo computacional para todas as instâncias de testes utilizadas em função do acréscimo de processadores. O tempo computacional (em segundos de CPU) das 10 primeiras instâncias pode ser visto no gráfico da Figura 4.4. Já as instâncias de 11 a 15 podem ser vistas no gráfico da Figura 4.5, as demais de 16 a 20 estão apresentadas no gráfico da Figura 4.6.

De acordo com as Figuras 4.4 a 4.6, pode-se observar que o algoritmo GRASP-PR proposto, foi adequado para paralelização, dado que apresentou uma relação (*speed-up*) linear na redução do tempo seqüencial com o acréscimo de processadores.

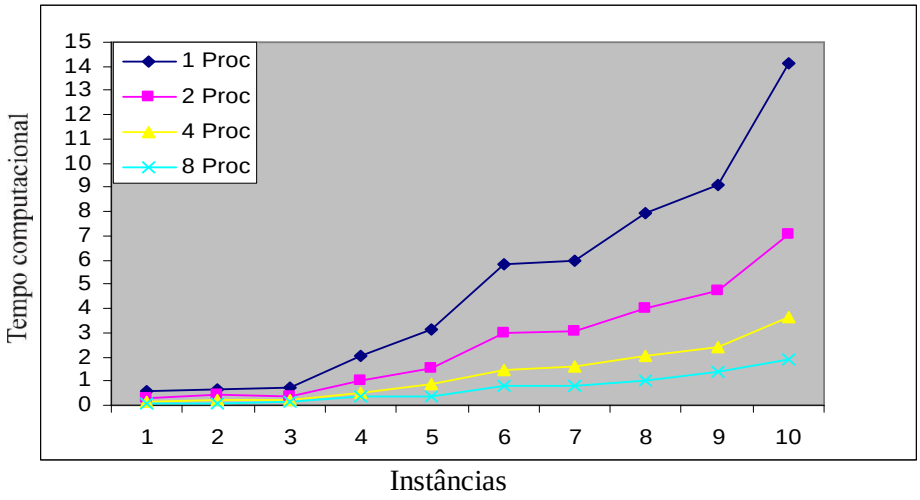

Figura 4.4 – Tempo computacional das instâncias de 1 a 10.

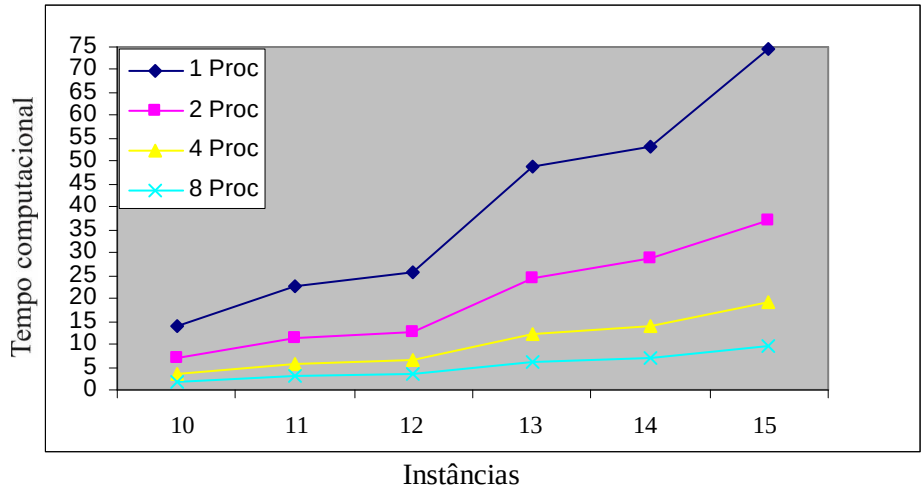

Figura 4.5 – Tempo computacional das instâncias de 10 a 15.

