CENTRO UNIVERSITÁRIO FEEVALE

LEANDRO RODRIGO BRAUN

AVALIAÇÃO DA IMPLEMENTAÇÃO DE CLUSTERS DE COMPUTADORES USANDO TECNOLOGIAS LIVRES

Novo Hamburgo, novembro de 2006.

LEANDRO RODRIGO BRAUN

AVALIAÇÃO DA IMPLEMENTAÇÃO DE CLUSTERS DE COMPUTADORES USANDO TECNOLOGIAS LIVRES

Trabalho de conclusão de curso

Centro Universitário Feevale

Instituto de Ciências Exatas e Tecnológicas

Curso de Ciência da Computação

Professor orientador: Ms.Rodrigo Rafael Villarreal Goulart

Co-orientador: Dr. Leandro Krug Wives

Novo Hamburgo, novembro de 2006.

DEDICATÓRIA

Dedico este trabalho especialmente aos meus pais, que sempre me apoiaram e incentivaram durante toda a minha vida. Tudo o que sou e conquistei até hoje, devo a eles.

AGRADECIMENTOS

A meu orientador, mestre Rodrigo Rafael Villareal Goulart pelo auxilio e orientação nesta longa caminhada de estudos.

Ao professor doutor Leandro Krug Wives, que mesmo não fazendo mais parte do quadro de professores da Feevale, manteve contato e ajuda, sempre que necessário.

Ao professor mestre Reynaldo Novaes pela atenção e empréstimos de materiais.

A minha namorada Ediele, pela compreensão e colaboração. Foram muitos dias que não pude passar com ela em função deste trabalho.

A empresa onde trabalho, pela flexibilidade quanto aos horários alterados em função dos estudos.

Por fim, agradeço a todos, e em especial aos meus pais, pelo apoio constante, que de uma forma ou outra, me ajudaram na conclusão deste extenso estudo.

"Um grande caminho começa com o primeiro passo. O resto fica por conta das circunstâncias" (Autor Desconhecido).

RESUMO

A crescente demanda por mais poder de processamento gera a necessidade de computadores cada vez mais rápidos e eficientes. Em áreas que utilizam elevada taxa de computação, como a visualização científica, a multimídia, servidores *web*, de banco de dados ou de *e-mail*, entre outras, geralmente requerem serviços de processamento rápido, alta disponibilidade e balanceamento de carga. Dentro desse contexto, tornar os serviços críticos disponíveis ou processar o mais rápido possível é essencial para o bom funcionamento da empresa ou negócio. Com o aperfeiçoamento das redes de computadores, surge um novo paradigma computacional: os *clusters* de computadores. Entende-se por *cluster* um sistema composto de dois ou mais computadores que trabalham em conjunto, executando tarefas ou aplicações de maneira que os usuários que utilizam o sistema tenham a impressão de que ele é único. Tal sistema é extremamente poderoso, capaz de fazer a distribuição do processamento entre computadores comuns e distintos, interligados por uma rede, sendo capaz de repartir e especializar as tarefas computacionais conforme a natureza da função de cada computador. A utilização de *cluster* de computadores aparece então como uma forma relativamente simples e barata para suprir essa demanda, que até então somente supercomputadores seriam capazes de resolver. Sendo assim, este trabalho tem como objetivo fazer um levantamento das tecnologias atuais existentes na área da computação paralela e distribuída, apresentando estudos de casos práticos sobre a implementação de *clusters* de alto desempenho para serviços que requerem grande capacidade de processamento, analisando a performance utilizando diferentes equipamentos com o uso de *software* livre, mais especificamente com sistema operacional (SO) GNU/Linux e ferramentas de *benchmarks* (avaliação).

Palavras chave: Sistemas Operacionais, *Clusters* de Computadores, Sistemas Distribuídos e Paralelos, Balanceamento de Carga.

ABSTRACT

Title: Evaluation of the implementation of computer clusters using open source technologies.

The increasing demand for more computer power introduces the necessity for faster and more efficient computers. In areas that uses high tax of computation, such as scientific visualization, multimedia, web services and data bases, generally require services of fast or parallel processing, high availability and load balancing. In this context, the availability and the performace of the critical services are essential features for the good functioning of the company or business. With the development of computer networks a new computational paradigm appears: the computer clusters. A cluster can be understood as a system composed of two or more computers working together, executing tasks or applications as unique entity. Those systems are extremely powerful, capable to distribute the load processing among distinct computers, linked by a simple network, and being able to distribute and to specialize the computational tasks taking in account the nature or the function of each computer. The use of a cluster appears as one form relatively simple and cheap to accomplish this demand, which until then could only be performed by large and expensive supercomputers. Thus, the goal of this work is to develop a survey of the current technologies on parallel and distributed computing, showing practical cases of high performace and open source (GNU/Linux) based clusters implementations for CPU-bound problems, analyzing its performace on different equipments.

Key words: Operating systems, Distributed and parallel computing, Clusters, Load balancing.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ATM	***A**synchronous **T**ransfer **M**ode*
BPS	***B**it* **p**or **S**egundo
CC-NUMA	***C**ache Coherent **N**on **U**niform **M**emory **A**ccess*
CI	**C**ircuito **I**ntegrado
Clumps	***C**luster of SMPs*
CMS	***C**luster **M**anagement **S**oftware*
COMA	***C**ache-**O**nly **M**emory **A**rchitecture*
COP	***C**luster **o**f **P**Cs*
COW	***C**luster **o**f **W**orkstations*
CUMULVS	***C**ollaborative **U**ser **M**igration, **U**ser **L**ibrary for **V**isualization and **S**teering*
DNS	***D**omain **N**ame **S**ystem*
HA	***H**igh **A**vailability*
HASP	***H**ouston **A**utomatic **S**pooling **P**riori*
HD	***H**ard **D**isk*
HPC	***H**igh **P**erformance **C**omputing*
HTTP	***H**yper **T**ext **T**ransfer **P**rotocol*
IBM	***I**nternational **B**usiness **M**achines*
IEEE	***I**nstitute of **E**lectrical and **E**lectronic **E**ngineers*
JES	***J**ob **E**ntry **S**ystem*
LSI	***L**arge **S**cale of **I**ntegration*
MB	***M**ega**B**yte*
MD	***M**ultiple **D**ata*

MHZ	***Mega**hertz*
MI	***Multiple** Instruction*
MIMD	***M**ultiple **I**nstruction **M**ultiple **D**ata*
MISD	***M**ultiple **I**nstruction **S**ingle **D**ata*
MOSIX	***M**ulticomputer **O**perating **S**ystem Un**ix***
MPMD	***M**ultiple **P**rogram **M**ultiple **D**ata*
MPI	***M**essage **P**assing **I**nterface*
MPP	***M**assively **P**arallel **P**rocessors*
NASA-GSFC	*National **A**eronautics and **S**pace **A**dministration, no **G**oddard **S**pace **F**light **C**enter*
NCC-NUMA	***N**on **C**ache **C**oherent **N**on **U**niform **M**emory **A**ccess*
NORMA	***N**om-**R**emote **M**emory **A**ccess*
NOW	***N**etwork **o**f **W**orkstations*
NUMA	***N**on **U**niform **M**emory **A**ccess*
PC	***P**ersonal **C**omputer*
PoP	***P**ile **o**f **P**cs*
PPE	***P**arallel **P**rogramming **E**nvironment*
PUC-RJ	**P**ontifícia **U**niversidade **C**atólica do **R**io de **J**aneiro
PVM	***P**arallel **V**irtual **M**achine*
RAM	*Random Access Memory*
SD	***S**ingle **D**ata*
SCI	***S**calable **C**oherent **I**nterface*
SC-NUMA	***S**oftware **C**oherente **N**on **U**niform **M**emory **A**ccess*
SI	***S**ingle **I**nstruction*
SIMD	***S**ingle **I**nstruction **M**ultiple **D**ata*
SISD	***S**ingle **I**nstruction **S**ingle **D**ata*
SMP	***S**ymmetric **M**ulti**p**rocessors*
SO	***S**istema **O**peracional*
SSI	***S**ingle **S**ystem **I**mage*
TFCC	***T**ask **F**orce on **C**luster **C**omputing*
TI	**T**ecnologia da **I**nformação
UFRGS	**U**niversidade **F**ederal do **R**io **G**rande do **S**ul
UFRJ	**U**niversidade **F**ederal do **R**io de **J**aneiro

UMA	***U**niform **M**emory **A**ccess*
ULSI	***U**ltra **L**arge **S**cale of **I**ntegration*
USP	**U**niversidade de **S**ão **P**aulo
VLSI	*Very **L**arge **S**cale of **I**ntegration*

SUMÁRIO

INTRODUÇÃO

As aplicações utilizadas nas empresas atualmente exigem cada vez mais poder de processamento, pois processar rápido e eficientemente é um recurso indispensável para o bom andamento do negócio. Tarefas que exigem grande poder computacional, até pouco tempo atrás, só podiam ser executadas por supercomputadores. No entanto, nos últimos anos, a Tecnologia da Informação (TI) evoluiu rapidamente fazendo com que as organizações usufruíssem dessa evolução. Com o avanço tecnológico dos computadores pessoais e das estruturas de redes locais, cada vez mais velozes, surge a idéia de utilizar o poder individual destes computadores, conectando-os através de um meio físico de comunicação e dotados de *softwares* específicos para executarem aplicações paralelas e distribuídas, dando ao usuário transparência quanto ao seu funcionamento. Assim, quando se utiliza dois ou mais computadores em conjunto para resolver um determinado problema, temos o que chamamos de *cluster*, que, do inglês, significa agrupamento, aglomerado ou concentração (PITANGA, 2004).

De uma maneira geral, os *clusters* podem ser formados por computadores dedicados ou agrupados fisicamente em um ambiente. Outra maneira de formação é através de uma conexão virtual de computadores espalhados em uma rede. Neste método, *softwares* apropriados auxiliam as estações distribuídas ao longo da rede para que possam servir a solicitações de terceiros como seus recursos computacionais (DANTAS, 2006). Esses *softwares* foram desenvolvidos como uma maneira de processar uma carga maior de dados, fazendo um controle sobre as distribuições de tarefas entre os computadores. Conforme

Bookman (2003), o processo de dividir tarefas entre diversas unidades de processamento, não apenas melhora o desempenho, como pode criar redundância em caso de falha.

Assim, podemos dividir os *clusters* em duas categorias básicas: Alta Disponibilidade (HA – *High Availability*) e Alta Performance de Computação (HPC – *High Performace Computing*).

Segundo Alecrim (2004), quando se fala de disponibilidade, pensa-se no tempo em que determinado sistema permanece ativo e em que condições de uso. A alta disponibilidade se refere a sistemas que praticamente não param de funcionar. Usados em aplicações de missão crítica, eles costumam ter meios eficientes de proteção e de detecção de falhas. Já alta performance refere-se à distribuição equilibrada de processamento aos nós (computadores) do *cluster*. É muito usado na Internet, em servidores *web* e de *e-mail*, comércio eletrônico e em bancos de dados distribuídos.

Nesse contexto, os *clusters* podem ser usados em uma infinidade de aplicações, basicamente em qualquer uma que exija 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).

A utilização de *clusters* de computadores apresenta vantagens competitivas em relação aos ambientes multiprocessados de memória[1] compartilhada (computadores que possuem diversos processadores em uma única placa-mãe). Dentre algumas vantagens, estão as facilidades em obter equipamentos e manutenção, além alta escalabilidade, onde é possível acrescentar mais equipamentos ao ambiente sem exigir modificações no *software* executado.

O estudo da computação de alto desempenho sempre esteve associado a centros de pesquisas e universidades. No Brasil, algumas poucas instituições contam com plataforma que suportam um grande volume de processamento. Entre elas, destacam-se Universidade Federal

do Rio Grande do Sul (UFRGS), a Universidade de São Paulo (USP), Universidade Federal do Rio de Janeiro (UFRJ) e Pontifícia Universidade Católica do Rio de Janeiro (PUC-RJ), dentre outras.

De uma maneira geral, a computação baseada em *cluster* já gerou o desenvolvimento de diversos ambientes de programação, com inúmeros testes de interconexão entre os nós, inclusive com a reimplementação de bibliotecas já existentes. Adicionalmente, foram estabelecidos diversos grupos de estudos ao redor do mundo, e conseqüentemente, muitos artigos foram elaborados. O ponto negativo nesse sentido é que todo este conhecimento adquirido está espalhado por vários tipos de fontes, dificultando geralmente, um entendimento maior e atualizado dentro deste contexto.

Baseando-se nisso e sabendo que as necessidades de investimentos tecnológicos do mercado geralmente são altos, este trabalho tem como objetivo fazer uma avaliação da implementação de *cluster* de alto desempenho em processos *CPU-bound*[2], utilizando tecnologias livres da plataforma GNU/Linux e ferramentas de *benchmarking*, apresentando as vantagens, desvantagens e viabilidades de implantar este serviço, mostrando como é possível ter soluções de baixo custo (se comparados com os preços de supercomputadores) para a computação de alto desempenho. Como conseqüência, contribuir para o desenvolvimento de novas tecnologias e simulações que precisam de alta velocidade de processamento. Todas as etapas serão demonstradas e explicadas nesse processo de construção do *cluster*.

Como objetivos específicos, destacam-se:

- Apresentar os diferentes tipos de *clusters*;
- Especificar *softwares* que trabalham em cima dessa tecnologia;

---

[1] Em termos computacionais, a memória é a parte do computador onde os programas e os dados são armazenados.

[2] Entende-se por *CPU-Bound* rotinas onde é necessário um tempo de execução longo e com um número baixo de comunicação entre processos, como aplicações científicas e matemáticas, processos em que exigem uma demanda alta de processamento, etc.

- Contribuir para responder perguntas como "quando" e "onde" deve ser aplicado este tipo de tecnologia;

- Apresentar as soluções *open-source* [3]desta tecnologia;

- Utilização computadores de arquitetura x86 em experimentos.

- Instalar, configurar e avaliar um *cluster* de balanceamento de carga para serviços que requerem computação intensiva de processamento com diferentes configurações de equipamentos.

Desta forma, este trabalho está organizado em sete capítulos estruturado da seguinte maneira. O primeiro capítulo constitui em um levantamento histórico da evolução das arquiteturas de computadores até o surgimento dos supercomputadores atuais. No segundo capítulo será visto os fundamentos da computação paralela segundo os diferentes níveis hierárquicos de um sistema de computação na atualidade. Características dos sistemas distribuídos, bem como os processos de comunicação destes ambientes são encontrados no terceiro capítulo. O quarto capítulo dá uma atenção especial às configurações dos *clusters* computacionais, com ênfase especial em suas características. Dar uma base para um entendimento sobre balanceamento de carga é o objetivo do capítulo cinco. O sexto capítulo está reservado para os meios nos quais podem ser feitas avaliações de performance em agrupamentos de computadores, além da descrição de algumas ferramentas de *benchmarks*. Na seqüência, temos no capítulo sete a descrição de experimentos práticos realizados sobre a implementação de um *cluster Openmosix* no Centro Universitário Feevale, na qual são relatados diversos testes realizados no ambiente montado, e por fim, as considerações finais a respeito deste trabalho.

---

[3] Um *software* denominado *open-source*, ou em português código aberto, é um tipo de *software* cujo código fonte é público.

1 SUPER COMPUTAÇÃO

Quando olhamos a rápida evolução, a quantidade de computadores disponíveis, a variedade de modelos e características existentes, logo relacionamos estes dispositivos a novas tecnologias existentes. É claro que, as tecnologias empregadas na construção dos computadores evoluíram, mas os conceitos básicos descendem da década de 40. Assim, este capítulo apresenta uma breve descrição da evolução da arquitetura dos computadores [4] pessoais e supercomputadores até os dias atuais.

1.1 A revolução do Computador

A primeira geração de computadores, na década de 1940 estabeleceu os conceitos básicos da organização dos computadores eletrônicos. Os computadores desta época utilizavam válvulas eletrônicas e uma quantidade enorme de fios. Eram muito grandes e lentos, além de esquentarem muito. Estes computadores possuíam somente dois níveis de linguagem de programação: o nível da linguagem de máquina, onde toda a programação era feita, através de zeros e uns, e o nível da lógica digital, local na qual os programas eram

[4] "O termo arquitetura de computador vem da possibilidade de se visualizar uma máquina como um conjunto hierárquico de níveis que permite entender como os computadores estão organizados" (BRASIL ESCOLA, 2006, p. 2).

efetivamente executados, sendo formado nesta geração por válvulas e posteriormente por transistores, circuitos integrados [5](CI), além do aperfeiçoamento tecnológico da arquitetura como um todo.

Em 1946, Von Neumann sugeriu um sistema binário, que viria a se tornar à base de outros projetos de desenvolvimento computacional subseqüente. A máquina proposta por Von Neumann era composta por uma unidade de processamento centralizado, responsável pela execução de operações aritméticas e lógicas, uma unidade de controle de programa, na qual determinava quais ações executar, uma unidade de memória principal e uma unidade de entrada e saída de dados (WEBER, 2000). De acordo com Pitanga (2004, p. 01), "o modelo de Von Neumann é composto basicamente por um processador, memória e dispositivos de entrada e saída de dados ou periféricos. O processador executa as instruções seqüencialmente, de acordo com a ordem estabelecida por uma unidade de controle". O modelo lógico de computador proposto por Von Neumann pode ser descrito pela Figura 1.1 abaixo:

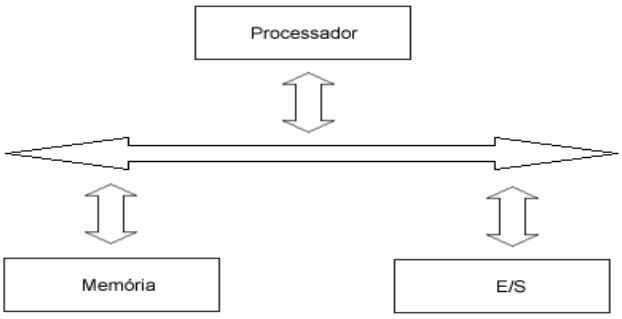

Figura 1.1: Modelo de Von Neumann
Fonte: PITANGA, 2004, p. 1.

A segunda geração de computadores (1956 – 1963) ficou marcada pela invenção do transistor que veio a substituir as válvulas eletrônicas e a troca dos fios de ligações por circuitos impressos. Com estas trocas, os computadores não só ficaram mais rápidos, mas tiveram também uma redução considerável no tamanho dos equipamentos, nos custos e no consumo de energia. Nesse período surgem também alguns dispositivos, como impressoras, fitas magnéticas e discos de armazenamento.

[5] Associação de transistores em pequena placa de silício.

A invenção dos CI proporcionou mais um grande avanço na evolução da computação e com eles surgem os computadores de terceira geração (1964 – 1970). Os transistores foram substituídos por esta nova tecnologia e sua utilização tornou os equipamentos mais compactos, rápidos e confiáveis, com baixo consumo de energia e, um custo ainda menor que os das gerações anteriores. Nesta geração, também foram desenvolvidas as linguagens de programação de alto nível orientada a procedimentos.

O período decorrente de 1970 até os dias de hoje, ficou caracterizado pelo aperfeiçoamento das tecnologias já existentes, denominando-se os computadores da quarta geração. Tecnologias de integração dos CI foram sendo aperfeiçoadas, abrigando milhões de componentes eletrônicos em um pequeno espaço ou *chip*, denominados por escala de integração: Escala de integração grande (LSI - *Large Scale of Integration*), escala de integração muito grande (VLSI - *Very Large Scale of Integration*), e escala de integração ultragrande (ULSI - *Ultra Large Scale of Integration*). Assim, avanços significativos com a otimização dos componentes e o surgimento dos microprocessadores e microcomputadores fizeram com que houvesse uma explosão no mercado de computadores a partir de 1980, sendo fabricados em escala comercial, uma vez que, computadores anteriores a esta geração eram restritos a poucas empresas e universidades, devido a custos e tamanho dos equipamentos. A partir daí, os avanços tecnológicos vem sendo cada vez maior até chegar nos micros atuais, com velocidades cada vez maiores e com custos muito mais baixo, se comparados com as tecnologias anteriores (PITANGA, 2003).

Sendo assim, a indústria da computação é uma das que mais cresce e é abastecida pelos desenvolvimentos tecnológicos rápidos na área de *software* e *hardware*. Os avanços de tecnologias de *hardware* incluem o desenvolvimento e fabricação de *chips* e processadores rápidos e baratos, com largura de banda elevada e baixa latência de interconexão. Destacam-se recentemente avanços na tecnologia VLSI, que deram um dos principais passos no desenvolvimento de computadores seqüenciais e paralelos. Assim como o desenvolvimento de *hardware* teve grande evolução, o desenvolvimento de *software* também evoluiu rapidamente. *Softwares* maduros como sistemas operacionais, linguagens de programação e ferramentas de desenvolvimento estão disponíveis no mercado a fim de suprir as necessidades comerciais (BUYYA, 1999). Uma revisão das mudanças na era da computação pode ser vista na Figura 1.2. Cada era da computação ficou caracterizada pelo inicio do desenvolvimento de

arquiteturas de *hardware*, seguido pelo desenvolvimento de sistemas operacionais e compiladores, até alcançar seu crescimento comercial.

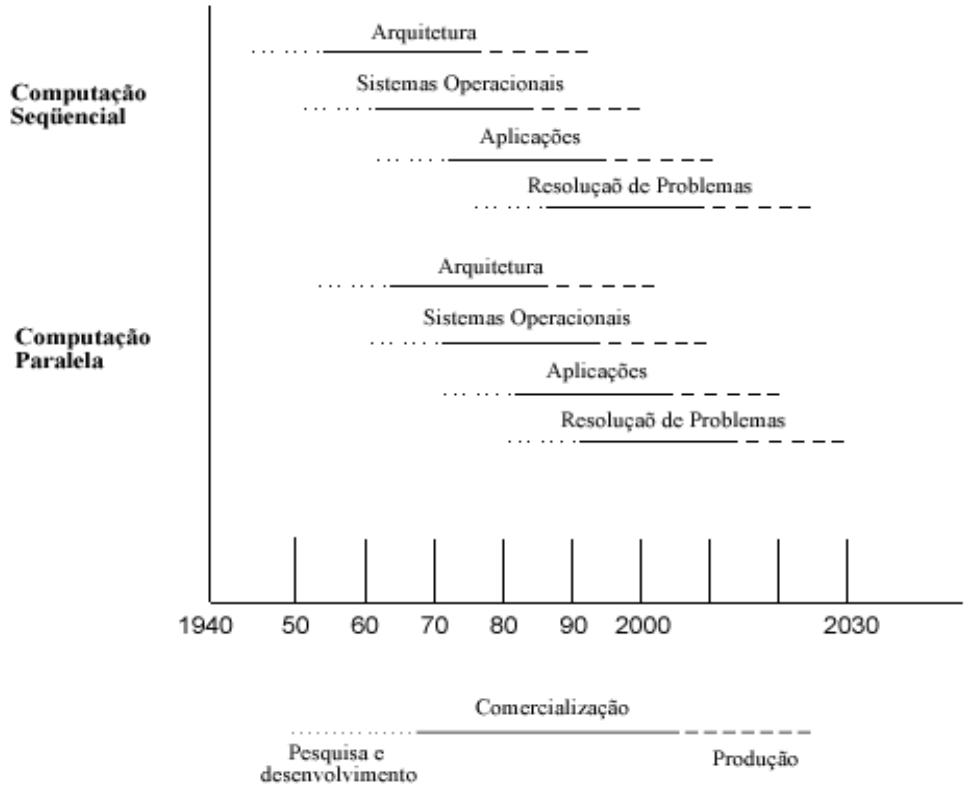

Figura 1.2: Duas eras da computação
Fonte: BUYYA, 1999, p. 5 (Tradução Nossa).

Na medida em que o tempo foi passando, novas necessidades foram surgindo e a utilização de simples computadores já não era suficiente. Dentro deste contexto, novos estudos foram sendo realizados para aprimorar a capacidade de processamento das máquinas, dando origem aos supercomputadores.

1.2 Supercomputadores

Os supercomputadores são máquinas de grande porte, capazes de processar grandes quantidades de informação a uma velocidade bastante elevada. Possuem grande capacidade de memória e são muito empregados em pesquisas científicas e militares, em empresas de altíssima tecnologia, na produção de efeitos e imagens computadorizadas de alta qualidade, entre outras. De uma maneira geral, os supercomputadores possuem diversos processadores, cada qual podendo executar uma operação em particular ou em paralelo, de acordo com o tipo do processo a ser executado (TANENBAUM, 2001). O que diferencia dos demais computadores é que vários processadores e módulos de memória são combinados para criar um sistema com grandes capacidades de processamento.

Os primeiros supercomputadores surgiram de fato nos anos 70. Máquinas estas que inovaram em arquitetura. Até aquele momento, as arquiteturas eram simples e funcionavam muito bem. No entanto, o aumento da eficiência computacional estava limitado pelo desenvolvimento tecnológico, principalmente pelo fato dos processadores terem que terminar uma tarefa antes de iniciar outra. Dessa forma, percebeu-se que a divisão de tarefas traria avanços significativos quanto ao desempenho das máquinas, surgindo a partir desta época duas novas áreas: Arquiteturas paralelas e sistemas distribuídos (PITANGA, 2004). Nos próximos capítulos serão abortados estes tipos de tecnologias.

Em 1975, nasce o Illiac IV, dotado de 64 processadores, sendo o computador mais rápido até o surgimento do supercomputador Cray-1, em 1976, fabricado pela Seymor Cray. Esta foi à primeira máquina *pipeline*[6], cujo processador executava uma instrução dividindo-a em partes, como na linha de produção dos automóveis, ou seja, enquanto a segunda parte de uma instrução estava sendo processada, a primeira parte de outra instrução era inicializada.

[6] É uma técnica de *hardware* que permite que o processador realize a busca de uma ou mais instruções além da próxima a ser executada.

O primeiro supercomputador da linha Cray, denominado Cray-1 era capaz de processar até 133 *mega flops*[7]. Seu sucessor, em 1985, o Cray-2, tinha capacidade de desempenho de 1,9 *giga flops*, tendo na época a maior memória do mundo: 2 *gigabytes*. Atualmente, os supercomputadores estão com capacidades altíssimas de poder de processamento. Para se ter uma idéia, o supercomputador mais rápido na atualidade opera a uma capacidade de 280.6 *tera flops*, tendo 131072 processadores. Foi fabricado pela *International Business Machines* (IBM) e inaugurado em 2005. Está localizado na cidade de Livermore (USA) e é utilizado em simulações climatológicas, sendo capaz de prever, por exemplo, terremotos com uma grande antecedência de dias. Hoje os supercomputadores são fabricados por empresas como IBM, DELL, CRAY etc. A Tabela 1.1 a seguir mostra a relação dos dez mais rápidos supercomputadores do mundo até 2005. A lista atualizada pode ser obtida através do *site* <www.top500.org>.

Tabela 1.1: Relação dos 10 maiores supercomputadores do mundo

	Fab.	Computador	Tf/s	Local de Instalação	País	Ano	Processadores
1	IBM	BlueGene/L eServer Blue Gene	280.6	Doe/NNSA/LLNL	USA	2005	131072
2	IBM	BGW eServer Blue Gene	91.29	IBM THOMAS WATSON	USA	2005	40960
3	IBM	ASC Purple eServer pSeries p575	63.39	Doe/NNSA/LLNL	USA	2005	10240
4	SGI	Columbia Atrix, Infiniband	51.87	Nasa Ames	USA	2004	10160
5	DELL	Thunderbird	38.27	Sandia	USA	2005	8000
6	CRAY	Red Storm Cray XT3	36.19	Sandia	USA	2005	10880
7	NEC	Earth Simulator	35.86	Earth Simulator Center	Japan	2002	5120
8	IBM	Mare Nostrum BladeCenter JS20, Myrinet	27.91	Barcelona Supercomputer Center	Spain	2005	4800
9	IBM	L eServer Blue Gene	27.45	ASTRON Uiversity Groningen	Nether lands	2005	12266
10	CRAY	Jaguar Cray XT3	20.53	Oak Ridge National Lab	USA	2005	5200

Fonte: TOP500, 2005, p. 2.

Uma idéia de tão grande foi à evolução da computação pode ser comparada entre os supercomputadores mais antigos em relação aos PCs atuais, onde hoje em dia, um simples computador pessoal oferece maior capacidade de processamento do que um supercomputador de alguns anos atrás. De uma maneira geral, os computadores estão se tornando cada vez mais rápidos e eficientes, sendo que, equipamentos que atualmente são considerados supercomputadores podem perder seu lugar em um tempo relativamente pequeno.

[7] É uma medida de desempenho que indica o número de instruções de ponto flutuante que um computador é capaz de executar por segundo.

As tendências atuais, tanto em *hardware* como em *software*, estão voltadas em torno de aplicações distribuídas, não somente em um ambiente local, mas sim em âmbito global, na qual sugerem um futuro onde o paralelismo faça parte não somente de supercomputadores e *clusters*, mas sim, da área computacional como um todo. A seguir, veremos maiores detalhes em relação ao paralelismo.

2 ARQUITETURAS PARALELAS

Atualmente, grande parte dos computadores ainda descende da arquitetura de Von Neumann, ou arquitetura serial, pois empregam um único processador. Essa arquitetura, aliada aos avanços da microeletrônica, ofertou-nos o atual mercado de computadores, rápidos e baratos. No entanto, o aumento da capacidade computacional dos dispositivos, depende diretamente ao desenvolvimento tecnológico e enfrenta um limite de velocidade, que é ditado pelas leis da física. Segundo Zelenovsky e Mendonça (2001, p. 1), "o tempo que um sinal elétrico gasta para trafegar entre dois pontos de um circuito eletrônico é muito pequeno, porém não é igual a zero". Em outras palavras, isto corresponde a dizer que existe um limite para a velocidade de execução das instruções em um processador. Assim, surgem questões de como continuar a evolução dos computadores? Ou ainda, como superar está limitação para melhorar o desempenho dos processadores e outros componentes de modo que se possa oferecer maior capacidade computacional requerida pelas aplicações.

Buyya (1999) cita três maneiras de melhorar o desempenho computacional fazendo uma analogia ao mundo real. São elas: trabalhar muito, utilizar boas estratégias e obter ajuda. Relacionando isto ao mundo computacional, trabalhar muito seria como usar ferramentas mais rápidas (processadores ou periféricos com alto desempenho, por exemplo). Utilizar boas estratégias está diretamente relacionado em torno do desenvolvimento de algoritmos e técnicas computacionais mais eficientes e otimizadas, ou seja, muitas vezes uma simples modificação no modo de resolver um problema pode fazer com que uma solução seja

encontrada com maior velocidade. Por fim, obter ajuda consiste na divisão das tarefas [8]por múltiplos processadores a fim de resolver uma atividade em particular, o que nos garante um maior número de instruções processadas por ciclo de *clock*[9].

Outra comparação entre o mundo real e o computacional é citada por Zelenovsky e Mendonça, 2001. Segundo eles, uma comparação entre o cérebro humano e um processador pode nos dar uma noção de como obter maior velocidade de processamento nas máquinas. "É sabido que o sinal elétrico trafegando por dentro de um CI é muito mais veloz que o trânsito de impulsos nervosos entre nossos neurônios" (ZELENOVSKY, MENONÇA, p. 1). Logo, o computador é muito mais rápido para executar tarefas, como operações numéricas, classificações e comparações. No entanto, eles apenas seguem uma seqüência de processos programados, sendo muito inferiores ao cérebro, pois não tem a capacidade de pensar, inovar e aprender. Por meio desta comparação, surge uma grande dúvida: Como será que o cérebro supera os processadores, sendo que a comunicação entre os circuitos eletrônicos é muito maior que a comunicação entre os nossos neurônios? A resposta é simples e obvia. O nosso cérebro opera com bilhões de neurônios em paralelo. Partindo desta premissa, chega-se a idéia básica do processamento computacional em paralelo, ou seja, fazendo a divisão de tarefas entre vários processadores para resolver um determinado problema.

Sabendo-se que a utilização de vários processadores pode melhorar o desempenho computacional, surgem novas questões a partir deste contexto. Como ter um controle destes processos de forma que o resultado seja o desejado? A fim de responder esta questão, alguns problemas existentes em computação paralela devem ser compreendidos. Para isso, vamos citar outro exemplo ligado ao mundo real, seguindo a lógica de raciocínio de Zelenovsky e Mendonça, (2001). Considerando que uma pessoa leve vinte dias para construir um determinado muro, logo duas pessoas levariam metade do tempo, ou seja, terminariam a mesma obra em dez dias. Seguindo este raciocínio, poderíamos afirmar ainda que o acréscimo de mais pessoas pode reduzir cada vez mais o tempo de conclusão da obra. Este é o conceito básico do processamento paralelo: a divisão de tarefas.

---

[8] Em termos computacionais, uma tarefa (task) consiste em uma unidade de processamento indivisível, caracterizada por seu comportamento externo: entradas, saídas, instruções e tempo de execução, com o objetivo de cumprir uma determinada função. Um conjunto de tarefas constitui um processo.

[9] Ciclo de *Clock*: Consiste nos intervalos de tempo que o processador usa para executar suas instruções.

No entanto, devemos lembrar que o acréscimo de pessoas para a execução da obra só será eficiente se as mesmas estiverem em perfeito equilíbrio e sincronismo, ou seja, há um limite a ser respeitado: todas as pessoas devem ter a mesma carga de trabalho, que em termos técnicos é denominado pela expressão "Balanceamento de Carga de Trabalho". O balanceamento de carga pode ser dividido em dois meios de execução: A divisão idêntica das tarefas entre as pessoas, onde cada tarefa é executada em partes iguais por cada uma, e por meio da especialização de cada pessoa, ou seja, enquanto umas fazem um determinado serviço, outros fazem à outra parte da obra, e assim por diante.

Agora, supondo que todos os procedimentos fossem executados ao mesmo tempo. Imagine por exemplo, 10 pessoas assentando o mesmo tijolo. Teríamos um grande problema de alocação de espaço entre elas. Assim, devemos respeitar um limite de pessoas que podem trabalhar na obra em paralelo, pois a partir deste limite, pioramos o desempenho e conseqüentemente aumentamos o tempo de execução da obra.

Voltando ao mundo computacional, o uso de arquiteturas paralelas tem como principais dificuldades o controle das tarefas que estão sendo executadas em paralelo, ou seja, sincronizar estes processos de forma que um processo não seja repetido pelo outro, e a determinação de quantos processadores devem ser alocados para solucionar o problema, garantindo o balanceamento de carga.

Segundo Zelenovsky e Mendonça (2001, p. 2):

> As dificuldades presentes no projeto do *hardware* de máquinas paralelas não são tão complexas quando comparados com os problemas de sua programação. Diz-se que os computadores estão sempre uma geração atrasada em relação às nossas necessidades e os programas, duas gerações atrasadas. Em suma, um desafio maior que o projeto de supercomputadores é a sua programação [...].

O paralelismo, conforme Pitanga (2004, p. 8), pode ser definido então "como uma técnica utilizada em grandes tarefas e complexas para obter resultados na forma mais rápida possível, dividindo-se então em tarefas pequenas que serão distribuídas em vários processadores para serem executadas simultaneamente". Isso pode ser observado através da Figura 2.1.

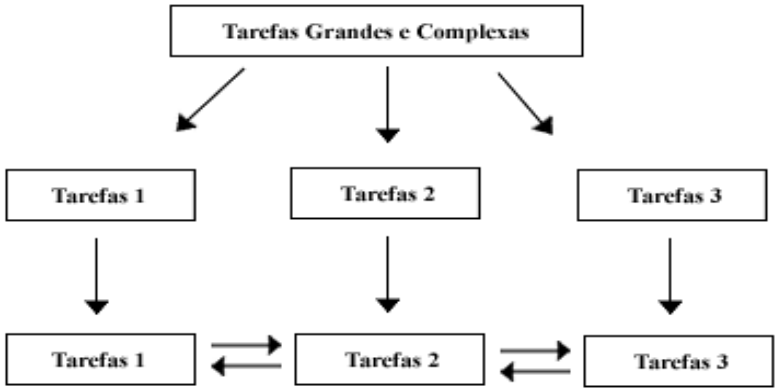

Figura 2.1: Paralelizando as tarefas
Fonte: PITANGA, 2004, p. 9.

De uma maneira geral, as tarefas podem ser divididas em (PITANGA 2003, p. 71):

- ***Serial jobs*** **–** Consiste em tarefas que são executadas em apenas um nó;
- ***Parallel jobs*** **–** São tarefas que podem ser executadas em múltiplos nós;
- ***Interactive jobs*** – Requerem velocidade de execução, com E/S direcionada para um terminal. Não precisam de muitos recursos. Espera-se que estas tarefas sejam executadas imediatamente, sem fila;
- ***Bacth jobs*** – Requerem mais recursos, como grandes espaços de memória ou bastante tempo de processamento, entretanto, a resposta não precisa ser imediata, podendo aguardar numa fila para escalonamento, até que os recursos sejam disponibilizados.

2.1 Classificação de máquinas paralelas

Muitos conceitos sobre processamento paralelo não são familiares à maioria dos usuários, embora estas técnicas venham sendo utilizadas há muitos anos. Michael Flynn em 1972, caracterizou diversos modelos de arquiteturas de computadores, baseando-se no fato de um computador executar uma seqüência de instruções sobre uma seqüência de dados. Apesar de ter sua origem ser em meados dos anos 70, ela é ainda válida e muito difundida. Dependendo se estes fluxos de dados e instruções são múltiplos ou não, e através de sua combinação, são classificados conforme o Quadro 2.1 a seguir (ROSE, 2001).

Quadro 2.1: Classificação de Flynn segundo o fluxo de instruções e o fluxo de dados

		Fluxo de Instruções (Instruction stream)	
		SI (*Single Instruction*) Serial ou Único	**MI** (*Multiple Data*) Paralelo ou Múltiplo
Fluxo de Dados)	**SD** *(Single Data)* Serial ou Único	**SISD** Máquinas vonNeumann convencionais	**MISD** Sem representante até agora
(*data stream*	**MD** *(Multiple Data)* Paralelo ou Múltiplo	**SIMD** Máquinas Vetoriais (Cray 1,CM-2,MasPar)	**MIND** Multiprocessadores e multicomputadores

Fonte: Adaptado de ROSE, 2001, p. 04.

2.1.1 SISD

Dos modelos de arquitetura propostos por Flynm, a classe SISD (*Single Instruction Single Data*, Instrução Única, Dados Únicos) é a mais simples. Neste modelo, o equipamento é considerado seqüencial, pois somente uma instrução é executada por vez para cada dado enviado (PITANGA, 2004). A Figura 2.2 mostra o esquema de funcionamento desta classe. A unidade de controle (C) é alimentada pelo fluxo de instruções (linha continua) ativando a unidade de processamento central (P). A unidade P por sua vez atua sobre um único fluxo de dados (linha tracejada) sendo lida, processada e reescrita na memória (M). Encaixa-se nesta arquitetura as maquinas de von-Neumann tradicionais, sem qualquer paralelismo, como os microprocessadores pessoais e estações de trabalho.

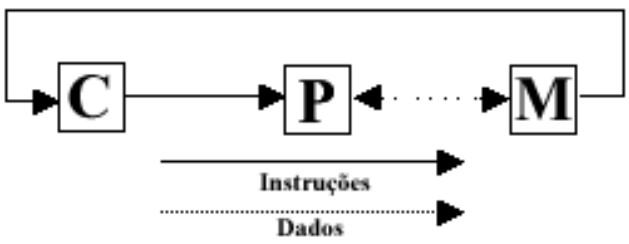

Figura 2.2: Diagrama da classe SISD
Fonte: ROSE, 2001, p.4.

2.1.2 MISD

A classe MISD (*Multiple Instruction Single Data*, Múltiplas Instruções, Dados Únicos) é a única que não possui representante computacional, sendo que até mesmo o próprio Flynn duvidou que isto um dia pudesse existir. No entanto, ele resolveu manter todas as combinações possíveis em sua classificação. Enquadraria-se nesta classe, máquinas que executam várias instruções ao mesmo tempo sobre um mesmo fluxo de dados. A classe MISD pode ser observada na Figura 2.3, onde temos múltiplas unidades de processamento (P), tendo para cada processador uma unidade de controle (C) própria, recebendo um fluxo diferente de instruções (ROSE, 2001).

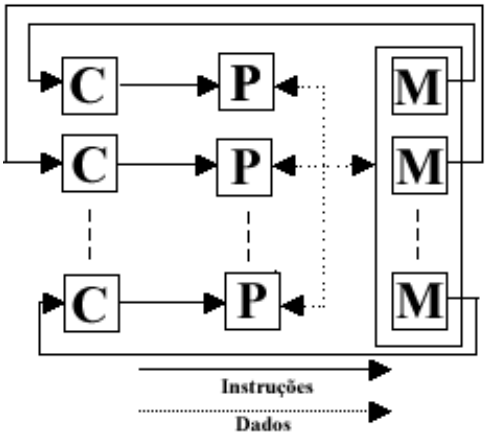

Figura 2.3: Diagrama da classe MISD
Fonte: ROSE, 2001, p. 5.

2.1.3 SIMD

As máquinas paralelas se concentram nas duas classes restantes: SIMD e MIND. O modelo de arquitetura SIMD (*Single Instruction Multiple Data*) corresponde ao processo onde uma única instrução consegue manipular ao mesmo tempo, múltiplos dados. Por exemplo,

usando um registrador de 32 bits para carregar e executar, de uma só vez, quatro dados de oito bits, teríamos a tarefa executada quatro vezes mais rápida do que manipular os mesmos dados individualmente.

O processamento é controlado através de uma única unidade de controle (C) e alimentado por um único fluxo de instruções. Sendo assim, a mesma instrução é dividida e enviada para os diversos processadores (P) envolvidos na operação. "Na prática pode se dizer que o mesmo programa está sendo executado sobre diferentes dados, o que faz com que o principio de execução SIMD se assemelhe bastante ao paradigma de execução seqüencial" (ROSE, 2001, p. 5).

Segundo Pitanga (2004, p. 12), SIMD equivale "ao paralelismo de dados, onde uma simples instrução é executada paralelamente utilizando vários dados de forma síncrona, em que se executa um único programa ao mesmo tempo". Dentro desta classe são enquadradas as máquinas Array e Vetoriais como o Cray 1 e CM-2. Computadores desta classe são compostos por menos *hardware* e memória, possuindo ainda, um *hardware* muito mais específico. A Figura 2.4 a seguir nos mostra o diagrama da classe SIMD.

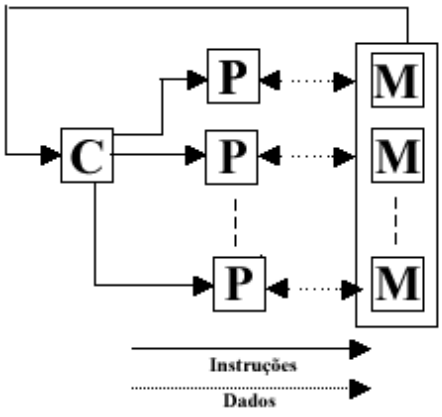

Figura 2.4: Diagrama da classe SIMD
Fonte: PITANGA, 2004, p. 12.

2.1.4 MIMD

Máquinas MIMD (*Multiple Instruction Multiple Data*, Múltiplas Instruções, Múltiplos Dados) caracterizam o modelo de execução paralela na qual cada processador opera independentemente dos demais, havendo de forma simultânea, o processamento de múltiplos fluxos de instruções e dados, agindo como se fossem um conjunto de máquinas SISD, onde cada processador tem a capacidade de rodar um programa diferente. Essa capacidade deve-se ao fato de que são construídas a partir de vários processadores operando de forma cooperativa ou concorrente, na execução de um ou vários aplicativos.

A Figura 2.5 nos mostra o diagrama da classe MIMD, onde cada unidade de controle (C) recebe um fluxo de instruções próprio e o repassa para a sua unidade processadora (P), sendo executado sobre um fluxo de instruções próprio. "Dessa forma cada processador executa o seu próprio programa sobre seus próprios dados de forma assíncrona" (ROSE, 2001, p. 6). Enquadra-se neste modelo de arquitetura de servidores com múltiplos processadores (*dual, quad*), sistemas *Massively Parallel Processors* (MPP - Processadores Paralelos Massivos) e os *clusters* de computadores.