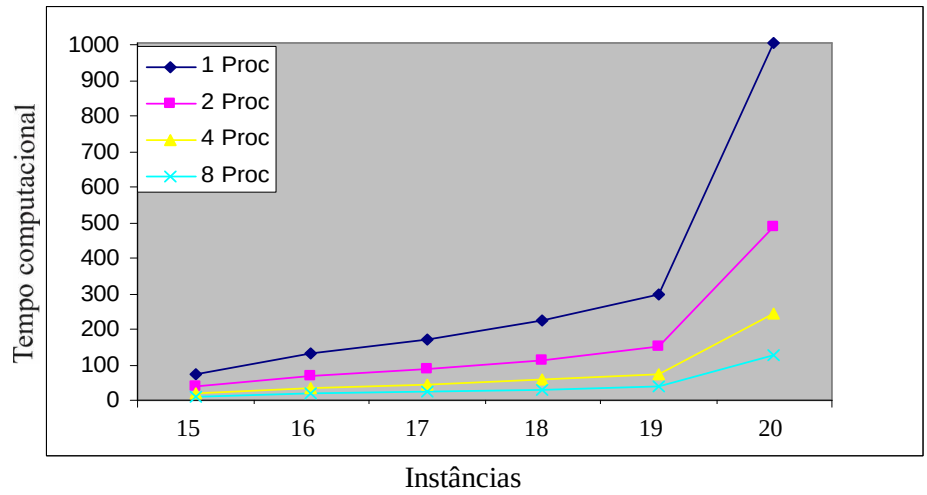

Figura 4.6 – Tempo computacional das instâncias de 15 a 20.

CAPÍTULO V

5 CONCLUSÕES E TRABALHOS FUTUROS

Quando se fala em computação de alto desempenho, logo imaginamos um supercomputador com um custo altíssimo. Mas, atualmente para se ter acesso à computação de alto desempenho como processamento paralelo e distribuído, pode-se utilizar de *clusters* de computadores que têm como "principal" vantagem o custo reduzido quando comparados aos supercomputadores, o que viabiliza o uso do processamento de alto desempenho na solução de problemas em diversas áreas.

Este trabalho teve como objetivo geral desenvolver uma aplicação paralela para solução do problema da Árvore Geradora de custo Mínimo com Grupamentos utilizando-se de *clusters* de Alto Desempenho em Linux, bem como a configuração e testes do *cluster*. Com a utilização de softwares livres como GNU/Linux, bibliotecas para programação paralela, como MPI e um laboratório de informática já existente, este objetivo foi plenamente alcançado com um custo financeiro zero, ou seja, a computação de alto desempenho está cada vez mais acessível em relação ao custo financeiro.

Após a configuração do *cluster* foram realizados vários testes já com a aplicação da área de otimização de [ALVARENGA e ROCHA, 2006], que foi paralelizada neste trabalho. Com os testes realizados sobre a aplicação paralelizada foi possível mostrar o tamanho poder do *cluster* e sua importância nesta área. Foi possível observar que o algoritmo GRASP-PR paralelo proposto, foi adequado para paralelização, dado que apresentou uma relação (*speed-up*) linear na redução do tempo seqüencial com o acréscimo de nós do *cluster*, conforme apresentado na Seção 4.2.2.

Como recomendações para trabalhos futuros, seria interessante a instalação de outras bibliotecas para troca de mensagens como, por exemplo, PVM e fazer uma comparação de desempenho, utilizando para testes aplicações da área de otimização como a AGMG, por exemplo. Além de outras bibliotecas, também seria interessante a configuração de um outro tipo de *cluster*, como *OpenMosix*[24], por exemplo.

[24] http://openmosix.sourceforge.net/

REFERÊNCIAS BIBLIOGRÁFICAS

ALVARENGA, F. V.; ROCHA, M. L. *Melhorando o Desempenho da Metaheurística GRASP Utilizando a Técnica Path-Relinking: Uma Aplicação para o Problema da Árvore Geradora de Custo Mínimo com Grupamentos*. In: XXXVIII Simpósio Brasileiro de Pesquisa Operacional, 2006, Goiânia. XXXVIII SBPO, 2006.

BEOWULF. [on-line]. Disponível na Internet via www. url: http://www.beowulf.org/overview/index.html. Arquivo capturado em Novembro de 2006.

BRAUN, L. R. *Avaliação da implementação de clusters de computadores usando tecnologia Livres*. Novo Hamburgo: 2006. Monografia (Ciência da Computação). Centro Universitário FEEVALE – RS, 2006.

BRFEM Paralelo. *Desenvolvimento de um Framework Orientado para Objetos do Método de Elementos Finitos Usando Algoritmos Paralelos***.** Belo Horizonte: 2006. Relatório (Departamento de Engenharia Elétrica). UFMG – MG, 2006. [on-line]. Disponível na Internet via www. url: http://mico.ead.eee.ufmg.br:3128/~renato/brfem/BRFEMParalelo.pdf. Arquivo capturado em Novembro de 2006.

CLUSTER. [on-line]. Disponível na Internet via www. url: http://pt.wikipedia.org/wiki/Cluster. Arquivo capturado em fevereiro de 2007.

DROR M.; HAOUARI M.; CHAOUACHI J.. *Generalized Spanning Trees*. European Journal of Operational Research, 120, p. 583-592, 2000.

FREITAS, L. E. *Uma comparação entre os modelos de messege passing MPI e PVM*. [on-line]. Disponível na Internet via www. url: http://www.inf.ufrgs.br/procpar/disc/cmp134/trabs/T2/981/mpi.html. Arquivo capturado em novembro de 2006.