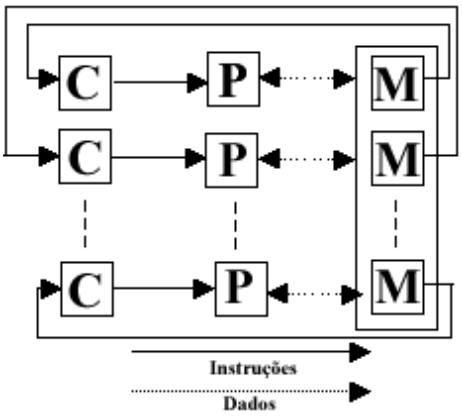

Figura 2.5: Diagrama da classe MIND
Fonte: ROSE, 2001, p. 7.

Embora a classificação de Flynm seja conveniente em um primeiro momento, ela é muito imprecisa quanto sua classificação, devido às variedades de multiprocessadores

existentes. Dentro deste contexto, segundo Rose (2001, p. 6), "o princípio MIMD é bastante genérico, pois qualquer grupo de máquinas, se analisado como uma unidade (executando, por exemplo, um sistema distribuído), pode ser considerada uma máquina MIMD". Logo, essa definição deixa margem para que várias topologias de máquinas paralelas e de redes de computadores sejam enquadradas como MIMD. Assim, para diferenciar as diversas topologias MIMD, Händler propôs em 1977 uma notação para distinguir o paralelismo (AMORIM, 1988). Em sua proposta, são utilizados como critérios de classificação, a maneira em que está organizada fisicamente a memória principal e o tipo de acesso que cada processador tem à totalidade da memória. Essa classe de arquitetura é dividida em arquitetura de computadores MIMD de memória compartilhada e memória distribuída. Nas próximas seções serão detalhados os principais aspectos com relação a este tipo de classificação.

2.2 Organização segundo o compartilhamento de memória

Em função de características como tempo de acesso, capacidade de armazenamento e custo, os dispositivos de armazenamento em computadores são classificados de acordo com uma hierarquia, denominada hierarquia de memória. Na parte superior desta hierarquia, estão os registradores do processador, que são dispositivos de armazenamento temporários, extremamente rápidos, com capacidade de armazenar apenas um dado (uma palavra) e está localizado junto ao processador, podendo ser acessado pelo processador rodando à sua velocidade máxima. Em seguida, vem a memória *cache*, que é uma memória pequena e muito rápida que age como uma espécie de *buffer*[10], ou seja, consiste em uma memória auxiliar que facilita a recuperação de informações recentemente/freqüentemente acessadas pelo processador. Seu acesso é rápido, porém inferior ao dos registradores. Na seqüência, vem a memória principal, cuja capacidade de armazenamento varia de 16 *megabytes* para sistemas

[10] Consiste numa pequena área de memória ultra-rápida, utiizada para melhorar a velocidade de acesso a um determinado dispositivo. É encontrado em HDs, gravadores de CD, modems, dentre outros dispositivos. O termo "*buffer*" normalmente é utilizado em relação aos dispositivos citados anteriormente, enquanto o termo "*cache*" é mais usado com relação aos processadores e memória RAM (*Random Access Memory, M*emória de Acesso Aleatório).

pequenos a dezenas de *gigabytes* para equipamentos de alta performance. Os níveis desta hierarquia citados até o momento são considerados voláteis, pois precisam estar energizados para manterem seu conteúdo gravado. Por outro lado, dando continuidade a esta hierarquia, temos os dispositivos para armazenamento permanente das informações, na qual se encaixam os discos magnéticos e por fim, na parte inferior, encontra-se as fitas magnéticas e os discos ópticos (TANENBAUM, 2001).

Quando vários elementos processadores são interconectados, a forma como cada um enxerga a memória é fator decisivo para a definição dos modelos de comunicação. Assim, conforme os aspectos lógicos da memória, podemos classificá-las em: Memória compartilhada (*shared memory*) ou memória privativa (*multiple private address space*, múltiplos espaços de endereçamento privados) (ROSE, 2001).

Memória Compartilhada refere-se quando existe um único espaço de endereçamento, que será usado de forma implícita para a comunicação entre os processadores, ou seja, cada processador consegue ter acesso a todo o espaço de memória da arquitetura. Sua comunicação e feita através de *load* (leitura) e *store* (escrita) nos endereços de memória. Temos memória privativa quando existe um espaço de endereço único e distinto para cada processador, o que implica em comunicação explicita entre os processadores, através da troca de mensagens com operações *send* (envio) e *receive* (recebimento).

Outra maneira de caracterizar as memórias é quanto aos aspectos físicos. Caso a memória for composta por vários módulos, estando cada módulo próximo ao processador, denominamos que está memória é distribuída (*distributed memory*). Quando a memória está localizada a mesma distância [11]de todos os processadores, independente se foi implementada com um ou vários módulos, ela é considerada memória centralizada (*centralized memory*).

[11] O termo distância neste caso, significa que todos os processadores levam o mesmo tempo para acessar a memória, ou seja, a diferença de tempo entre o início de um evento e o momento em que seus efeitos tornam-se perceptíveis, será igual em todos os processadores.

Dependendo da forma como uma máquina paralela utiliza a memória, teremos uma arquitetura MIMD do tipo multiprocessador ou multicomputador.

2.3 Multiprocessador

Um computador multiprocessador constitui no tipo de máquina paralela onde todos os processadores dispõem do mesmo espaço de endereço de memória, sendo que a comunicação entre os processos é feita através de memória compartilhada, através de operações do tipo *load* e *store*. Sua estrutura assemelha-se à colocação de múltiplos processadores da arquitetura convencional em uma máquina de von-Neumann tradicional. Através de uma rede de interconexão, uma memória (M) pode ser acessada por todos os processadores (P). Segundo Rose (2001, p. 7), "este tipo de máquina possui apenas um espaço de endereçamento, de forma que todos os processadores (P) são capazes de endereçar todas as memórias (M)". Na Figura 2.6 é ilustrado um modelo de arquitetura de um multiprocessador. Enquadra-se nesta categoria, os sistemas de processamento simétrico (SMP, *Symmetric Multiprocessors*), que possuem de dois a sessenta e quatro processadores.

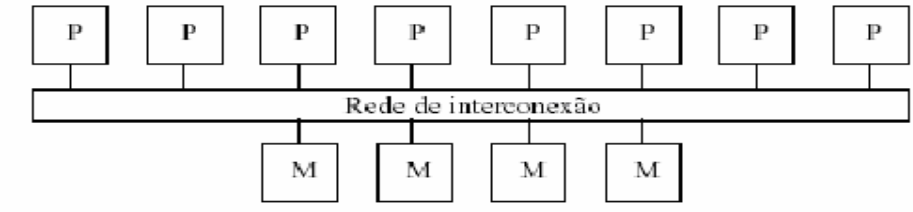

Figura 2.6: Modelo de arquitetura de um multiprocessador
Fonte: ROSE, 2001, p. 8.

Com relação ao tipo de acesso às memórias do sistema, os multiprocessadores podem ser classificados quanto à distância dos processadores à memória, e quanto aos esquemas de

coerência de *cache*. Os computadores multiprocessados podem ser constituídos com diferentes formas de organização de memória, como vemos a seguir.

2.3.1 UMA (*Uniform Memory Access*, Acesso uniforme a memória)

Neste modelo de arquitetura, a memória utilizada é centralizada, ou seja, todos os processadores têm o mesmo tempo de acesso aos dados alocados na memória em todas as suas posições. É utilizada normalmente em multiprocessadores pequenos devido ao aumento do tempo de acesso à medida que o sistema é ampliado (PITANGA, 2003). A forma de interconexão mais comum neste tipo de máquina é o barramento, sendo que a memória geralmente é implementada em um único módulo. Na Figura 2.7 podemos observar este modelo de acesso a memória.

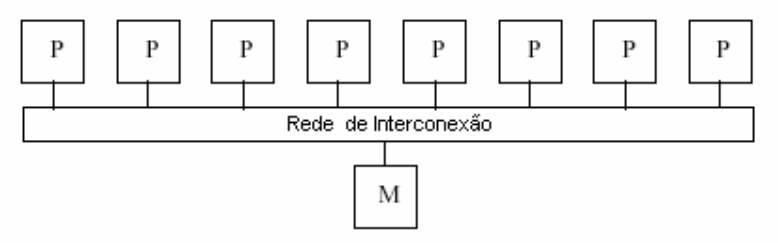

Figura 2.7: Arquitetura UMA
Fonte: ROSE, 2001, p. 8.

> É importante ressaltar que máquinas com outras redes de interconexão e com memórias entrelaçadas (implementadas com múltiplos módulos e desta forma permitindo acesso paralelo em diferentes módulos) também se enquadram nesta categoria se mantiverem o tempo de acesso à memória uniforme para todos os processadores do sistema [...] (ROSE, 2001, p. 08).

Um problema encontrado nestas máquinas é quanto ao congestionamento das interligações, pois o acesso à memória fica limitado a uma única transferência por vez. Uma alternativa utilizada tem sido o uso de memórias *cache*, a fim de diminuir a diferença entre a

memória principal e o processador. No entanto, podem existir muitas vezes, diversas cópias de um mesmo dado sendo manipulados simultaneamente nas *caches* de vários processadores. Neste caso, é necessário que se garanta que os processadores sempre acessem a cópia mais atualizada da memória. Esta garantia é chamada de coerência de *cache* (*cache coherence*). Normalmente, este problema é resolvido pelo *hardware* destas máquinas (ROSE, 2001).

2.3.2 NUMA (*Non Uniform Memory Acess*, Acesso não uniforme à memória)

Neste tipo de multiprocessadores, a memória utilizada é dividida em tantos blocos quanto forem os processadores do sistema. Cada módulo de memória é conectado por intermédio de um barramento a um processador resultando em uma memória local. No entanto, é possível ter acesso aos módulos ligados a outro processador, uma vez que o espaço de endereçamento é único, sendo que, o tempo de leitura e escrita na memória que pertence ao outro processador, dito remoto, é maior que o tempo utilizado para o mesmo procedimento na memória do próprio processador, dito local (Figura 2.8). Surge daí o nome de acesso não uniforme a memória, pois o tempo de acesso entre as memórias muda de acordo com o endereço desejado. Esta diferença faz com que sejam necessários cuidados especiais ao se programar em arquiteturas deste tipo.

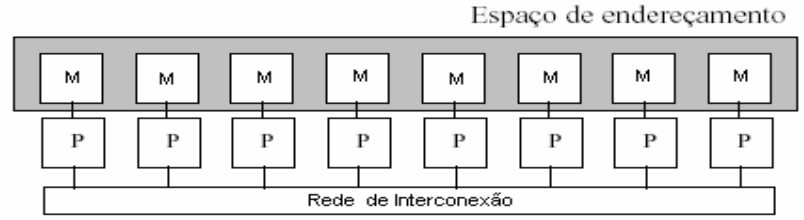

Figura 2.8: Arquitetura NUMA
Fonte: PITANGA, 2004, p. 16.

Ao contrário da solução anterior, este modelo de arquitetura não é comprometido pelo congestionamento das ligações. Geralmente é utilizado em processadores de grande

escala, uma vez que esta metodologia proporciona baixos valores de acesso à memória local, contribuindo para o bom desempenho (PITANGA, 2003).

A fim de resolver a consistência da memória, dada à utilização de uma *cache* local em cada processador, é necessário garantir a coerência dos dados entre a *cache* e a memória principal. Dependendo da forma como isto é tratado, e se foi feito via *software* ou *hardware*, esta classe é sub-dividida em:

- NCC-NUMA (*non-cache coherent non-uniform memory access*, acesso não uniforme à memória sem coerência de *cache*): Este modelo é uma variação da classe NUMA, onde não se tem coerência garantida de *cache* em *hardware*;

- CC-NUMA (*cache coherent non-uniform memory access*, acesso não uniforme à memória com coerência de *cache*): Neste modelo, têm-se coerência de *cache* garantida em *hardware*. "Nessa arquitetura, a memória é dividida em tantos blocos quanto forem os processadores do sistema, e cada bloco de memória e conectado via barramento a um processador como memória local" (PITANGA, 2003, p. 10).

- SC-NUMA (*software coherente non-uniform memory access*, acesso não uniforme à memória com coerência de *cache* em *software*). Esta classe diferencia da anterior pelo fato do tratamento de coerência de *cache* ser implementado através de *software*. No entanto, este procedimento é implementado de forma transparente ao usuário.

2.3.3 COMA (*Cache-Only Memory Arquitecture*, Arquitetura de memória somente com *cachê*).

Considerada um tipo especial de NUMA, onde as memórias globais distribuídas são substituídas por *caches*, não existindo uma hierarquia de memória para cada processador,

formando um espaço de endereçamento global único (PITANGA, 2003). Essa tecnologia é a única que tem suporte de *hardware* para a replicação efetiva do mesmo bloco de *cache* em múltiplos nós (nas arquiteturas anteriores os blocos são invalidados e não atualizados). (ROSE, 2001). Resumindo, as máquinas COMA oferecem uma melhor performance do que as máquinas NUMA, no entanto, poucas máquinas deste tipo são construídas, visto que neste modelo faz-se necessário à utilização de um complexo mecanismo de coerência de *cache* em *hardware*, o que acarreta em um desenvolvimento mais caro deste tipo de tecnologia. Na Figura 2.9 pode ser visto o modelo de arquitetura COMA.

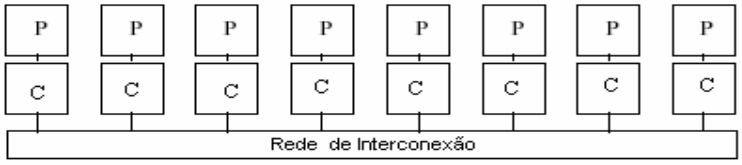

Figura 2.9: Arquitetura COMA
Fonte: PITANGA 2004, p. 17.

2.4 Multicomputadores

O termo multicomputadores vem do fato deste tipo de máquina paralela ser construído a partir da replicação de todos os componentes da arquitetura convencional, ao contrário dos multiprocessadores, que replicavam apenas o componente processador. Neste modelo de arquitetura (Figura 2.10), não existe um espaço de endereçamento central, ou seja, cada processador tem sua própria memória local, sendo que a troca de informações é feita a partir da troca de mensagens entre as máquinas (*message passing systems*, sistema de troca de mensagens).

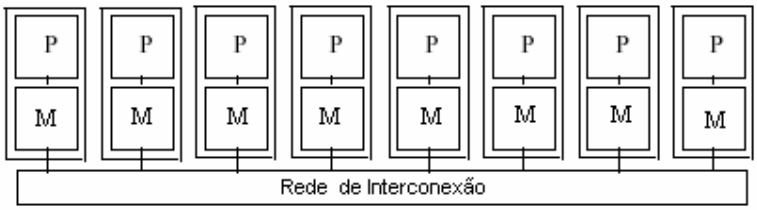

Figura 2.10: Modelo de arquitetura de um Multicomputador
Fonte: ROSE, 2001, p. 11.

Quanto ao tipo de acesso à memória do sistema, os multicomputadores podem ser classificados em NORMA (*Nom-Remote Memory Access*, sem acesso a variáveis remotas). Segundo Rose (2001), como a arquitetura tradicional inteira foi replicada na elaboração deste modelo, os registradores de endereçamento de cada nó da rede só conseguem endereçar a sua memória local. Desta forma, conforme Pitanga (2003), as redes de computadores podem ser vistas como arquiteturas NORMA. A seguir, na Figura 2.11 é mostrado uma visão geral das classificações das máquinas MIMD, segundo o compartilhamento de memória.

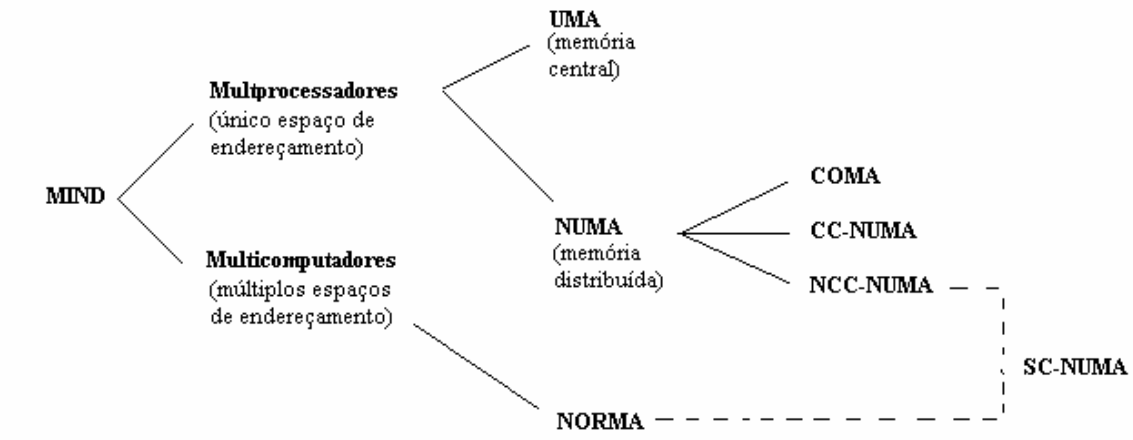

Figura 2.11: Classificação das máquinas MIND
Fonte: PITANGA, 2004, p. 19.

Duas categorias diferenciam os multicomputadores:

MPP (*Massively Parallel Processors*, Computadores Massivamente Paralelos). São multicomputadores compostos por um grande número de processadores, fortemente acoplados através de uma rede de conexão de alta velocidade. Cada nó pode ter uma variedade de

componentes de *hardware*, mas consiste geralmente em uma memória principal e em um ou mais processadores. Normalmente são arquiteturas de custo elevado, pois utilizam processadores específicos e redes de interconexão proprietárias (BUYYA, 1999).

COW (*Cluster of Workstations*, Conjunto de estações de trabalho). Também chamadas de NOW (*Network of Workstations*, Redes de estações de trabalho), essas máquinas são construídas a partir de computadores comuns (PCs) ligados por redes de interconexão tradicionais.

3 SISTEMAS DISTRIBUÍDOS

Sistemas distribuídos é uma área da computação que difere de maneira fundamental dos demais sistemas computacionais em que há compartilhamento de memória, e por isso, um entendimento sólido dos princípios de computação distribuída é de natureza essencial.

A programação distribuída ou paralela terá vantagens em arquiteturas de sistemas paralelos e distribuídos se a aplicação estiver apropriada para este tipo de ambiente. Em um primeiro momento, considera-se a diferença entre concorrência e paralelismo. Temos concorrência quando parte de um programa pode ser executado independentemente. Já o paralelismo é quando temos partes concorrentes de um programa podendo ser executado ao mesmo tempo em processadores distintos. Esta distinção é importante, visto que a concorrência é uma propriedade do programa e um eficiente paralelismo é uma propriedade do equipamento.

Define-se como sistema distribuído um conjunto de elementos, onde cada elemento é composto por um ou mais processadores e de uma ou mais memórias, que se comunicam através de uma rede de interconexão, dispondo de *softwares* específicos para este tipo de tarefa, a fim de resolver um determinado problema. Os *softwares* neste ambiente, seja ele aplicativo ou mesmo um sistema operacional, têm que estar de acordo com as características dos sistemas distribuídos. Ao contrário de um sistema mono-processado, neste ambiente existe a necessidade de sincronismo, interação e controles entre os diferentes ambientes de *software*, ou diferentes instâncias, que compõem o sistema na sua totalidade (GOULART, 2002).

A utilização de sistemas distribuídos permite a coordenação de atividades através da cooperação e interação entre os processos, compartilhando informações e recursos do sistema como *software* e *hardware*. Uma das vantagens principais de se utilizar sistemas distribuídos é quanto a sua escalabilidade, ou seja, seu crescimento computacional pode ser obtido através da inclusão de mais recursos de processamento junto à rede. Outra vantagem de grande valia é a possibilidade de implementação de sistemas tolerantes a falhas por meio de replicação de serviços ou de máquinas distintas. No entanto, para que tais recursos sejam aproveitados da melhor forma possível, é preciso fornecer um suporte adequado de *software*, dentre os quais, existem diversos aspectos relacionados ao controle destes ambientes (PITANGA, 2004).

3.1 Características dos sistemas distribuídos

Um critério utilizado para classificar os sistemas distribuídos é quanto à forma de comunicação entre os processadores. Esse critério divide os sistemas distribuídos entre aqueles que à comunicação é feita por *broadcast* e aqueles em que a comunicação se dá de forma ponto-a-ponto.

3.1.1 Rotinas de Comunicação *Broadcast*

Caracteriza-se pela participação de dois ou mais processos em cada operação de comunicação. Neste tipo, todos os processadores do sistema podem se comunicar diretamente com todos os outros, utilizando para isso um ou mais canais de comunicação compartilhados por eles. Nesse sistema, existe o problema claro de colisões das mensagens quando dois ou mais processadores tentam utilizar o meio compartilhado simultaneamente (AMORIM, 1998). Na Figura 3.1 pode ser observado este tipo de sistema, onde N processadores compartilham com N canais de comunicação.

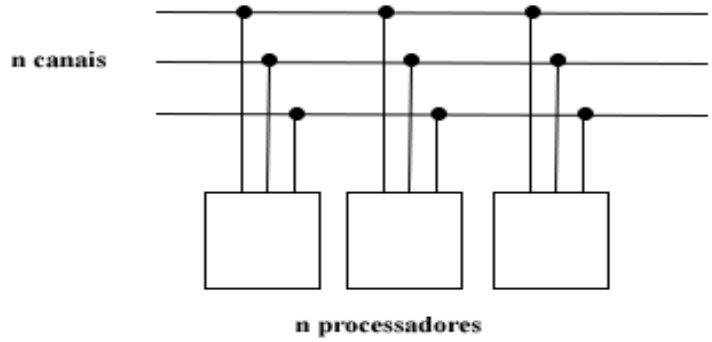

Figura 3.1: Um sistema distribuído por Broadcast
Fonte: AMORIM, 1998, p. 79.

3.1.2 Rotina de comunicação ponto a ponto

As rotinas de comunicação ponto-a-ponto são responsáveis pela ação básica de uma biblioteca de troca de mensagens, que se trata da transferência de uma mensagem. Cada transferência ponto-a-ponto envolve somente dois processos, um transmissor e um receptor. Segundo Amorim (1998, p.80), "essa exclusividade nos diversos canais elimina o problema de colisões, mas causa por outro lado à necessidade de mensagens precisarem ser enviadas ao longo de rotas que passam por outros processadores que não a sua origem e seu destino". A Figura 3.2 ilustra este tipo de comunicação.

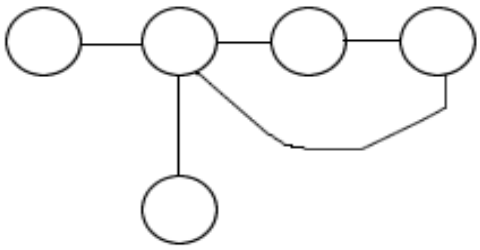

Figura 3.2: Comunicação Ponto-a-Ponto
Fonte: AMORIM, 1998, p. 80.

Outro aspecto que sugere uma classificação na computação distribuída é no que diz respeito às bases de tempo dos diversos processadores envolvidos no sistema. Através deste critério, os sistemas distribuídos podem ser síncronos ou assíncronos.

3.2 Sistemas distribuídos síncronos

Neste caso, todos os processadores do sistema funcionam segundo um relógio global comum, onde todos têm acesso. O sincronismo é um atributo de modelos computacionais e sistemas distribuídos que está relacionado ao comportamento tanto dos processos quanto da própria rede. É caracterizado basicamente por limites em latência[12] de comunicação e em prazos de processamento, ou seja, neste modelo, existem meios que procuram garantir a existência de uma referência única de tempo, tanto para transmissor quanto para o receptor. Na prática, a implementação de modelos síncronos, quando possível, envolve certa complexidade, impossibilitando na maioria das vezes a sua construção. Um exemplo disso pode ser observado na Internet, onde é impossível prever limites máximos ou mínimos para o atraso no envio de mensagens, pois a carga da rede e o caminho a ser percorrido pelos pacotes no momento do envio nunca são iguais (BESSANI, 2002; AMORIM, 1998).

3.3 Sistemas distribuídos assíncronos

Enquanto de um lado temos o modelo síncrono, onde todas as variáveis importantes do ambiente são limitadas e conhecidas, do outro lado temos o modelo assíncrono, onde não existe uma limitação para as variáveis do ambiente computacional. Neste modelo, a principal característica é a ausência total de uma base de tempo comum a todos os processadores que o

[12] É o tempo que uma mensagem demora a atravessar um sistema.

compõe o sistema, ou seja, cada processador possui então um relógio local e independente dos demais, não sendo possível fazer uma previsão exata de quanto tempo uma determinada informação ira chegar a seu destino (AMORIM, 1998). Em suma, os sistemas distribuídos reais são sistemas assíncronos.

3.4 Ambientes de troca de mensagens

Para que se haja processamento em conjunto na execução de uma aplicação, é importante que exista comunicação e sincronismo. Em um ambiente compartilhado, isto é obtido através do compartilhamento de espaços de memória entre os diversos processos em paralelo. Porém, quando temos memória distribuída, onde cada processador possui sua própria memória local, é necessário definir uma maneira de interação entre os processos em andamento. Essa comunicação se dá através da troca de mensagens. Este paradigma define um conjunto de primitivas que permite a comunicação entre os processos. Logo, a latência e a velocidade com que as mensagens podem se comunicar são fatores limites neste modelo de comunicação.

De acordo com Linhalis e Fiats (1998, p. 3) "um programa que utiliza troca de mensagens pode ser definido como um conjunto de programas seqüenciais, distribuídos em vários processadores, que se comunicam através de um conjunto limitado em bem definido de instruções". Essas instruções formam o ambiente de troca de mensagens. Sua estrutura é formada através de uma linguagem seqüencial, como (C, Fortram, etc) e uma biblioteca de trocas de mensagens que é responsável pela ativação e interação entre os processos em paralelo.

A grande aceitação deste paradigma de troca de mensagens pode ser justificada através de alguns fatores:

- Pode ser executado em diversas plataformas;

- Adapta-se naturalmente ao aumento de componentes do sistema;

- Não deve se tornar obsoleto nas próximas décadas, pois de alguma maneira, sempre será necessário utilizar uma comunicação entre os processos.

Inicialmente, os ambientes de troca de mensagens foram desenvolvidos para máquinas MPP. A ausência de um padrão nestes ambientes fez com que cada fabricante desenvolve-se um modelo próprio, sem se preocupar com a portabilidade [13]do *software* gerado. Através disso, muito conhecimento foi adquirido, pois os diferentes projetos destes ambientes tratavam de aspectos distintos para o seu sistema. Alguns exemplos destes ambientes são: nCUBE, PSE, IBM EUI, Meiko Cs System e Thinking Machines CMMD.

Hoje em dia, os ambientes de passagem de mensagens estão modificados possuindo alguns objetivos especiais:

- Tirar proveito dos sistemas distribuídos no desenvolvimento de aplicações paralelas;

- União de plataformas heterogêneas (MPP e/ou redes de estação de trabalho);

- Permitir a portabilidade das aplicações.

Para que estes ambientes sejam empregados em *clusters* e *grids computacionais*[14], alguns aspectos devem ser levados em consideração: acoplamento fraco entre computadores; a possível diferença de largura de banda e atrasos na rede de comunicação; a possível

---

[13] É a possibilidade de executar ou modificar um programa para que possa ser executado em mais de um sistema de computador, ou sob mais de um sistema operacional. Os *softwares* de alta portabilidade podem ser transferidos para outros sistemas sem grandes esforços.

[14] A nomenclatura *grid* é baseada nas malhas de interligação dos sistemas de energia elétrica. Em um sistema elétrico, o usuário utiliza a energia sem se perguntar aonde à mesma foi gerada. De maneira semelhante, a idéia de um *grid computacional* é que possamos criar um ambiente computacional distribuído que possua mecanismos que permitam o processamento, o armazenamento e o uso de equipamentos de forma transparente para os usuários da configuração (Dantas, 2005, p. 204).

heterogeneidade em termos de *hardware*, sistemas operacionais e linguagens, bem como diferentes autoridades e políticas de segurança (DANTAS, 2005).

Dentro deste contexto, vários grupos de pesquisas desenvolveram ambientes de passagem de mensagens independentes do computador a serem utilizados, denominados como ambientes de passagem de mensagem com plataforma portátil. Alguns exemplos desses ambientes são: P4, Express, PARMACS, Linda, Parallel Virtual Machine (PVM) e Message Passing Interface (MPI). Por serem ambientes de programação que são representativos nas configurações distribuídas e paralelas, respectivamente, a seguir serão apresentados maiores detalhes sobre PVM e MPI.

3.4.1 PVM - *Parallel Virtual Machine* (Máquina Virtual Paralela)

A biblioteca de programação paralela de mensagens PVM foi implementada em 1989 pelo *Heterogeneous Network Project*, um esforço conjunto da O*ak Ridge National Laboratory, University of Tennesse, Emory University e Carneige Mellon University*, a fim de facilitar a computação paralela distribuída (PITANGA, 2004).

Conforme Dantas (2005, p. 114), "o pacote de *software* PVM tem como abordagem prover as facilidades de um ambiente de programação paralela em adição a um transparente mecanismo de agregar inúmeras máquinas com arquiteturas homogêneas ou heterogêneas".

Já, de acordo com o grupo de desenvolvimento desta biblioteca, o PVM é um pacote de *software* que permite a execução de sistemas operacionais tais como Unix, Solaris, Windows, em um conjunto de computadores heterogêneos, possibilitando esses serem agregados de uma rede e utilizados como um único e grande computador paralelo. Possui uma grande portabilidade, podendo ser utilizado de uma máquina tipo *laptop* até um supercomputador Cray. Sua distribuição é gratuita e pode ser obtida através do site <www.csm.ornl.gov/pvm/> (PVM, 2006).

O sistema PVM é composto em duas partes representadas através da Figura 3.3. A primeira parte é composta por um ambiente de programação, ou seja, um ambiente de troca de mensagens, na qual contém rotinas chamáveis pelo usuário para codificar, enviar e receber mensagens, que deverá ser utilizado dentro de uma linguagem de programação. Essa biblioteca permite que os desenvolvedores de aplicações paralelas possam utilizar o ambiente distribuído de maneira transparente, ou seja, sem se preocuparem com a localização física das máquinas.

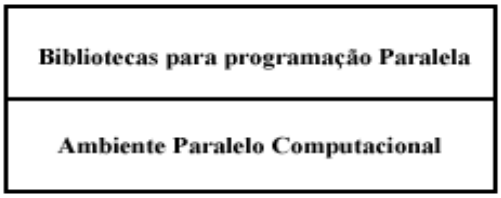

Figura 3.3: Facilidades do pacote PVM
Fonte: DANTAS, 2005, p. 115.

A segunda parte compreende o ambiente paralelo computacional (PPE – *Parallel Programming Environment*), que é um conjunto de mecanismos que disponibiliza para a camada de biblioteca as facilidades de interligação e gerenciamento de conexões com máquinas remotas (DANTAS, 2005). Consiste em um processo *daemon* [15], que deve estar instalado em todas as máquinas que compõem o conjunto de computadores, criando assim uma abstração de máquina paralela virtual para os usuários das aplicações (PITANGA, 2004). Cada versão do *daemon* depende exclusivamente do SO instalado.

De acordo com Dantas (2005), o PVM é muito utilizado em *clusters* e *grids* computacionais, uma vez que uma grande quantidade de recursos nestes ambientes pode ser agregados de uma maneira produtiva e prover para os usuários de aplicações uma forma eficiente de execução de suas tarefas. Algumas características deste ambiente podem ser observadas a seguir:

[15] Programa executado continuamente e que existe com a finalidade de administrar as solicitações periódicas de um serviço recebidas através de outros programas para a execução de uma determinada ação.

- *Software* de domínio público;
- Grande aceitação e utilização no mundo;
- Facilidades de instalação e uso;
- Interoperabilidade e portabilidade;
- Abstração dos controles e recursos dos processos.

Diversos aplicativos utilizam o PVM a fim de auxiliar os usuários dos ambientes paralelos, dos quais destacam-se:

- XPVM: Consiste numa interface gráfica que fornece informações sobre o console PVM, ou seja, traz informações desse ambiente, apresentando às máquinas que estão interligadas na rede e que estão ou não participando na execução de uma tarefa. Possibilita também, monitorar o funcionamento de tarefas ao longo de um determinado tempo, bem como a comunicação entre elas (XPVM, 2006). Esses monitores do ambiente paralelo, segundo Dantas (2005, p.116) é extremamente importante, "[...] pois através deles é possível entender como está funcionando e como é possível melhorar o desempenho". A interface de um destes monitores pode ser visualizada na Figura 3.4 abaixo:

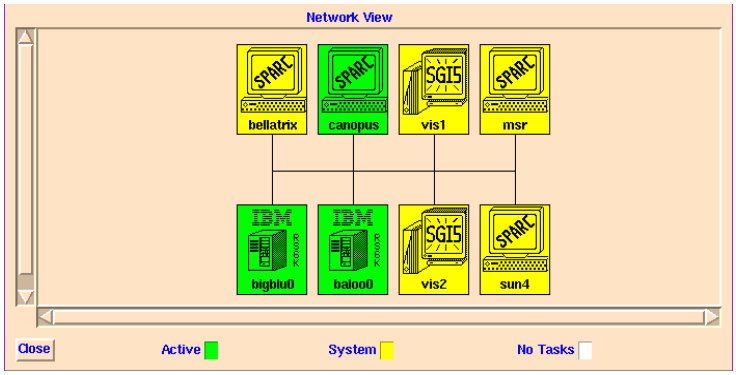

Figura 3.4: Vista da rede do software XPVM

Fonte: XPVM, 2006, p.2.

- Cumulvs (*Collaborative User Migration, User Library for Visualization and Steering*): Compreende em um conjunto de *softwares* com infra-estrutura para o desenvolvimento de ambientes colaborativos. Como características destacam-se: facilidades de visualização interativa, mecanismo para o desenvolvimento de sistemas tolerantes a falhas, agregação remota de aplicações distribuídas e capacidade de migração de aplicações em ambientes heterogêneos distribuídos (CUMULVS, 2005).

3.4.2 MPI - *Message Passing Interface* (Interface de Passagem de Mensagens)

A utilização de diversas bibliotecas de troca de mensagens para programação paralela gerou a necessidade dentro da comunidade de desenvolvedores para que houvesse uma padronização deste ambiente, a fim de obter uma maior interoperabilidade entre as máquinas. Assim, através de um esforço em conjunto de um grupo de empresas envolvidas na comercialização de produtos de alto desempenho, centros de pesquisas e universidades, surge um padrão denominado MPI. Conforme Bookman (2003, p.162) "o objetivo de MPI é desenvolver um padrão para passagem de mensagem através de plataformas heterogêneas". Assim como Bookman, Pitanga (2004, p.122) ressalta que "o principal objetivo do MPI é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas que utilizem troca de mensagens".

As funcionalidades básicas do MPI descendem de práticas comuns de paralelismo e outras bibliotecas de passagem de mensagem. Assim como em outros pacotes, a comunicação entre os processos requer vários pedaços de informações: processo transmissor e receptor, endereço inicial de memória do destino, mensagem de identificação e o grupo de processos que podem receber a mensagem. O MPI disponibiliza uma relação que permite a comunicação entre processos paralelos, no entanto, o método na qual são criados os processos e como é estabelecida está comunicação não é especificado.

O MPI possui rotinas para programação em linguagens distintas como: Fortran 77/90, C / C++. Logo, os programas são escritos e compilados em uma determinada linguagem e ligados à biblioteca MPI. "Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado. Porém, esses processos podem executar diferentes programas" (PITANGA, 2004, p.123). Diante disso, muitas vezes este padrão é citado como *Multiple Program Multiple Data* (MPMD).

Um fato relevante que deve ser prestado atenção em relação ao padrão MPI reside no fato de que o esforço de padronização se atém somente ao nível de troca de mensagens, ou seja, o MPI é tão somente a biblioteca de programação paralela. Resumindo, este padrão corresponde somente à camada 1 (figura 3.3) referente ao PVM, podendo ser utilizado em qualquer plataforma que execute código paralelo (DANTAS, 2005).

> O MPI implementa um excelente mecanismo de portabilidade e independência de plataforma computacional. 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 com SO Solaris ou PC com Linux com quase nenhuma modificação no código fonte da aplicação" (PITANGA, 2004, p.124).

Esta biblioteca, assim como diversos outros documentos relevantes ao desenvolvimento podem ser obtidos através do Fórum de MPI <http://www.mpi-forum.org> (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*;
- MPINCH: ANL/MSU – *Argonne National Laboratory e Missipi State University;*
- UNIFY: *Missipi State University*;
- LAM: *Ohio Supercomputer Center*;
- MPL: *IBM Reserch.*

4 CLUSTERS COMPUTACIONAIS

A opção de pedir ajuda, vista no capitulo dois, e que traduzida em termos computacionais pelo uso de distribuição de processos, através da computação paralela e distribuída, assume um papel importante na melhoria de performance das aplicações. Este tipo de processamento tradicionalmente era baseado ao longo de muitos anos em computadores paralelos específicos, tais como MPP e SMP.

Adicionalmente, a constante necessidade de maior poder de processamento por parte da evolução das aplicações, assim como o elevado custo para a aquisição de arquiteturas paralelas dedicadas, que acaba tornando inacessíveis para a maioria das empresas e usuários, partiu a idéia de utilizar o poder computacional empregado em *hardware* comum, criado originalmente para outros fins, mas que, se utilizado de forma conjunta poderia atingir um desempenho equivalente ao de supercomputadores. Essa agregação de computadores é denominada pela literatura como *cluster* computacional.

Entretanto, a utilização de computação em *cluster* não é tão recente. Sua origem é datada nos anos 60 pela IBM, onde duas de suas máquinas, *Houston Automatic Spooling Priori* (HASP) e *Job Entry System* (JES) permitiram a interligação entre *mainframes* [16]a um custo relativamente baixo. Desde então, estudos e aperfeiçoamentos vêem sendo

[16] É um computador de grande porte, dedicado normalmente ao processamento de um grande volume de informações.

desenvolvidos dentro da computação paralela e distribuída, integrando características antes peculiares a diferentes sistemas proprietários.

Segundo Dantas (2005, p.147), "as configurações, conhecidas como *clusters* computacionais, tem como seu maior objetivo a agregação de recursos computacionais para disponibiliza-los para a melhoria de aplicações". De uma maneira geral, um *cluster* computacional é composto por dois ou mais computadores, que trabalham em conjunto na execução de tarefas. Esse trabalho é executado de forma transparente ao usuário, ou seja, sua implementação cria uma ilusão de um único recurso ou computador virtual.

Conforme Pitanga (2004, p. 17):

> [...] um *cluster* é composto por um conjunto de nós processadores (PCs ou estações) autônomos e que interligados comportam-se como um sistema de imagem única. O conceito de imagem única dita que um sistema paralelo ou distribuído, independentemente de ser composto por vários processadores ou recursos geograficamente distribuídos, deve comportar-se com um sistema centralizado do ponto de vista do usuário. Dessa forma, todos os aspectos relativos à distribuição de dados e de tarefas, comunicação e sincronização entre tarefas e a organização física do sistema devem ser abstraídos do usuário, ou seja, devem ser transparentes a ele.

Atualmente, a fim de obter um melhor desempenho das aplicações, tem se observado um agrupamento físico e virtual de inúmeros computadores. Um fato relevante para o aumento de este paradigma ser cada vez maior, deve-se ao fato da grande oferta de PCs existentes no mercado, bem como a capacidade de disponibilizar maiores e melhores recursos computacionais para as aplicações com excelente custo-benefício (DANTAS, 2005).

> A utilização de um agrupamento de máquinas como solução para computação de alto desempenho, alta disponibilidade e balanceamento de carga é uma realidade e não se pode virar os olhos quanto a isso. Com apenas os equipamentos já existentes e *softwares* disponíveis gratuitamente, essas tecnologias podem ser utilizadas em cursos universitários, pesquisas e soluções comerciais (PITANGA 2003, p. 2).

O aumento gradativo da utilização de *clusters*, proveu um forte impacto no mercado de supercomputadores, onde a partir da segunda metade dos anos 90, principalmente, as

vendas de máquinas paralelas tiveram uma considerável redução, chegando a levar a falência diversas empresas do setor.

Conforme já apresentado, os *clusters* compreendem um conjunto de nós que operam de maneira cooperativa e transparente, disponibilizando serviços e/ou efetuando tarefas a fim de atender uma demanda específica. A fim de atenderem tal demanda, os *clusters* apresentam uma arquitetura específica e elementos constituintes, onde o entendimento de conceitos chaves nesse cenário é importante. De acordo com Buyya (1999) e Pitanga (2004), seguem alguns conceitos:

- Nó ou node: Consiste na unidade básica do *cluster*, ou seja, cada unidade de processamento (pode ser composto por um simples PC ou poderosas estações de trabalho SMP). A comunicação entre os nós é efetuada por mensagens transmitidas através de uma rede;

- Recursos: Podem ser físicos (unidade de disco) assim como lógicos (*softwares*), sendo que existe a possibilidade de migração de recursos entre os nós;

- Dependência de Recursos: Como o próprio nome sugere, este termo indica a situação na qual um recurso depende da disponibilidade de um outro recurso. Por exemplo, um servidor de *e-mails* depende da interface de rede *on-line*[17] e de um sistema operacional em execução para prover seus serviços;

- Grupo de Recursos: É usado para representar a unidade de recursos, de tal forma que indique critérios no qual os recursos devem ser agrupados. Cabe ao administrador associar uma quantidade de recursos independentes em um único grupo, garantindo assim a migração de todos os recursos desejados;

[17] Estar *on-line* significa estar ligado e ou conectado a uma rede maior. Dentro do contexto da Internet, significa que está disponível para acesso.

- *Middleware*: Consiste em um ambiente que disponibiliza de forma simplificada facilidades em tarefas como projetar, programar e gerenciar aplicações distribuídas, fornecendo um ambiente simples, integrado e consistente, abstraindo através de uma camada a heterogeneidade e a complexidade do *cluster*, fornecendo assim, uma imagem única do sistema (SSI, *Single System Image*). Atualmente, existe uma grande discussão entre diversos grupos de trabalho em criar uma forma padronizada para determinar quais serviços devem compor essa camada. Entretanto, esse grupo de serviços é dividido de acordo com o tipo de aplicação na qual o *cluster* é destinado, visto que cada classe de aplicações possui necessidades e serviços básicos distintos.

4.1 Imagem única do sistema – SSI

O sistema de imagem única consiste na propriedade de esconder a complexidade envolvida em um ambiente distribuído de recursos, seja por *hardware* ou por *software*, de maneira que os usuários enxerguem o sistema em geral como se fosse um único recurso, ou seja, para o operador, o ambiente é visto de maneira globalizada, não importando em qual nó ele esteja fisicamente associado.

O SSI é considerado uma camada *middleware*, localizada entre o sistema operacional e o nível de usuário. O ambiente *middleware* é composto essencialmente por duas sub-camadas de infra-estrutura de *software*: a infra-estrutura de imagem única (SSI) e a infra-estrutura de disponibilidade. A primeira parte, SSI, está presente em todos os sistemas operacionais a fim de oferecer uma unificação dos recursos. Já a infra-estrutura de disponibilidade oferece serviços ao *cluster* como *checkpoint*[18], *failover* [19]automático, recuperação e tolerância à falhas para todos os nós do *cluster* (BUYYA, 1998).

[18] Método para conservar o estado de um trabalho, ou seja, é útil para recuperar um sistema, por exemplo, logo após um erro, o sistema retorna ao ponto definido, na qual seu funcionamento estava correto.

Conforme Pitanga (2003, p. 52), o SSI é definido como:

> [...] uma ilusão criada por *hardware* ou por *software*, que apresenta uma coleção de recursos que atuam como se fossem um só componente. Portanto, um recurso unificado poderosíssimo, ou de uma forma mais clara, a imagem única do sistema oferece o desempenho, serviços e funcionalidades que são realizados de modo distribuído, pela cooperação de múltiplos componentes, com os benefícios de uma única máquina massivamente paralela (MPP).

Uma demonstração do ambiente SSI pode ser observado na Figura 4.1 abaixo:

Figura 4.1: Abstração de um nó sob um ambiente SSI
Fonte: Adaptado de PITANGA, 2003, p. 53.

4.1.1 Características do SSI

De uma forma geral, algumas características podem ser apontadas como diferenciais com relação ao ambiente SSI. Segundo Dantas (2005) e Pitanga (2003) são apontadas:

- Os serviços podem ser solicitados a partir de qualquer nó, ou seja, provem uma transparência na execução das aplicações;

[19] Consiste na tarefa de transferir a operação de um componente falhado (por exemplo, um HD) a um componente similar, redundante para assegurar que a operação em andamento não seja interrompida.