GEIST, Al et al. *PVM: parallel virtual machine.* Cambridge, MA: MIT Press, 1994.

INTRODUÇÃO AO MPI. Centro Nacional de Processamento de Alto Desempenho. CENAPAD-SP. [on-line]. Disponível na Internet via

www. url: http://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_MPI.ps.gz. Arquivo capturado em fevereiro de 2007.

JÚNIOR, Esli Pereira Faustino; FREITAS, Reinaldo Borges. *Construindo Supercomputadores com Linux: Cluster Beowulf*. Goiânia: 2005. 104p. Monografia (Tecnólogo em redes de comunicação) – Departamento de Telecomunicações, Cefet-Go, 2005.

LANDEROS, R. D. C. *Clustering Para Procesamiento Matemático*. Temuco: 2006. Trabajo de Titulo. (Ingeniero de Ejecución en Informática). Facultad de Ciencias Escuela de Informática – Chile, 2006.

LINHALIS, F; Fiats, M. I. *PVM e MPI*. São Carlos: 1998. Monografia (Departamento de Ciência da Computação e Estatísticas). Universidade de São Paulo – SP, 1998.

MARTINS, E; Ferreira, G; Mendes, H. M. *Cluster*. Unicamp. Campinas São Paulo. 2005. [on-line]. Disponível na Internet via www. url: http://www.ic.unicamp.br/~rodolfo/Cursos/mc722/2s2005/Trabalho/g04-cluster.pdf. Arquivo capturado em novembro de 2006.

MERKEY, Phil. [on-line]. Disponível na Internet via www. url: http://www.beowulf.org/overview/history.html. Arquivo capturado em janeiro de 2007.

MPI FORUM. *The MPI message passing interface standard***.** Knoxville: University of Tenessee, 1994.

MPI. [on-line]. Disponível na Internet via www. url: http://www.mpi-forum.org. Arquivo capturado em novembro de 2006.

MPICH. [on-line]. Disponível na Internet via www. url: http://www.ucb.br/prg/professores/giovanni/disciplinas/2005-2/pc/mpich1.html. Arquivo capturado em janeiro de 2007.

OCHI L.; BRUNO L.; FERNANDO C.. *Técnicas Para Melhorar o Desempenho de Algoritmos Evolutivos: Uma Aplicação Para o Problema de Árvore Geradora de custo Mínimo Com Grupamentos*. Itajaí: 2003. III Congresso Brasileiro de Computação. Itajaí-SC, p. 661-672, 2003.

OLIVEIRA, D. C. *Desenvolvimento de Clusters de Alto Desempenho.* Palmas: 2004. Monografia (Sistemas de Informação) – Centro Universitário Luterano de Palmas. ULBRA-TO, 2004.

PAIVA, G. F. V.; JUSTO, R. P. *Processamento Paralelo em Ambiente Linux.* Goiânia: 2004. Monografia. (Ciência da computação) – Escola de Engenharia Elétrica e de Computação. UFG – GO, 2004.

PITANGA, Marcos. *Construindo Supercomputadores com Linux*. 2ª edição. Rio de janeiro: Brasport Livros e Multimídia Ltda, 2004.

PITANGA, Marcos, *Construindo Supercomputadores com Linux*, Brasport, Rio de Janeiro , 2002.

RIGONI, E. H; ÁVILA, R. B; BARRETO, M. E; SCHLEMER, E; DEROSE, C; DIVERIO, T. A; NAVAUX, P. O. A. *Introdução à programação em clusters de Alto desempenho*. Porto Alegre: 1999. Relatório de pesquisa (Programa de Pós-Graduação em Computação) UFRGS-RS, 1999.

ROCHA, J. M. G. *Clusters Beowulf: Aspectos de projeto e implementação*. Belém: 2003. Dissertação. (Mestrado em Engenharia Elétrica). – Departamento de Pós Graduação em Engenharia Elétrica. UFPA-PA, 2003.

SANCHES, A. L. G. *Sistema de Comunicação de Alto Desempenho Baseado em Programação Genérica*. Florianópolis: 2003. Dissertação (Ciência da Computação). UFSC – SC, 2003.

SIMÕES, S. N., de SOUZA, S. F., MUNIZ, L., FARDIN Jr., D., de SOUZA, A. F., REIS Junior, N. C., VALLI, A. M. P., CATABRIGA, L.; *Instalação e Configuração de Clusters de Estações de Trabalho: Experiência do Laboratório de Computação de Alto Desempenho do Departamento de Informática da UFES***.** Laboratório de Computação de Alto Desempenho do Departamento de Informática da UFES. 2003.