- Não há necessidade da localização física dos dispositivos;

- Gerência do sistema pode ser reduzida significativamente;

- Aumenta a robustez da configuração uma vez que reduzem a intervenção de operadores;

- Permitem um modelo de processos com espaço globalizado para balanceamento de carga.

4.1.2 Pacotes de Softwares SSI

A importância do SSI pode ser levada em consideração devido a grande quantidade de pacotes de *softwares* disponíveis, tanto em níveis acadêmicos, que possuem a abordagem de utilização aberta, quanto ao nível comercial. No Quadro 4.1 a seguir pode ser visualizado alguns pacotes existentes a nível acadêmico e comercial. É importante compreender que tantos os sistemas acadêmicos quanto os comerciais, foram desenvolvidos para serem utilizados sob determinado modelo de arquitetura distribuída.

Quadro 4.1: Pacotes de *softwares* SSI

Nome do SSI	Local de desenvolvimento ou Fornecedor	Nível
OpenMosix	Hebrew University	Acadêmico
Glunix	Universidade de Berkeley	Acadêmico
OSCAR	Open Cluster Group	Acadêmico
CJava	IBM – Haifa Research Lab	Acadêmico
Snowflake	Dartmouth College	Acadêmico
Sprite	University of Berkeley	Acadêmico
Nomad	UFRJ – COPPE	Acadêmico
Future System	HP	Comercial
SSI	Compaq	Comercial
Memory Channel	DEC-HP	Comercial
Solaris MC	SUN	Comercial
Unixware	SCO	Comercial

Fonte: Adaptado de DANTAS, 2005, p.132 a 133.

4.2 Métricas para Classificação dos Clusters

Uma questão a ser definida no que diz respeito ao conceito de *clusters* é como classifica-lo junto às demais arquiteturas paralelas e distribuídas. Atualmente, não existe uma taxonomia globalmente aceita para este ambiente computacional, ao contrario das arquiteturas paralelas que dispõe da classificação de Flynn. No entanto, através da observação de alguns aspectos, é possível fazer uma classificação levando em conta alguns pontos relevantes (Figura 4.2): limite geográfico, a forma de utilização dos nós, o tipo de *hardware* e conexão entre os nós, os tipos de nós e os recursos utilizados pelas aplicações.

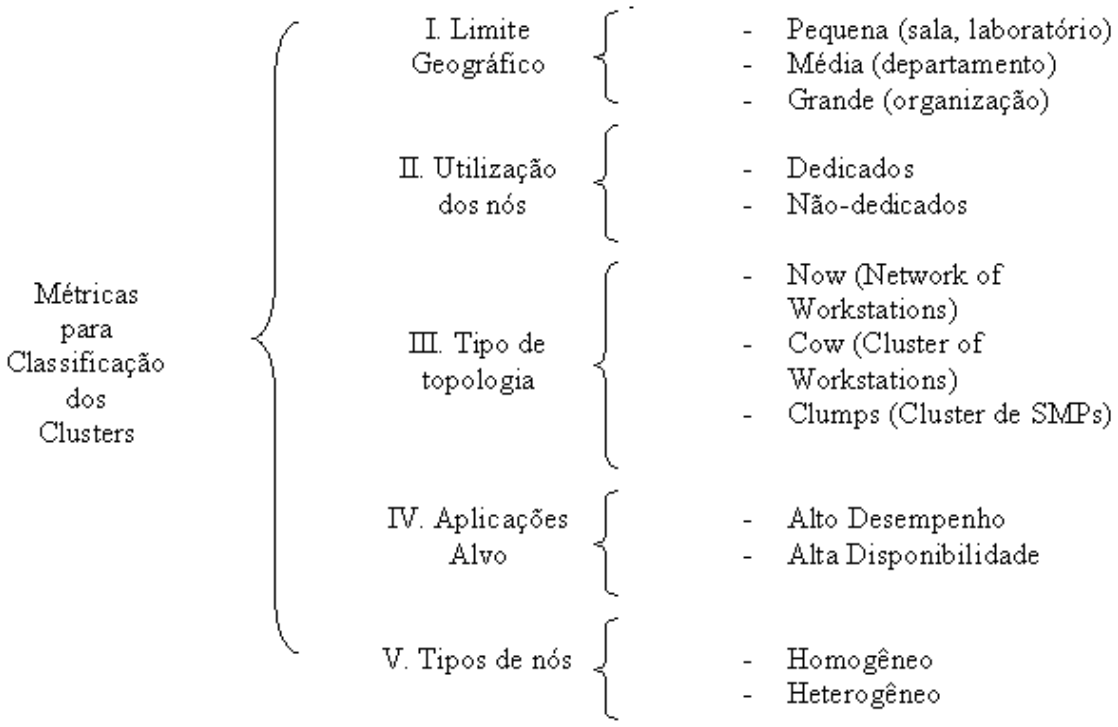

Figura 4.2: Métricas para classificação dos *clusters*
Fonte: DANTAS, 2005, p. 148.

4.2.1 Limite Geográfico

A maioria das implementações de *clusters* tem surgido em locais específicos, como salas e laboratórios, ou seja, em locais com certa limitação geográfica. Essas implementações podem variar desde uma simples interligação de computadores do tipo PC, através do uso de *hubs*[20] ou *switches*[21], até o uso de dispositivos especiais de ligação como *Myrinet, Memory Channel, Quadrics*. Uma vez montada a estrutura de um *cluster* localmente, a formação departamental tende a ser o próximo passo quando se pretende obter maior desempenho das aplicações. Por fim, quando um marco de sucesso é alcançado em um determinado departamento com o uso de *clusters*, a organização procura estabelecer condições para propagar o uso deste paradigma computacional. De uma maneira geral, quanto ao número de nós os *clusters* enquadram-se em:

- *Clusters* de Grupo: de 2 a 100 nós;
- *Clusters* Departamentais: de dezenas a centenas de nós;
- *Clusters* Organizacionais: com várias centenas de nós.

4.2.2 Utilização dos nós

Um dos aspectos mais relevantes a ser observado na implementação dos *clusters* é a definição de utilização dos nós. Assim, os *clusters* podem ser divididos quanto a sua

[20] Aparelho utilizado em redes de computadores para interligar diversas estações.
[21] É um dispositivo utilizado em redes de computadores para reencaminhar dados entre os diversos nós, sendo que cada nó corresponde a um segmento diferente.

propriedade em dedicados e não dedicados, sendo que esta distinção depende exclusivamente da sua aplicabilidade.

Enquadra-se como não dedicados quando as aplicações são executadas baseadas na ociosidade das estações de trabalho, ou seja, as máquinas podem operar normalmente como uma simples estação de trabalho, sendo que, quando não estão em operação, são utilizadas para executarem tarefas do *cluster*.

Neste tipo de ambiente, normalmente um dispositivo de rede como *hub* ou *switch* é responsável pela interconexão das máquinas, sendo que cada computador da rede tem um conjunto de aplicativos locais necessários para uso dos usuários localmente. Assim, para que o *cluster* seja formado, faz se necessário à utilização de pacotes de *software* instalados em todas as estações que deverão fazer parte do agrupamento. Exemplos de *software* interessantes deste ambientes são Condor e *OpenMosix*. A Figura 4.3 ilustra um ambiente não-dedicado.

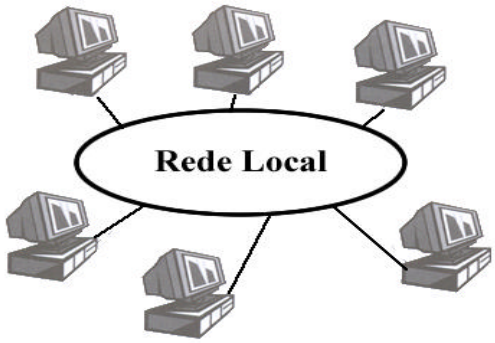

Figura 4.3: Ambiente de *cluster* não dedicado
Fonte: DANTAS, 2005, p. 155.

De acordo com Dantas (2005, p 154), "um ambiente de *software* para ser apropriado para a configuração não-dedicada deve levar em consideração as possíveis diferenças de *hardware* (exemplos são os processadores, as memórias e dispositivos de armazenamento), sistemas operacionais e carga de utilização das máquinas".

Por outro lado, diz-se que um *cluster* é dedicado quando seus nós são utilizados exclusivamente para a computação paralela. Essa configuração apresenta vantagens em

relação às não-dedicadas quanto à execução de aplicações criticas, ou seja, através desta configuração consegue-se uma melhor execução de aplicações vitais para a organização. Neste ambiente, é possível efetuar uma melhor escolha na distribuição dos aplicativos para processamento, pois existe um prévio conhecimento da forma em que funcionam as configurações e quais aplicativos tem prioridade de execução. Nestas configurações, todos os nodos são conectados através de um dispositivo de rede do tipo *switch,* sendo que os computadores não dispõem normalmente de teclados, mouses e monitores específicos para cada estação. Esse dispositivo de rede, em conjunto com um sistema de imagem única, pode criar uma abstração do ambiente dedicado separado do restante da rede da corporação.

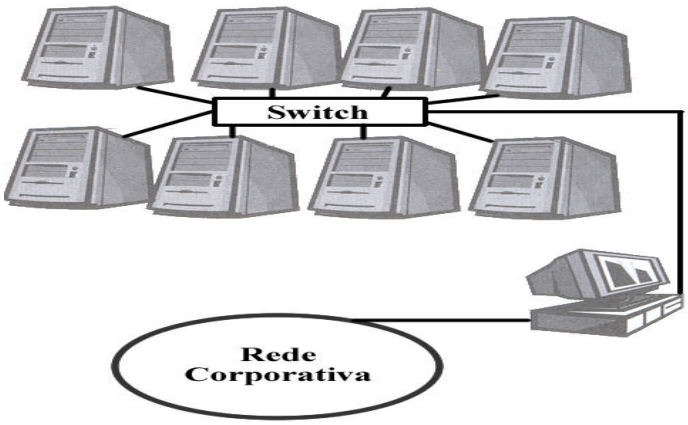

Figura 4.4: Ambiente de cluster dedicado
Fonte: DANTAS, 2005, p. 165.

4.2.3 Tipo de Hardware

Quanto ao tipo de *hardware* utilizado, os *clusters* podem ser classificados da seguinte forma:

- *Network of Workstations* (NOW): constituem pelo uso de estações de trabalho distribuídas em uma rede local para compor o ambiente do *cluster*. Alguns

pesquisadores ainda consideram *Pile of Pcs* (PoPs, pilhas de PCs) e *Cluster* of PCs (CoPs, Cluster de PCs).

- *Cluster of Workstations* (COW): geralmente são constituídas de máquinas homogêneas e dedicadas à execução de tarefas específicas. Sua implementação requer uma rede específica para a conexão entre as máquinas.

- Clumps (*Cluster of sMPS*): enquadra-se nesta classificação um ambiente composto por máquinas de arquitetura SMP.

Dentro deste contexto, normalmente, quando o *cluster* é formado por multicomputadores ou multiprocessadores específicos, apenas uma avaliação do sistema de interconexão em relação ao material fornecido pelo fabricante pode ser realizada. Entretanto, em configurações COW é possível analisar a melhor forma de conexão entre os nodos a fim de sugerir o dispositivo mais apropriado para a comunicação. Vale lembrar que, os *clusters* podem ser projetados para montar ambientes de memória compartilhada e distribuída. Assim, a escolha do tipo de interconexão é um ponto determinante para um bom desempenho do *cluster*.

4.2.4 Aplicações Alvo

Com relação às aplicações, os *clusters* são divididos em duas categorias básicas: Alta Disponibilidade (HA – *High Availability*) e Alta Performance de Computação (HPC – *High Performace Computing*). É importante salientar, que em alguns casos é possível à utilização das duas categorias em conjunto.

4.2.4.1 Alta Disponibilidade

Caracteriza por aplicações que não podem sofrer interrupções. As aplicações de alta disponibilidade estão diretamente ligada a nossa crescente dependência dos computadores, pois na atualidade elas possuem um papel essencial nas empresas, principalmente naquelas em que oferecem algum tipo de serviço critico como *e-business*[22]*, sites web*, banco de dados, entre outros. "A questão de *clusters* de alta disponibilidade é que o aplicativo é de tal importância que você toma etapas extras para garantir que ele está disponível" (BOOKMAN, 2003, p.7), ou seja, tem como função deixar um sistema no ar vinte e quatro horas por dia, sete dias por semana, a fim de evitar paradas não planejadas que podem influenciar diretamente a qualidade do serviço e conseqüentemente levar a algum tipo de prejuízo.

Esta disponibilidade é conseguida através de *hardware*, como sistemas de redundância de discos (*raid*), placas e fontes redundantes, sistemas de redes que provem caminhos alternativos em caso de quebra de algum *link*, bem como a utilização de *softwares* que verificam o funcionamento destes equipamentos (PITANGA, 2003). A Figura 4.5 mostra um exemplo de uma simples configuração de um *cluster* do tipo HA.

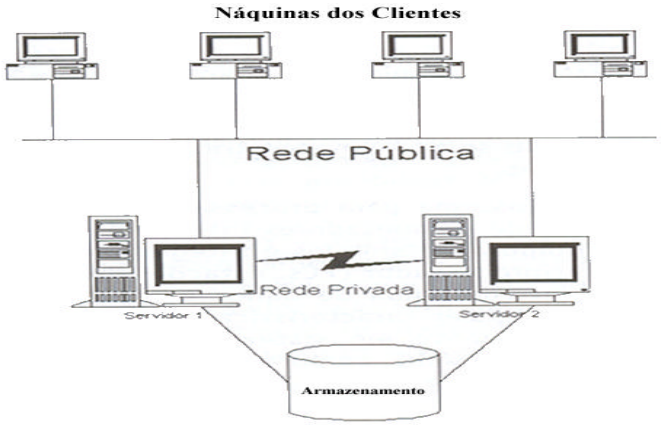

Figura 4.5: Um simples *cluster* de alta disponibilidade
Fonte: PITANGA, 2004, p. 25.

[22] É o termo que se utiliza para identificar os negócios efetuados por meios eletrônicos, geralmente na Internet.

4.2.4.2 ALTA PERFORMANCE DE COMPUTAÇÃO

Este tipo de cluster visa obter o máximo do poder computacional a fim de realizar uma instrução no menor tempo possível. Aliado com algoritmos paralelos eficientes, refere se a distribuição da carga de trabalho entre os recursos computacionais disponíveis em um ambiente, melhorando o desempenho das aplicações.

De uma maneira geral, quando se fala em processamento de alta performance, que pode ser traduzido em processamento paralelo e distribuído, logo é associado a supercomputadores, que tem um custo elevado e são difíceis de serem operados. No entanto, atualmente, um ambiente de *clusters* com seus inúmeros processadores e memórias, pode ser um local ideal para o uso destas aplicações. Neste tipo de *cluster*, é necessário que haja monitoração constante da comunicação e mecanismos de redundância. Caso contrário, qualquer falha pode interromper o funcionamento do *cluster* (ALECRIM, 2004).

4.2.5 Tipos de Nós

Quanto à configuração dos nós, os *clusters* podem ser classificados como homogêneos e heterogêneos. Um *cluster* dito homogêneo, refere-se aqueles que possuem arquiteturas de *hardware* similar e executam o mesmo sistema operacional. Neste tipo de configuração, o esforço de interoperabilidade entre os processos é inexistente. Uma vantagem apresentada nesta configuração é no que diz respeito à reposição de peças, uma vez que todos os equipamentos são iguais. Quanto aos heterogêneos, são caracterizados por possuírem diferentes arquiteturas e executarem diferentes sistemas operacionais. Neste caso, há um esforço maior a fim de garantir que a heterogeneidade seja transparente ao usuário, bem como o uso de máquinas e pacotes de *softwares* compilados em SOs distintos, sejam executados de maneira correta (DANTAS, 2005).

4.3 Redes de conexão

Um aspecto fundamental na implementação de *clusters* de computadores é a maneira na qual os componentes deste ambiente estão interligados. As redes de conexão estão diretamente relacionadas à performance do *cluster*, sendo responsáveis pelas ligações entre as unidades de processamento e memórias nos computadores com arquitetura paralela.

> A escolha de uma melhor solução de interconexão para um *cluster* é muito importante, e pode ser muito difícil a sua construção. Se não houver um bom balanceamento entre os nós de processamento e a comunicação entre eles, os nodos poderão disperdiçar recursos enquanto estiverem aguardando por dados. Por outro lado, dependendo do tipo de rede escolhida, pode tornar-se um percentual muito grande no custo do projeto (PITANGA, 2003, p. 121).

Hoje em dia, existem diversos meios de comunicação de alto desempenho que estão em voga no contexto dos *clusters* computacionais, dois quais encontram-se *Fast Ethernet, Gigabit, Myrinet, Asynchronous Transfer Mode* (ATM), *Scalable Coherent Interface* (SCI), entre outras. Por ser uma área por demais ampla e, não ser o foco deste trabalho, não serão apresentadas as características específicas com relação a cada tipo de rede de conexão. Entretanto, um aprofundamento teórico com relação aos tipos de redes existentes é importante para a implementação da melhor solução de interconexão entre os nós do *cluster*.

4.3.1 Metas a serem alcançadas em redes de comunicação para *cluster*

A fim de obter um melhor desempenho na comunicação entre os nós, alguns pontos são de extrema importância e devem ser levados em consideração na montagem de um *cluster* com relação à rede de comunicação (PITANGA, 2003):

- Largura de banda: consiste na capacidade de transporte de dados que pode ser transferida em uma rede em um determinado período. É medida em *bits* por

segundo (bps), sendo que quanto maior a largura de banda, melhor será o desempenho da rede. Por exemplo, se tivermos um servidor *web* com largura de banda de 100 Kb/bps, dez acessos simultâneos poderão ter uma velocidade de *donwload* [23]do *site* de 10 Kb/bps;

- Vazão: define a número de pacotes de dados livres de erros que trafegam através de uma rede em unidade de tempo. Se fosse considerado pela teoria, a vazão deveria ser igual à capacidade da rede, o que na prática, difere um pouco. Logo, a vazão da rede depende muito do meio físico empregado, sendo que o mesmo deve estar adaptado à soma de todos os dados que os nós do *cluster* estão preparados para transmitir em um intervalo de tempo. Na realidade, existem diversos fatores, como instalações elétricas mal feitas, materiais de má qualidade, ruídos, entre outros fatores, que ocasionam retransmissões e interferem sensivelmente no fluxo dos dados, assim como na performance do cluster.

- Latência: é a medida definida como o tempo decorrido entre o início da transmissão de um pacote de dados até a sua recepção na outra extremidade. Conforme Pitanga (2003, p. 124), "fatores como o meio físico empregado na transmissão, tempo de comutação dos pacotes (no caso *switches*, roteadores, *hubs* e *bridges*[24]) e protocolos de acesso ao meio tendem a aumentar o atraso na comunicação";

- *Multithreading*: corresponde à capacidade de uma rede enviar múltiplas mensagens concorrentemente, tendo também à habilidade de um nó utilizar os dispositivos sobre contextos distintos. Algumas placas de redes conseguem se comunicar com vários processos de alto nível para a troca de mensagens.

[23] Ato de transferir o arquivo de um computador remoto para o seu próprio computador, usando qualquer protocolo de comunicação.
[24] Dispositivo de informática que faz a ligação entre duas redes distintas.

4.4 Áreas de Uso

A utilização de *clusters* é aplicada em diversos segmentos. Pode ser utilizada basicamente em qualquer lugar que tenha um grande problema computacional, em que o processamento em paralelo seja considerado uma vantagem. Normalmente são aplicações muito grandes que necessitam de enormes quantidades de memórias e poder de processamento que utilizam este ambiente computacional. Estas aplicações incluem ambientes de trabalho cooperativo, videoconferência, indústria de entretenimento, servidores *web*, dentre outros. Alguns exemplos de aplicabilidade de *clusters* são vistos a seguir (PITANGA, 2004):

- Segurança: oferece benefícios em qualquer processo de identificação de quebra na segurança e verificação de possíveis soluções;

- Servidores *Web*: Possibilita a distribuição de carga de tarefas e aumenta a capacidade de resposta do servidor;

- Banco de Dados: Pode reduzir consideravelmente o tempo de consulta a grandes bases de dados;

- Computação Gráfica: o tempo de processamento nesta área é determinante para a evolução da qualidade dos projetos. Um *cluster* nesta área pode diminuir o tempo de renderização de imagens na produção de um filme, por exemplo;

- Aerodinâmica: estudo de novas capacidades tecnológicas e econômicas na pressão enfrentada pelas aeronaves como o lançamento de naves espaciais;

- Análise de elementos finitos: utilizado no cálculo de pontes, navios, aviões, grandes construções, etc;

- Sensoramento remoto: análise de imagens de satélites para a obtenção de informações;

- Automação e inteligência artificial: visão por computador, reconhecimento de voz, máquinas de inferência, reconhecimento de padrões;

- Engenharia Genética: projeto Genoma;

- Oceanografia e astrofísica: estudo de recursos dos oceanos e formação da terra, dinâmica das galáxias;

- Previsão do tempo: maior agilidade e precisão do que quando gerado em computadores seqüências;

- Exploração sísmica: Utilizado pelas companhias petrolíferas para determinar locais de poços de petróleo;

- Pesquisas militares: simulação de efeitos de armas, projetos de armas nucleares, geração automática de mapas, acompanhamento de submarinos submersos;

- Segurança de reatores nucleares: treinamento através de simulação de operações, análises de condições dos reatores;

- Problemas de pesquisa básica: em química, física e engenharia, tais como estatística, mecânica quântica, dinâmica molecular, analise de trajetória de partículas, entre outros.

4.5 Cluster Beowulf

O projeto *Beowulf* foi idealizado em 1994 pela *National Aeronautics and Space Administration,* no *Goddard Space Flight Cente*r (NASA-GSFC) com a finalidade de suprir a crescente e elevada capacidade de processamento encontrada na ciência espacial a um custo mais acessível. A origem do nome *Beowulf* vem de um dos poemas épicos mais antigos da língua inglesa, na qual "conta à história de um cavaleiro inglês, de grande força e valentia em sua saga para derrotar o monstro de *Grendel*" (PITANGA, 2003, p. 36).

Este projeto consistia na utilização de componentes de *hardware* comuns, tipo PCs, de fácil aquisição e preço acessível, dotado de sistema operacional GNU/Linux, que já permitia o uso de troca de mensagens (MPI) e gerenciamento de máquinas paralelas (PVM). Esse projeto foi identificado como uma alternativa importante pela NASA em suas aplicações de missão critica, atendendo as expectativas da corporação em utilizar equipamentos "rápidos e baratos". Na primeira versão do *Beowulf,* foi utilizada a distribuição *Linux Slackware* para implementação da arquitetura. Atualmente, utilizam-se principalmente distribuições Linux REDHAT ou similares, pois apresenta maiores facilidades para o gerenciamento de pacotes e disponibilidade de atualizações, no entanto, pode ser usada sob qualquer distribuição ou SO que gerencie aplicações paralelas.

Estas facilidades contribuíram para o sucesso e popularização deste tipo de arquitetura, sendo logo implementado em muitas universidades e instituições de pesquisa. No Brasil, universidades como UFRGS, USP e a UFRJ, entre outras, efetuam pesquisas neste tipo de *cluster*. Entre empresas estatais, destaca-se a Petrobrás, que implementou esta solução em 1999.

Um *cluster* do tipo *Beowulf* é considerado especificamente dedicado ao processamento paralelo e enquadra-se na taxonomia de Flynn como MIND. Possuí uma arquitetura de diversos computadores formados por um nó controlador (*frontend*) e nós clientes (*backends*), interligados através de uma rede *Ethernet* ou outra tecnologia de rede qualquer, dedicada exclusivamente para a passagem de mensagens, com um sistema

operacional que implemente estruturas PVM e/ou MPI, para realizar uma comunicação que otimize o processamento paralelo. O nó controlador tem como função controlar, monitorar e distribuir tarefas entre o *cluster*, além de atuar como um elo entre ele e os usuários. Existe a possibilidade de ter mais de um nó controlador ou mestre no sistema. Já os nós clientes ou escravos, têm como tarefa única e exclusiva processar as tarefas enviadas pelo nó controlador. Neste *cluster*, não existe a necessidade da utilização de teclados e monitores em todos os terminais, sendo possível implementar inclusive terminais sem a utilização de discos rígidos. Neste caso, o *boot* é efetuado a partir do nó mestre, que armazena informações sobre a configuração da rede (PITANGA, 2003; BEOWULF, 2004). Uma arquitetura básica de um cluster *Beowulf* é demonstrada na figura 4.4 da sessão anterior.

4.5.1 Vantagens

Quanto à utilização do *cluster* Beowulf, podemos citar algumas vantagens:

- Independência de fornecedor, ou seja, sistema pode ser construído usando dispositivos de diferentes origens;
- Continuidade tecnológica, pois componentes atualizados são encontrados no mercado com facilidade;
- Flexibilidade de configuração;
- Escalabilidade;
- Alta disponibilidade.

4.5.2 Desvantagens

Seguem a seguir alguns problemas que podem ocorrer na implementação:

- Nem todas as aplicações podem ser paralelizadas;
- Difícil programação;
- Ocorrência de gargalos tradicionais em comunicações, como latência e largura de banda;
- É preciso experiência em ambientes de redes para a elaboração e configuração do *cluster*.

4.6 Cluster OpenMosix

O *OpenMosix* é um projeto de código aberto oriundo do projeto *Multicomputer Operating System Unix* (MOSIX) da *Hebrew University*. O *OpenMosix* é uma extensão do *kernel* do Linux para que os ambientes paralelos possuam uma imagem uniforme. A extensão provida pelo *OpenMosix*, de acordo com alguns pesquisadores do projeto, faz com que uma rede comum de computadores executando estes pacotes, venha a se tornar um supercomputador para aplicações executadas sob o sistema operacional Linux (DANTAS, 2005). Entretanto, cabe ressaltar ao leitor, que existe total incompatibilidade entre o *Mosix* e o *OpenMosix*.

> O projeto traz uma nova visão para a computação baseada em *cluster* com GNU/Linux. Esse permite a construção de um sistema não dedicado de alta performance, escalável com componentes genéricos, em que o crescimento do sistema não introduz nenhum problema relacionado à sobrecarga da performance. A principal vantagem do projeto sobre outros sistemas em *cluster* é a capacidade de resposta em conformidade das exigências de execução dos recursos imprevisíveis e irregulares dos diversos usuários conectados ao sistema (PITANGA, 2003, p. 241).

Após a instalação dos pacotes do *OpenMosix*, os nós da configuração começam a comunicação entre si, adaptando a carga de trabalho submetida para execução. Sendo assim, a principal característica deste modelo de *cluster* é o método de migração de processos, na qual é equilibrada a carga entre os computadores de maneira tal que se consiga executar mais processos do que se fosse executado em apenas uma estação, auxiliando diretamente aplicações dotadas de muitos processos independentes (OPENMOSIX, 2006).

Assim, processos que são iniciados em um nó que esteja muito ocupado são transferidos para uma outra máquina com maior disponibilidade de recursos. Diferentemente de um *cluster Beowulf*, neste modelo todos os processos são efetivados sem um controle central, não havendo nenhuma relação mestre/escravo entre os nós. Para que isso seja possível, cada nó do *cluster* trabalha de forma independente e autônoma, tornando uma configuração dinâmica e escalável. Em outras palavras, tão logo um novo membro faça parte da configuração, seus recursos e serviços podem ser disponibilizados aos demais participantes do grupo, causando o mínimo de impacto sobre o sistema.

Para a instalação de um *cluster Openmosix*, é necessário a existência de no mínimo duas máquinas, com uma rede de interconexão, existindo três maneiras de configuração, cada qual com suas vantagens e desvantagens.

- ***Single Pool***: neste tipo de configuração, todas as máquinas, servidores e estações, são reconhecidas como integrantes do *cluster*, onde todas as maquinas podem migrar seus processos para outros nós do agrupamento.

- ***Server Pool***: neste caso, apenas os servidores compõem o *cluster*, deixando as estações de fora, sem receberem os processos remotos.

- ***Adaptative Pool***: com esta configuração, os servidores ficam compartilhados enquanto as estações podem ou não fazerem parte do sistema. Isso é interessante em casos que, por exemplo, uma estação opere normalmente como outro computador qualquer em horário de expediente, mas ao fazer *logoff* do sistema, um *script* é executado, ativando a estação a fazer parte do *cluster*.

Para atingir um melhor desempenho, este modelo de *cluster* é controlado através de algoritmos de Balanceamento Dinâmico de Carga e algoritmo de Anunciação de Memória. A finalidade desses algoritmos é responder dinamicamente às variações da utilização dos recursos pelos diversos nós, garantindo a escalabilidade do mesmo, tendo ele muitos ou poucos nós. O algoritmo de balanceamento de carga é utilizado quando um programa que está sendo executado em um determinado nó está muito sobrecarregado, fazendo uma migração desta atividade ou parte dela para um nó com mais recursos disponíveis. Com isso, o sistema ganha maior desempenho. Esse algoritmo é executado de maneira independente em todos os nós do *cluster*. Por outro lado, existe um algoritmo que é utilizado para evitar um total esgotamento da memória, cujo objetivo é tentar ocupar, quando possível, a memória do *cluster* na sua totalidade, evitando a ocorrência de paginações ou utilização de memória virtual.

Para a migração de tarefas entre os nós, o *OpenMosix* implementa os processos de modo preemptivo, ou seja, os processos são divididos em duas partes: a que é transferida para ser executada em outro nó, denominada de contexto do usuário (*remote*), composta pelo código do programa, a pilha, os dados, os mapas de memória e o estado dos registradores do processo. A outra parte contém todas as descrições dos recursos dos quais o processo está utilizando ou está alocado para ele, bem como a pilha do sistema que controla a execução do programa. Essa parte é conhecida como contexto do sistema (*deputy*). Todavia, "o *remote* pode ser transferido tantas vezes que se fizer necessário, para qualquer nó do *cluster*, mas o *deputy* jamais pode ser removido" (PITANGA, 2003, p. 251).

Esse processo de migração pode ser por meio automático ou manual. No caso de manual, o administrador indica qual processo deverá migrar para determinado nó, podendo ainda indicar a capacidade máxima que um determinado nó deve tolerar antes de iniciar o

processo de migração para outro nó do conjunto. Entretanto, existem muitos processos que podem migrar para outros nós, todavia, isso não quer dizer que os mesmos se beneficiarão com esta migração, ou seja, existem certas aplicações que podem se beneficiar e aumentar sua performance com a aplicação desta solução. Porém, por outro lado, existem aquelas que não são recomendadas para este ambiente.

4.6.1 Vantagens

Dentre as aplicações que se beneficiam com o *OpenMosix*, destacam-se:

- Aquelas cujos processos requerem longos tempos de execução e baixo volume de comunicação, como aplicações científicas, engenharia, etc;
- Grandes compilações;
- Banco de dados que não usam memória compartilhada;
- Processos na qual é possível fazer migração manualmente.

4.7.2 Desvantagens

Algumas aplicações podem não obter vantagens utilizando este *cluster*, entre elas:

- Aplicações que requerem memória compartilhada;
- Aplicativos que utilizam grande comunicação interprocessos;

- Aplicações que dependem de um *hardware* específico presente em algum nó do *cluster*;

- Aplicativos com grandes quantidades de *threads*[25];

- Aplicações que utilizam um único processo, por exemplo, um *browser*[26].

[25] As threads são múltiplos caminhos de execução que rodam concorrentemente na memória compartilhada e que compartilham os mesmos recursos e sinais do processo pai. Uma thread é um processo simplificado, mais leve para o SO, sendo fácil de criar, manter e gerenciar.
[26] Denominação de um programa utilizado para abrir e exibir as páginas da web.

5 BALANCEAMENTO DE CARGA

Uma empresa fictícia decide oferecer novos serviços em seu *site* via *web*. O novo serviço consiste em fazer conversões *on-line* de documentos dos mais variados formatos (*.doc, *.xls, *.txt) em arquivos *.pdf. Entretanto, para que o serviço seja de qualidade, é necessário que o tempo de conversão leve o menor tempo possível. Supondo que a empresa disponha do mais recente e melhor sistema multiprocessador e, tudo está funcionando perfeitamente nos primeiros dias do lançamento. No entanto, devido ao sucesso do novo serviço, o número de acessos ao sistema aumenta exponencialmente, sobrecarregando o servidor, de modo que o acesso ao sistema fica impossibilitado ou muito lento, não atendendo as expectativas da direção da empresa. Então, o que fazer para resolver este problema? Uma solução possível neste caso, seria fazer uma redistribuição dos serviços para mais de um servidor, através de um *cluster* de balanceamento de carga, evitando que apenas um servidor responda a todas as solicitações.

Em linhas gerais, balanceamento de carga consiste em dividir aplicativos ou processamentos através da rede para outras unidades de processamento. "Ainda que a maioria dos aplicativos de equilíbrio de carga gerencie servidores *web*, o equilíbrio de carga pode distribuir praticamente qualquer serviço através de qualquer quantidade de computadores e localizações geográficas" (BOOKMAN, 2003, p. 104).

Na maioria das vezes, em sistemas que é possível fazer a distribuição de tarefas, pode ocorrer de alguns computadores terminarem suas tarefas antes de outros, ficando os mesmos ociosos. Causas para isso podem ser que uns processadores são mais rápidos que

outros, ou às vezes não se tem uma carga igualmente distribuída entre eles, ou ainda, podemos ter as duas situações. O ideal e desejável, seria que todos os servidores que compõe a rede do *cluster*, trabalhem continuamente na execução das tarefas, a fim de concluí-las no menor tempo possível.

A tarefa de pegar a entrada de carga e distribuí-la através do *cluster* é apenas a primeira etapa. A questão que surge é como fazer para distribuir as tarefas a serem processadas através dos diferentes servidores. De acordo com Bookman (2003, p. 105), a maneira na qual será feita à distribuição da carga de trabalho "[...] depende do tipo de serviço que é oferecido, do tamanho de sua rede e do número de computadores em seu *cluster*". Dependendo da topologia do *cluster*, existem diversos algoritmos capazes de fazer a melhor distribuição das tarefas. Dentre estes, serão abordados a seguir Circulo de Ocultação e Equilíbrio Baseado em Carga.

- Circulo de Ocultação – Consiste num dos métodos mais fáceis para distribuição de carga. Embora existam maneiras mais eficazes, este método pode ser obtido através do uso do *Domain Name System* (DNS – Servidor de Nomes). Supondo que existam três servidores *web*, A, B e C respectivamente, utilizando este método, a primeira solicitação é repassada para o servidor A, a segunda para o servidor B e assim sucessivamente, até chegar ao último servidor, retornando o ciclo novamente ao servidor A. Sendo assim, este algoritmo funciona bem quando há maquinas processadoras idênticas, não sendo muito eficaz quando existem computadores com capacidades diferentes. Entretanto, existem outros meios capazes de contornar este problema, a qual da precedência especial a servidores seletos. Neste caso, se o computador com maior capacidade de processamento for o servidor C, por exemplo, as requisições seriam A, B, C, C, A, B, C, C respectivamente (BOOKMAN, 2003).

- Equilíbrio baseado em carga - Neste caso, a solução de equilíbrio de carga é medida através de processos ou conexões por servidor. Assim, um *cluster* com este método toma as solicitações e as distribui através da rede. Baseado na quantidade de carga que cada servidor tem no momento, as solicitações são repassadas através dos recursos disponíveis no *cluster*. Em servidores que utilizam este tipo de algoritmo

baseado em carga, deve existir um tipo de *daemon* sendo executado em cada máquina do *cluster*, com a função de mostrar ao servidor principal, denominado mestre, informações sobre os recursos determinados, permitindo que o servidor mestre tenha conhecimento para a tomada de decisões sobre cada solicitação subseqüente.

5.1 Gerência de Recursos

Fazer o balanceamento de carga não é tão simples quanto se imagina. Considerando que podemos ter múltiplos usuários utilizando o *cluster*, sendo que algumas aplicações funcionam bem neste ambiente, outras não utilizam todos os recursos disponíveis, deixando os mesmos ociosos, alguns nós possuem mais memórias que outros. Também, deve ser levado em consideração à prioridade de execução de certas tarefas, bem como a monitoração de uma estação caso a mesma venha a falhar. Esses e outros problemas que possam a vir a ocorrer e que dificultam a operacionalidade destes ambientes, podem ser resolvidos através da utilização de *softwares* de gerenciamento de recursos (CMS – *Cluster Management Software*). Segundo Pitanga (2004, p. 212), "esse *software* maximiza a entrega dos recursos às tarefas, de acordo com as requisições dos usuários e com a política de utilização adotada".

Esses *softwares* são designados para administrar e gerenciar tarefas solicitadas ao *cluster*, utilizando algumas técnicas de enfileiramento (*queuing*), escalonamento (*scheduling*), monitoramento (*monitoring*) e contabilidade.

5.1.1 Enfileiramento

De acordo com Pitanga (2004, p. 213), enfileiramento é "[...] ato de coletar todos os processos que juntos vão ser colocados em execução em um conjunto de recursos". Essa função consiste em montar filas específicas de tarefas a serem executadas, de acordo com as solicitações dos usuários que são submetidas ao *cluster*, até que os recursos sejam disponibilizados. É empacotada em um *script* denominado "*batch job*", composto por um conjunto de diretivas de recursos, como a quantidade de memória ou processadores necessários, e a descrição da tarefa, onde deve conter informações necessárias para o inicio da operação, tais como o nome do arquivo que será executado, listagem de dados necessários, variáveis e argumentos a serem processados.

5.1.2 Escalonamento de tarefas

Escalonamento é um processo de tomada de decisões na qual se preocupa em alocar recursos limitados para tarefas ao longo do tempo, e tem como meta a otimização de uma ou mais funções e ou objetivos. Em outras palavras, possui o objetivo de escolher a melhor maneira de dividir as tarefas a fim de minimizar o tempo de execução de uma atividade computacional (GUIA, 2006).

Escalonadores de tarefas são componentes de software comumente integrados a sistemas operacionais paralelos e/ou distribuídos e que têm como função a distribuição de trabalho computacional as unidades de processamento integrantes do sistema, de modo a maximizar o desempenho global do processamento realizado, isto é, promover o balanceamento de carga entre as unidades de processamento envolvidas. Sendo assim, o principal objetivo do escalonador é diminuir o tempo de conclusão de todas as tarefas e balancear a carga dos processos no *cluster*. Para que isso seja possível, alguns conceitos básicos devem ser levados em consideração, como o poder de processamento e a ociosidade das máquinas.

Nos sistemas homogêneos, o problema de balanceamento de carga pode ser minimizado com a divisão de um determinado trabalho computacional em partes iguais e que possam ser distribuídas e executadas por unidades de processamento, supostamente idênticas, do sistema. Por outro lado, em sistemas heterogêneos, o balanceamento de carga é considerado muito mais complexo e, neste caso, o escalonador de tarefas ganha especial importância. Dentro deste contexto, existem duas classes principais de escalonadores: os determinísticos (ou estáticos) e os não-determinísticos (ou dinâmicos).

Temos um escalonamento estático, quando a decisão de escalonar as tarefas é realizada em tempo de compilação, ou seja, o número de processos, meios de comunicação, granularidade[27], dentre outras informações, são conhecidas antes da sua execução. Neste método, um processo iniciado em um processador, roda no mesmo até a sua finalização. Sendo assim, uma estimativa imperfeita de alguma parte integrante do sistema ou a ocorrência de eventos inesperados que podem afetar o desempenho do sistema durante a execução de alguma tarefa previamente escalonada, podem ocasionar decréscimos significativos no desempenho global.

Em contrapartida, temos o escalonamento dinâmico, cuja decisão sobre distribuição dos processos entre os processadores é realizada em tempo de execução. Neste caso, não há muito ou praticamente nenhuma necessidade de conhecer os processos antes de sua execução, o que torna o sistema capaz de se ajustar conforme o comportamento do sistema durante a execução dos programas. Entretanto, embora as decisões ainda se baseiem em estimativas de desempenho do sistema e, conseqüentemente, estimativas imprecisas ainda podem significar um escalonamento ineficiente, as conseqüências de um mau escalonamento não são tão grandes quanto seriam no caso do escalonamento estático. Neste caso, atrasos ou adiantamento na conclusão de uma tarefa, podem ser utilizados em tempo de execução para

---

[27] Granularidade está relacionada ao conceito de decomposição de dados e tarefas. Refere-se à quantidade de computação em uma unidade básica de trabalho, caracterizada pela razão entre a computação e a comunicação entre os processos, podendo ser fina ou grossa. A granularidade fina têm origem na alta taxa de comunicação e pouca tarefa computacional. Quando ocorre o inverso, alta tarefa computacional e pouca comunicação, têm-se a granularidade grossa.

um re-escalonamento das tarefas ainda pendentes. Dentro do escalonamento dinâmico, temos uma sub-divisão: Escalonamento dinâmico distribuído e escalonamento dinâmico não-distribuído.

Denomina-se escalonamento dinâmico não-distribuído quando um processador é responsável pelas decisões de todo o sistema, ou seja, os outros processadores atuam apenas como coletores de informações locais, passando as informações para o processador central, atuando como um esquema mestre-escravo. Por outro lado, no escalonamento dinâmico distribuído, o algoritmo é executado em todos os nós, onde cada nodo possui informações sobre a carga de trabalho das demais estações. Este modelo de escalonamento é utilizado pelo *cluster* do *OpenMosix*, podendo ser classificado ainda como cooperativos e não-cooperativos.

Em um ambiente cooperativo, todas as decisões individuais devem levar em consideração um objetivo comum para o sistema como um todo, ou seja, tem uma política pré-definida, onde os processadores e processos colaboram entre si. Por outro lado, em um ambiente não-cooperativo, as decisões são tomadas de forma independente por cada processador, sem haver preocupação sobre os resultados que isso porventura possa acarretar sobre o sistema em geral.

5.1.3 Monitoração

A monitoração disponibiliza informações úteis aos administradores, usuários e ao sistema de escalonamento sobre o andamento das tarefas e dos recursos (PITANGA, 2004). Sub-divide se em três tópicos críticos na monitoria de recursos:

- Quando os nodos estão inativos;

.

- Quando os nodos estão ocupados executando alguma tarefa;

- Quando uma tarefa for concluída.

5.1.4 Contabilidade

Consiste na coleta dos recursos utilizados pelos aplicativos que estão em execução no *cluster*, bem como o total de recursos consumidos pelas tarefas. Essa coleta de informações pode ter como resultado uma variedade de objetivos, tais como relatório de produção semanal, executar alocações de recursos por projetos, obter melhoras na política de escalonamento, assim como calcular futuras alocações de recursos, antecipar exigências relacionadas a aplicativos, etc. Além disso, outras informações podem estar disponíveis e vai depender exclusivamente do *software* de gerenciamento de recursos utilizado.

6 ANÁLISE DE PERFORMANCE

Quando um novo produto é lançado, para que o mesmo seja aceito, é necessário que se haja qualidade. Essa necessidade de aceitação faz com que sejam utilizados indicadores que comprovem o sucesso do mesmo. Essa premissa é valida também para a área de tecnologia, na qual novos lançamentos devem ser testados antes de sua introdução no mercado. Entretanto, muitas vezes, para ser feito a avaliação de um sistema real, tem-se um custo muito alto. Por exemplo, como saber se um projeto de uma determinada máquina será capaz de resolver um determinado problema, com um desempenho mínimo desejável? Neste caso, não é aceitável chegar ao final do projeto para saber se o mesmo alcançará o seu objetivo (CECHIN e NETTO, 2001).

Dentro deste contexto, com relação à área computacional, algumas técnicas e ferramentas podem ser utilizadas a fim de avaliar o desempenho dos mesmos, fazendo comparativos numéricos que podem ser usados para estimar comportamentos futuros de um sistema sob determinada situação, auxiliando na tomada de decisões antes que eles sejam realmente utilizados.

6.1 Técnicas de avaliação

A necessidade de provar novos experimentos dentro da comunidade científica em geral, trouxe a tona necessidades naturais de criar modelos matemáticos quer provem seus fundamentos de estudo (PITANGA, 2003). Geralmente, estes modelos são difíceis de serem obtidos sem o auxilio de uma ferramenta ou uma técnica de avaliação. Assim, foram criadas inúmeras técnicas e ferramentas que são capazes de emular diversas situações para os mais variados tipos de áreas, a fim de fazer uma análise mais detalhada sobre determinado assunto.

No caso de avaliação de desempenho computacional, estas técnicas podem ser divididas em técnicas elementares e técnicas indiretas. Diz-se que uma técnica é elementar quando a mesma é aplicada diretamente sobre os sistemas reais. Neste caso, há a necessidade que o sistema a ser avaliado exista para poder ser submetido aos testes, não podendo ser usado durante o seu desenvolvimento. Porém, quando sua aplicação for adequada, pode fornecer resultados bastante próximos do mundo real.

> Outro fator que deve ser levado em consideração é o da interferência no sistema. Quando um sistema é instrumentado para permitir a extração de medidas de desempenho, este estará operando em condições diferentes daquelas consideradas normais. A aplicação deste tipo de técnica deve ser acompanhado de um estudo que demonstre não ser significativa a interferência da instrumentação na operação do sistema (CECHIN, NETTO, 2001, p. 128).

As técnicas elementares são monitoração e *benchmarks*. Quando se pretende obter um índice de desempenho (um número que identifique o desempenho médio do sistema), pode ser utilizado a monitoração. Um exemplo seria fazer uma análise de um conjunto de computadores diferentes, fazendo um levantamento de quanto tempo levam para resolver um determinado algoritmo. Quanto aos *benchmarks*, consistem na aplicação de uma carga de trabalho (*workload*) sobre um sistema em si. Entretanto, esta carga de trabalho deve ser bem conhecida, devendo ter sido projeta com base nas características na qual deseja-se fazer a análise. Maiores detalhes desta técnica serão abordados na seqüência do trabalho.

Já as técnicas indiretas fazem parte de um grupo capaz de fazer avaliações de sistemas que ainda não existem. Essas técnicas são aplicadas no estágio do desenvolvimento, ou seja, são efetivadas na modelagem do sistema. Dispõem de ferramentas de previsão, possibilitando a modelagem de sistemas através de algum paradigma. De acordo com Cechim e Neto (2001, p129), "as técnicas indiretas usam ferramentas que podem ser analíticas, um conjunto de fórmulas e/ou diagramas que podem ser manipulados adequadamente, ou uma descrição que será "executada" em alguma ferramenta computacional".

6.2 Benchmarks

Fazer a análise de desempenho em máquinas paralelas ou *cluster* pode ser uma tarefa relativamente complexa, devido ao não determinismo das aplicações utilizadas, ou seja, não se tem uma garantia que uma determinada tarefa será executada de maneira igual em instantes consecutivos de tempo. Dentro deste contexto, definir métricas adequadas, bem como uma forma confiável para obtenção das mesmas é essencial para a avaliação do desempenho. Nesse âmbito, surgiram os *benchmarks* como uma poderosa ferramenta de avaliação.

Sill define um *benchmark* como "um teste que mede a performance de um sistema, ou subsistema, em uma tarefa, ou conjunto de subtarefas, bem definidas" (SILL, apud Cechin, Neto, 2001, p. 143).

> O desempenho em sistemas paralelos envolve um conjunto de medidas totalmente diferente em relação aos sistemas seqüenciais. Em sistemas paralelos as principais medidas são o ganho de velocidade (*sppedup*) e a capacidade de crescimento (escalabilidade). O desempenho em qualquer sistema computadorizado pode-se resumir em velocidade na execução das tarefas. Quanto mais rapidamente as aplicações forem executadas melhor o desempenho (PITANGA, 2004, p. 161).

Normalmente os *benchmarks* são utilizados para mensurar características gerais e performance de um sistema desconhecido em uma tarefa já conhecida ou bem definida. Dessa

forma, um *benchmark* pode refletir o comportamento de um recurso computacional, seja este recurso um programa, um subsistema ou uma arquitetura completa.

6.3 Ferramentas de benchmarks

A seguir, serão apresentadas algumas ferramentas de *benchmarks open-source* para medir o desempenho computacional em *clusters* computacionais. Esses *benchmarks* são propostos por membros do *Institute of Electrical and Electronic Engineers, Task Force on Cluster Computing* (IEEE-TFCC). Fundado em 1884, nos Estados Unidos, o IEEE é em uma sociedade técnico-profissional internacional dedicada ao avanço da teoria e prática da engenharia nos campos da eletricidade, eletrônica e computação, sendo composto por mais de 380.000 associados, como estudantes, engenheiros, cientistas, entre outros profissionais, de vários paises (IEEE, 2006; CLUSTER, 2006).

O grupo de desenvolvimento dos *benchmarks* é composto dentre outras pessoas, por Cosimo Anglano (coordenador), Mark Baker, Rajkumar Buyya, Niraj Srivastava, Rajat Todi, Greg Lindahl. De acordo com este grupo, podemos organizar os *benchmarks* em três categorias:

- *Benchmarks* de domínio de aplicação, na qual mede o desempenho das aplicações e o *kernel* do *cluster*;

- *Benchmarks* de comunicação específica, onde são analisados os meios de comunicação do *cluster*;

- *Benchmarks* para *hardwares* específicos, que mede o desempenho de vários sub-sistemas do *cluster*.

A seguir, dentro de cada categoria, são apresentadas algumas ferramentas de avaliação:

6.3.1 *Benchmarks* de domínio da aplicação

- *WebStone* (MINDCRAFT, 1998; CLUSTER, 2006): Este *benchmark* cria uma carga em um servidor *web* fazendo requisições *Hyper Text Transfer Protocol* (HTTP). Ele cria requisições HTTP com conteúdos arbitrários onde mede o *throughput* [28]e a latência de cada transferência do HTTP, apresentando inclusive, relatórios de falhas da transação.

- HTTP LOAD (CLUSTER, 2006): É uma ferramenta simples de *benchmark* para servidores *web* que mede o número de processos simultâneos por segundo, o *throughput*, a latência da conexão, e o tempo de resposta que o usuário pode entregar as requisições solicitadas.

6.3.2 *Benchmarks* de comunicação específica

- MPBENCH (MPIBENCH, 2006): Foi desenvolvido com o objetivo principal de verificar o desempenho de implementações MPI em arquiteturas paralelas, podendo ser utilizado também para avaliar o sistema de comunicação ponto-a-ponto e de comunicação coletiva.

---

[28] Em computadores, consiste na medida da quantidade de trabalho que pode ser processada em um período de tempo definido. Na rede, é uma medida da quantidade de dados que podem ser transferidos com êxito em um período de tempo definido.

- *Special Karlsruher* MPI (SKAMPI) *Benchmark* (SKAMPI, 2006): Mede o desempenho como latência e *throughput* em implementações MPI, tanto em comunicações ponto-a-ponto como em comunicações coletivas.

6.3.3 *Benchmarks* para hardware específicos

- LINPACK (PITANGA, 2004; CLUSTER 2006): *High Perfomace* LINPACK (HPL) *Benchmark* utilizado para avaliar a lista dos 500 maiores computadores do mundo, denominada TOP500.

- LMBENCH (CLUSTER, 2006; PITANGA, 2006): *Software* que mede a latência e a largura de banda da movimentação dos dados entre o processador e a memória, na rede, em sistema de arquivos e discos.

7 EXPERIMENTOS REALIZADOS

Este capítulo visa documentar os procedimentos adotados para a instalação de um *cluster Openmosix* em uma situação específica, na qual foi utilizado para experimentos no Centro Universitário Feevale, podendo ser utilizado (mediante adaptação) como referência para instalações em configurações diversas. Também, serão apresentados os diversos testes realizados sobre os equipamentos instalados, fazendo uma análise do comportamento deste tipo de tecnologia em processos que requerem grande capacidade de processamento (*CPU-Bound*).

A seguir, é relatada a metodologia utilizada nos experimentos, os procedimentos realizados para por o *cluster Openmosix* em funcionamento e os resultados obtidos nos experimentos executados.

7.1 Metodologia utilizada

Os experimentos realizados são um comparativo entre máquinas descontinuadas, montadas na estrutura de um *cluster* em relação a uma máquina de maior capacidade

computacional, fazendo um teste da linearidade do desempenho. Ou seja, o objetivo era analisar se ao utilizar uma estrutura de *cluster* em tarefas que exigem grande poder de processamento, se obtêm uma melhora de performance em relação a uma máquina mais rápida e moderna, mas independente de uma estrutura paralela.

Nestes testes, máquinas foram adicionadas e/ou retiradas da estrutura, a fim de testar o quanto melhora o desempenho computacional em relação ao número de máquinas que compõe o sistema, trazendo um levantamento, por exemplo, se um *cluster* de duas máquinas tem o dobro de performance de um *cluster* com quatro máquinas.

Para a montagem da estrutura na qual foram feitas às análises, foram solicitadas ao Centro Universitário Feevale equipamentos descontinuados, a fim de fazer os experimentos em um ambiente independente das demais máquinas da instituição, uma vez que, para a configuração do *cluster*, fez-se necessária a instalação do sistema operacional GNU/Linux e *softwares* para análises.

Tanto o sistema operacional, quanto o *kernel* do *cluster Openmosix* e outros *softwares* para monitoração e testes utilizados foram elaborados com tecnologias não proprietárias, ou seja, todo material utilizado pode ser obtido através da Internet, sem qualquer tipo de custo.

Como forma de obtermos uma computação intensiva da CPU, utilizamos um *software* decodificador de áudio, na qual é utilizado para decodificar CDs de áudio no formato "WAV" para o formato compactado "MP3". Dentre os inúmeros *softwares* existentes no mercado (proprietários e livres) para este tipo de tarefa, foi escolhido o decodificador Bladeenc [29]para a utilização nos experimentos. Primeiramente, por ser um *software* de código fonte aberto e de fácil aquisição via *web*. Segundo, pelo fato de se poder ripar uma coleção de faixas de áudio com um simples *script*, o qual será descrito mais adiante.

[29] http://bladeenc.mp3.no

Para os testes, utilizamos diferentes e pré-determinadas configurações para conversão, como: i) número fixo de faixas (músicas) iguais (mesmo arquivo de música); ii) número fixo de faixas diferentes (diferentes arquivos de música). A quantidade de músicas definidas aqui como medida de carga foi variada, afim de avaliar a melhoria da escalabilidade na paralelização dos processos. Além da carga gerada pelo *software* decodificador de áudio, foi realizado também um segundo teste para a verificação do ambiente montado, através de um *script*, desenvolvido para gerar alta taxa de processamento *CPU-Bound.* Todos os experimentos realizados foram executados no mínimo quatro vezes em cada situação, como forma de garantir que os resultados descritos na seqüência deste trabalho não estavam sofrendo algum tipo de interferência em instantes diferentes de tempo, na qual pudesse vir a alterar os resultados relatados.

7.1.1 Configurações de hardware

Para os experimentos, foram disponibilizados pelo centro universitário Feevale três computadores tipo PC e um equipamento para conexão entre as máquinas (Hub). Além dos equipamentos fornecidos pela instituição, foi utilizado também um *notebook,* totalizando um agrupamento com quatro nodos em alguns casos, detalhados a seguir no Quadro 7.1 e ilustrados através da Figura 7.1 abaixo:

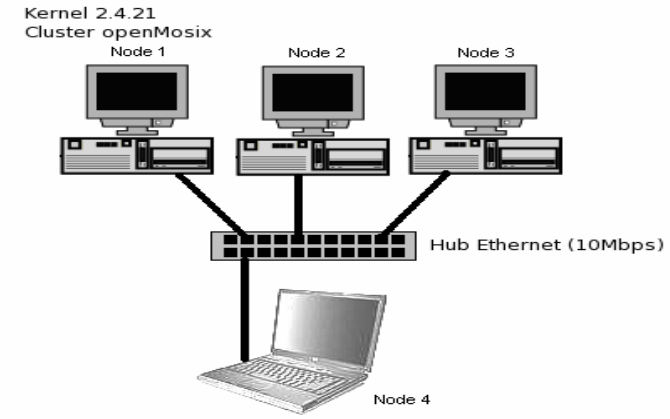

Figura 7.1: Estrutura do *cluster* montado no Centro Universitário Feevale

Quadro 7.1: Descrição dos equipamentos montados no laboratório da Feevale

EQUIPAMENTO 1	
Tipo de Máquina	*Personal Computer*
Modelo do Processador	AMD K6-2 500Mhz (5x100)
Modelo da Placa Mãe	SOYO 5EH
Freqüência Placa Mãe (Extern Clock)	100Mhz
Cachê L1	64Kb
Cachê L2	512Kb
Memória RAM	SDRAM 128Mb
HD	9,41Gb
Placa de Rede	SIS 900 *Fast Ethernet* 10/100 LAN *Adapter*
EQUIPAMENTO 2	
Tipo de Máquina	*Personal Computer*
Modelo do Processador	AMD DURON 800Mhz (4x200)
Modelo da Placa Mãe	SOYO K7VTA Pro
Freqüência Placa Mãe (Extern Clock)	100Mhz
Cachê L1	128Kb
Cachê L2	64Kb
Memória RAM	SDRAM 128Mb
HD	9,64Gb
Placa de Rede	Realtek RTL8139 10/100 *Fast Ethernet Adapter*
EQUIPAMENTO 3	
Tipo de Máquina	*Personal Computer*
Modelo do Processador	AMD DURON 800Mhz (4x200)
Modelo da Placa Mãe	SOYO K7VTA Pro
Freqüência Placa Mãe (Extern Clock)	100Mhz
Cachê L1	128Kb
Cachê L2	64Kb
Memória RAM	SDRAM 256Mb
HD	9,64Gb
Placa de Rede	Realtek RTL8139 10/100 *Fast Ethernet Adapter*
EQUIPAMENTO 4	
Tipo de Máquina	*NoteBook*
Modelo do Processador	Intel Celeron-SB Móbile 1333Mhz (10x133)
Modelo da Placa Mãe	Gateway Solo 1450
Freqüência Placa Mãe (Extern Clock)	132,89Mhz
Cachê L1	32Kb
Cachê L2	256KB
Memória RAM	SDRAM 256Mb
HD	20Gb
Placa de Rede	Intel Pro/100 VE *Network Connection*

Os equipamentos descritos acima, na seqüência do texto serão referenciados através de algumas denominações, como Máquina, Nó ou Node seguidos do número identificador descrito, ou seja, o equipamento 1 indicado no Quadro 7.1, pode ser citado como Máquina 1,

Nó 1 ou Node 1, correspondendo na realidade a mesma referência, seguindo a mesma regra para os demais equipamentos.

Para a conexão entre os nodos que compõem o agrupamento, utilizamos uma rede *ethernet* de 10Mbps, interligada através de um *hub*, modelo PSHUB 40 *Super Strack II*, fabricado pela 3Com. Vale salientar que a rede utilizada nestes testes tende a afetar o desempenho, visto que não é uma rede de grande capacidade de comunicação. Todavia, para fim destes testes é valido, uma vez que não estamos comparando *clusters* com diferentes tipos de redes, mas sim, tentando demonstrar como é possível obter processamento paralelo a um custo relativamente baixo, mesmo com equipamentos e dispositivos de redes já ultrapassados.

A máquina 4 (*Notebook*), por ser uma máquina de maior capacidade que as fornecidas pela instituição, na maioria dos testes neste agrupamento ficou desativada para o recebimento de processos remotos. Sua tarefa nestes casos, designou-se única e exclusivamente para capturar informações das outras máquinas do agrupamento, através de uma ferramenta gráfica de gerenciamento específica para este tipo de função, na qual será detalhada na seqüência do trabalho. Isso se deu, pelo fato de se querer avaliar em alguns casos, apenas as máquinas com menor poder de processamento.

Sendo assim, o *cluster* configurado é considerado de porte pequeno (localizado em apenas uma sala), com topologia do tipo NOW, composto por nós não-dedicados e heterogêneos. A estrutura real do *cluster* montado na instituição pode ser vista na Figura 7.2 abaixo:

Figura 7.2: Foto do *cluster* montado no laboratório da Feevale

Por outro lado, como maneira de testar o *cluster* em outras configurações e ter base para comparações, foi utilizado também um segundo agrupamento, com equipamentos de melhor capacidade, interligados de maneira ponto-a-ponto[30], duas máquinas distintas, através de uma rede *ethernet* de 100Mbs. Um *notebook* (mesma configuração do agrupamento anterior) e uma máquina *desktop*, descrita abaixo:

Quadro 7.2: Descrição do equipamento complementar utilizado nos experimentos

EQUIPAMENTO 5	
Tipo de Máquina	*Personal Computer*
Modelo do Processador	AMD Athlon XP 1600+ 1400Mhz (5,25 x 267)
Modelo da Placa Mãe	ECS M863
Freqüência Placa Mãe (Extern Clock)	133,34Mhz
Cachê L1	128Kb
Cachê L2	256Kb
Memória RAM	512Mb
HD	40Gb
Placa de Rede	*SIS 900 Fast Ethernet 10/100 Lan Adapter e ADMTEK AN983 Fast Ethernet*

A estrutura deste segundo agrupamento, montado na residência do autor deste trabalho, pode ser observado através da Figura 7.3.

[30] Máquinas conectadas diretamente, através de um cabo de rede, sem auxilio de dispositivos como *hub* ou *swith.*

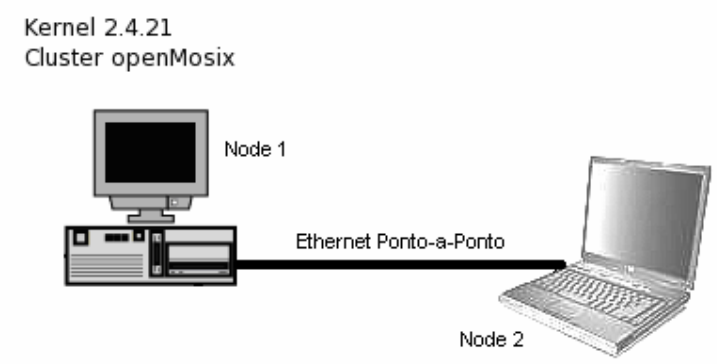

Figura 7.3: Estrutura de um *cluster* ponto-a-ponto montado para experimentos

7.1.2 Software Utilizados

Para estes experimentos, foram utilizados diversos tipos de *software*, como sistemas operacionais, bibliotecas para funcionamento do *cluster Openmosix*, ferramentas de monitoração, codificador de áudio, além de um *script* escrito para a geração de processos *CPU-Bound*.

7.1.2.1 Sistema operacional

O sistema operacional instalado em ambas as máquinas foi a distribuição GNU/Linux RedHat versão 9.0. Porém, pode ser utilizado outras distribuições Linux disponíveis, como Mandrake, Suse e Conectiva. Após a instalação, alguns serviços do SO que não seriam utilizados foram desativados a fim de operar com o mínimo de recursos necessários para seu funcionamento, sendo comentados maiores detalhes mais adiante. O sistema operacional pode ser obtido através do site: <http://www.linuxiso.org/>.

7.1.2.2 Kernel do Cluster

Para que as máquinas funcionassem em forma de *cluster*, foi utilizado um *kernel* pré-compilado do *Openmosix*, versão 2.4.21. Atualmente, já existem versões mais atualizadas deste *kernel*, que pode ser feito *donwload* pelo site: <http://sourceforge.net/projects/openmosix>.

7.1.2.3 Visualização e Monitoramento

Para a visualização e monitoramento do *cluster*, utilizamos uma ferramenta gráfica, conhecida por *Openmosixview*. O *Opemosixview* é um conjunto de ferramentas gráficas utilizadas para gerenciar e monitorar os principais parâmetros do *cluster*, como visualização da carga de todos os nós do agrupamento, visualização de processos remotos em outros nós, executar ou ainda transferir processos para outros computadores que compõem o agrupamento. São elas:

- ***Openmosixview***: é a aplicação principal de administração e monitoramento, na qual informa detalhes como quantas máquinas compõem o *cluster*, qual máquina está ativada e/ou desativada, percentuais de ocupação de cada processador, etc. A interface do *Openmosixview* pode ser vista na Figura 7.4.

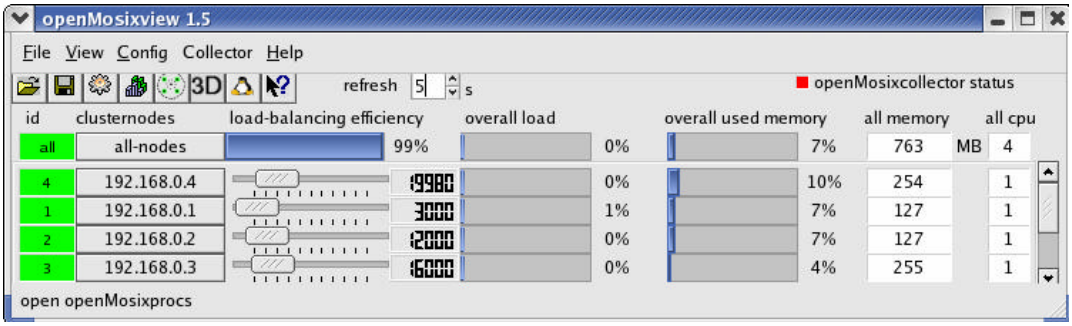

Figura 7.4: Interface do Openmosixview

- ***Openmosixprocs***: consiste numa interface para gerenciamento de processos em execução. Nesta janela, podemos saber maiores detalhes de cada processo, por exemplo, se o mesmo está sendo executado na máquina local ou ainda, se foi transferido para execução remota e para qual o mesmo foi repassado. A Figura 7.5 ilustra esta interface.

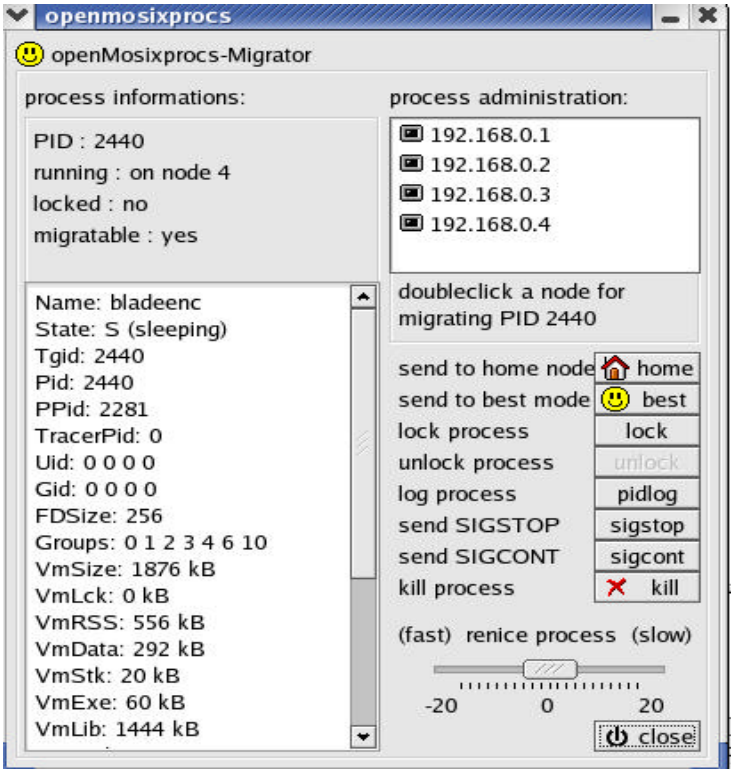

Figura 7.5: Interface do Openmosixprocs

- ***Openmosixcollector***: serve para coletar informações dos nós, através de um processo *daemon* para geração de *logs* do sistema, ou seja, ao ativar esta função, diversas informações passam a ser coletadas para posteriores análises na ferramenta Openmosixanalyzer.

- ***Openmosixanalyzer***: permite através de gráficos, analisar os dados coletados através da ferramenta *Openmosixcollector*. Abaixo, nas Figuras 7.6 e 7.7, podemos visualizar detalhes desta ferramenta.

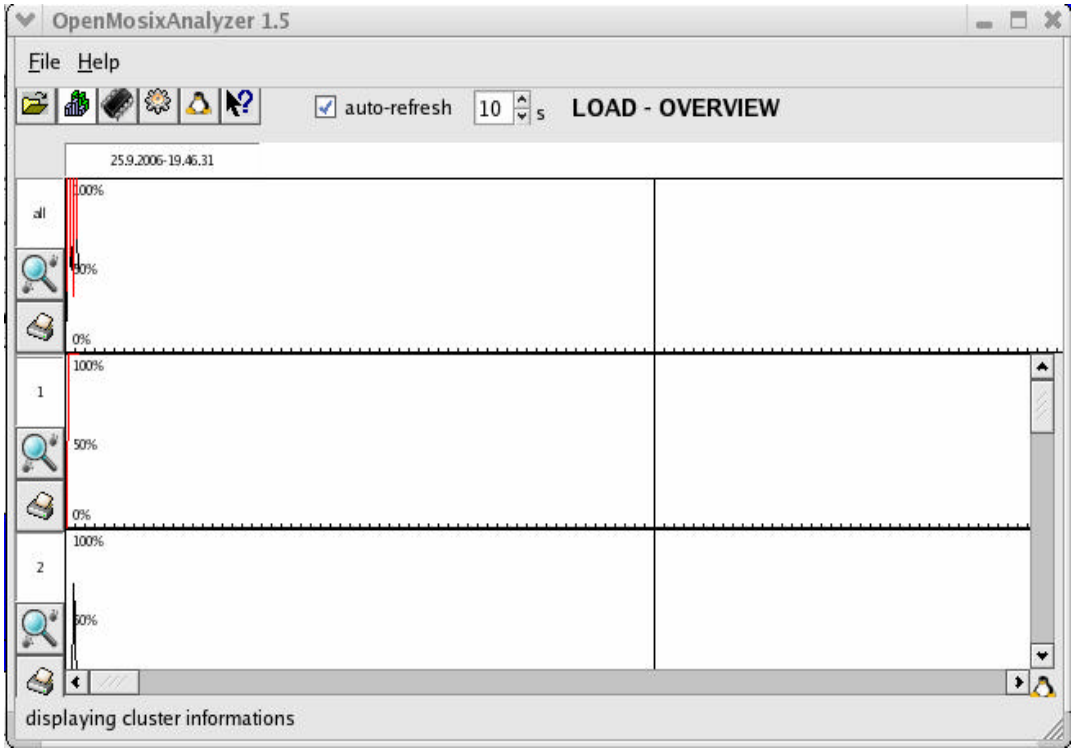

Figura 7.6 Interface gráfica do *Openmosixanalyzer*

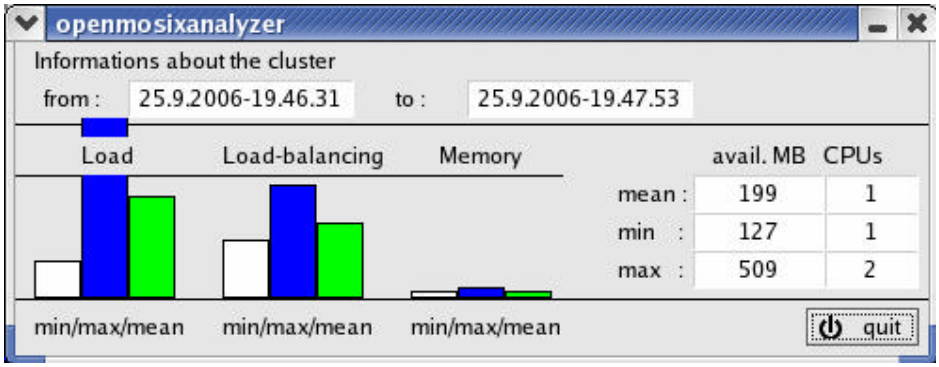

Figura 7.7 Detalhes da ferramenta *Openmosixanalyzer*

- ***Openmosixhistory***: apresenta histórico dos processos do agrupamento. Nesta interface, podemos observar todos os processos que estão em execução no momento. Os processos marcados com ícones em verde são processos que foram migrados para outros nós do *cluster*. Por outro lado, processos que não podem ser migrados são visualizados com um ícone em forma de um cadeado. Na imagem a seguir, Figura 7.8 observamos alguns destes detalhes.

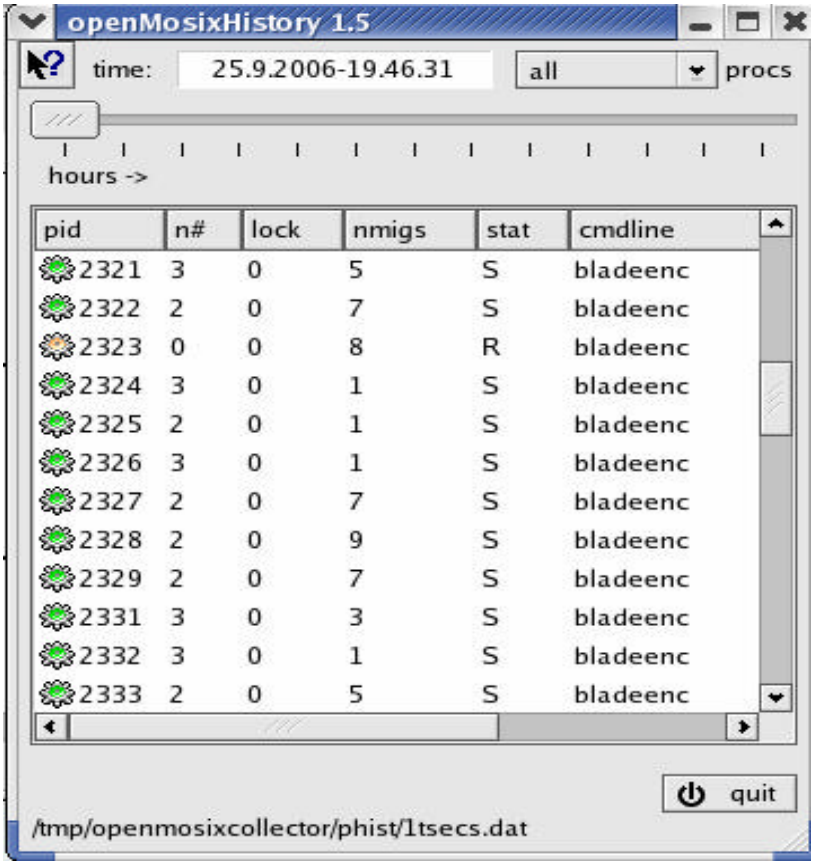

Figura 7.8: Interface do *Openmosixhistory*

- ***Openmosixmigmon**:* monitora os processos migrados pelo *cluster*. Com esta interface fica fácil visualizar como anda a migração dos processos entre todos os nodos do *cluster*. Por exemplo, na Figura 7.9 a seguir, podemos observar de cara uma grande quantidade de processos enviados para um único nó, enquanto os demais, praticamente não estão recebendo processos.

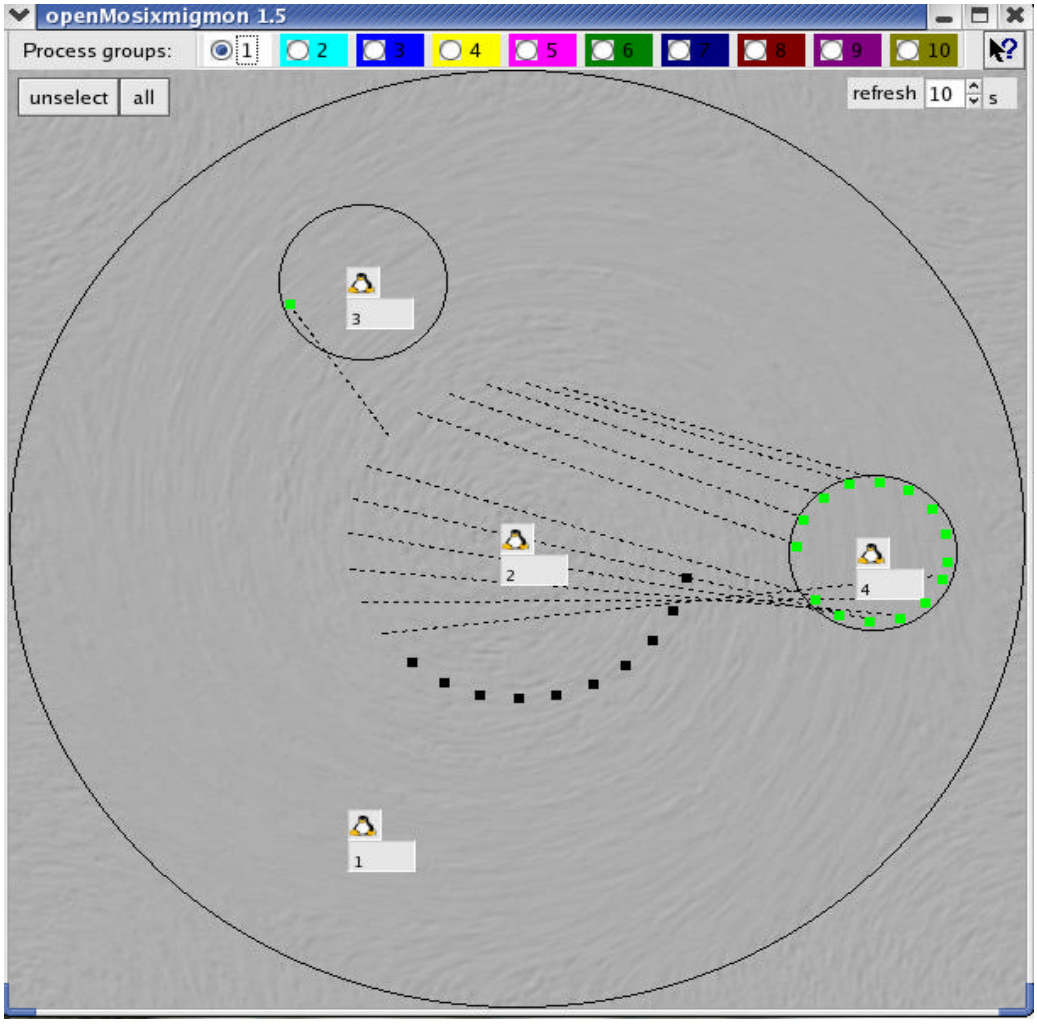

Figura 7.9: Interface do *Openmosixmigmon*

Todos os programas citados acima, são acessíveis através da janela principal do *Openmosixview*, onde os comandos do *cluster* podem ser executados pelo simples clicar do mouse. O conjunto de ferramentas do *Openmosixview* pode ser adquirido através do site: <www.openmosixview.com>.

7.1.2.4 Decodificador de áudio

Para a conversão de músicas, adotamos o Bladeenc, que nada mais é que um *software* que *ripa* uma coleção de arquivos de áudio para o formato MP3. O Bladeenc pode ser encontrado em: <http://bladeenc.mp3.no/>.

7.1.2.5 Script Gerado

Como forma de obtermos uma computação intensiva, e testarmos se o estava ocorrendo à migração automática dos processos, foi criado um simples algoritmo que gera diversos processos *CPU-Bound.* O *script* referido é composto por 6 instâncias do programa Awk, que pode ser visto a seguir:

```
  Awk 'BEGIN {for (x=0; x< 100000000; x++) for (y=0; y < 100000000; y++);}`
```

7.2 Procedimentos

A seguir, será relatado os principais procedimentos que utilizamos para a montagem de um *cluster Openmosix* do tipo *Single Pool*. Por serem processos rápidos, de fácil configuração e com poucas máquinas, todos os procedimentos adotados foram realizados individualmente em cada nodo do agrupamento. Entretanto, em casos que serão utilizados muitas máquinas, recomenda-se fazer a instalação em uma máquina e criar uma imagem do sistema, o que agilizará o processo de instalação e configuração nas demais estações.

O processo de instalação do sistema operacional adotado foi por meio de uma unidade de CD-ROM, através de um modo gráfico, para facilitar a interatividade entre o usuário e o sistema. Detalhes quanto à instalação do sistema operacional, neste caso o Red Hat 9.0, não serão relatados, por ser um processo distinto, que depende da distribuição Linux escolhida. Das instalações do sistema operacional feitas, o que difere no modo de instalar em cada terminal, é a configuração da rede, onde deve ser mudado o endereço da rede e o nome de cada estação. Nos estudos foi adotado o IP inicial 192.168.0.1 para a primeira máquina e as subseqüentes 192.168.0.2, .3, e .4 respectivamente, tendo a mesma seqüência lógica para os *hostnames*: node1.cluster.br para a Máquina 1 e node2.cluster.br, node3 e node4 sucessivamente para as demais. A máscara de rede ficou definida como: 255.255.255.0.

Após a instalação do sistema operacional em todas as máquinas e configuração da rede de comunicação, á próxima etapa consiste em recompilar o *kernel* do linux para utilização do *cluster Openmosix*. Existe duas maneiras de se instalar o *cluster*. Através de *patches* no *kernel* com compilação do mesmo, ou por instalação de um *kernel* pré-compilado. Por ser mais simplificado, utilizamos à versão pré-compilada para a instalação. O *download* e instalação do *kernel* procedeu da seguinte maneira:

Começamos baixando o *kernel* e um conjunto de ferramentas pré-compilados (extensão rmp) do *Openmosix*, que pode ser baixado através do site <www.openmosixview.com> . Para a instalação do mesmo, basta dar um duplo *click* sobre os arquivos baixados (openmosix-kernel-2.4.21-openmosix1.i386.rpm e openmosix-tools-0.3.6-2.i386.rpm) com o usuário *root*, que o sistema operacional (RedHat) se encarrega de fazer o restante da instalação. Após a conclusão das instalações, editamos e gravamos gestor de inicio Lilo, para habilitar o novo *kernel* na inicialização da máquina. Nada impede ou influência em utilizar outro gestor de inicio, por exemplo, o Grup. A escolha foi por preferência do autor deste trabalho.

Uma vez instalado o novo *kernel* e configurado o gestor de inicio, é necessário reiniciamos o computador, para que as novas instalações entrem em vigor. Depois de reiniciar o sistema, já com o novo *kernel*, restam poucos detalhes para que o *cluster Openmosix* entre em funcionamento. È necessário editarmos um arquivo de configuração chamado

openmosix.config, localizado em /etc/openmosix, e alterar alguns parâmetros. Para a edição deste arquivo, pode ser utilizado qualquer editor de texto disponível no linux.

Neste arquivo, podemos indicar qual placa de rede será utilizada para comunicação entre os processos, à velocidade (*Speed*) que será utilizada como parâmetro pelo *cluster* para a migração dos processos, se um nó pode ou não ter processos migrados automaticamente, se os nós serão auto-detectados, entre outros parâmetros. Este arquivo, por padrão vem com todas as linhas de configuração comentadas (inicio da linha com o caracter #), deixando o *cluster* inicialmente desativado. Para que as configurações entrem em vigor, deve-se retirar o caracter "#" do inicio da linha dos parâmetros na qual deseja-se configurar. Pelo menos, a linha referente à placa de rede deve ser desmarcada, para que o *cluster* reconheça por onde será feita a comunicação entre os processos.

Neste caso, desmarcamos a linha correspondente à placa de rede e definimos qual placa de rede seria utilizada para a comunicação. Nas máquinas utilizadas para o estudo, apenas em uma tivemos que alterar a placa de rede, pois á maquina em questão possuía mais de uma placa de rede. Nas demais, a única tarefa executada foi desmarcar a linha referente ao dispositivo de rede, pois por padrão, este arquivo vem definido que a placa de comunicação é a eth0. Abaixo, é demonstrado o parâmetro inicial do arquivo de configuração referente à placa de rede e logo após, o parâmetro editado com a placa de rede escolhida para comunicação. Neste caso, a ilustração refere-se à máquina que possuía mais de uma placa de rede, onde definimos que a placa a ser utilizada pelo *cluster* seria a eth1.

```
    # Specify which network interface the autodiscovery-daemon
should listen to
    # AUTODISCIF=eth0

    # Specify which network interface the autodiscovery-daemon
should listen to
     AUTODISCIF=eth1
```

Definimos também, que os processos executados nas máquinas poderiam migrar para outros nós. Isso é definido através da linha de configuração abaixo:

```
    # Processes are allowed to migrate to other nodes.
     MIGRATE=yes
```

Outro parâmetro importante, e o que corresponde à velocidade do nó. Aqui, deve ser tomado muito cuidado em relação aos valores informados, uma vez que os algoritmos de balanceamento de carga do *Openmosix* baseiam-se nesta configuração para saber a capacidade de processamento da máquina. Valores iguais, por exemplo, para uma máquina de maior capacidade com uma de menor capacidade, pode fazer com que o *cluster* divida processos entre elas igualmente, sendo que uma não terá o mesmo desempenho da outra, afetando o desempenho global. Mas isso será observado mais adiante em experimentos executados. Caso não for definida a velocidade da estação, o *cluster Openmosix* define um valor automaticamente. O critério utilizado para este valor inicial informado pelo *Openmosix*, não foi descoberto, mas muito provavelmente, deve ser com base em levantamentos dos recursos de *hardware* disponíveis. Inicialmente, não alteramos esta configuração, deixando a cargo do *Openmosix* fazer esta definição. Por padrão, o arquivo de configuração vem desabilitado, conforme a ilustração abaixo.

```
    # Pass the following value to mosctl setspeed when starting
the node
    # man mosctl for details
    #NODESPEED=10000
```

Devemos também, configurar um sistema de arquivos do *Openmosix*, denominado oMFS, para que todos os nós possam ler e escrever num diretório padrão. O oMFS, nada mais é que um sistema de arquivos que provê consistência de cachê para manter uma única cachê entre os nodos.

Para a configuração do oMFS, basta criar em cada estação um diretório com o comando mkdir /mfs; e adicionar a linha de comando abaixo no arquivo *fstab*, localizado em */etc*.

```
mfs_mnt      /mfs  mfs   dfsa=1 0 0
```

Pronto, basta reiniciar o sistema com o comando *reboot* que o *Openmosix* já deverá estar em funcionamento. Para verificar detalhes do *cluster,* como se o serviço *Openmosix* está ativado e quantos nós fazem parte do agrupamento, execute o comando a seguir:

```
/etc/init.d/openmosix status
```

Para ativar ou desativar um nó do agrupamento, basta executar a linha de comando abaixo, respectivamente:

```
/etc/init.d/openmosix start
/etc/init.d/openmosix stop
```

Por fim, após instalação e configuração de todas as estações do *cluster*, a próxima etapa foi instalar o decodificador de áudio Bladeenc, para que os experimentos propostos pudessem ser efetivados, assim como a ferramenta *Openmosixview*, para que gerenciamento e monitoramento do *cluster* pudessem ser visualizados.

Tanto o *Openmosixview*, quanto o Bladeenc, possuem bibliotecas pré-compiladas, o que facilita muito o processo de instalação, sendo que para os dois casos, foi utilizado este tipo arquivo. Para a instalação destes *softwares*, após o *donwload* dos arquivos openmosixview-1.5-redhat90.i386.rmp e benc-0942-linuxrpm.i386.rpm, foi executado o arquivo de instalação em cada estação. Este processo é muito rápido, não levando toda a operação mais do que três minutos por máquina.

Para que se possamos visualizar os processos em andamento em máquinas distintas, é preciso ter o *Openmosixview* em todas as estações. Quanto ao decodificador de áudio, não é necessário ter instalado em todas as máquinas do *cluster*, apenas na máquina que será iniciado o processo de decodificação. Entretanto, como foram feitos experimentos com diferentes configurações de *hardware*, onde máquinas eram adicionadas e/ou retiradas do grupo, foi instalado o *software* Bladeenc em todos os nodos que compõem o agrupamento.

Outra tarefa executada na instalação do *cluster*, não que seja necessário, mas que pode influenciar do desempenho do *cluster*, é quanto à relação dos serviços do sistema operacional que estão sendo executados em cada terminal. O ideal, para tirarmos o máximo de proveito de cada estação, é deixar às máquinas com o mínimo de serviços necessários para o seu funcionamento, devendo é claro, ser levado em consideração, cada situação em especial. Neste caso, foram desativados vários serviços do sistema operacional que não seriam utilizados, inclusive a interface gráfica do linux, onde todos os testes foram executados com terminais rodando em modo texto. Apenas a máquina 4 (*Notebook*), não foi desativada a interface gráfica, em virtude deste nó, em muitos casos, utilizar ferramentas de

monitoramento que necessitavam destes recursos habilitados, para visualização dos processos em execução no agrupamento.

7.3 Experímentos executados

A seguir, serão relatados os experimentos efetivados com o *cluster Openmosix* instalado no Centro Universitário Feevale e em um instalado na residência do autor deste trabalho.

7.3.1 *Script* Gerado

Com o *cluster* configurado, a próxima etapa foi verificar se os processos estavam migrando de maneira automática entre os nodos do agrupamento. Para isso, foram executados 6 instâncias simultâneas de um script gerado (descrito na seção anterior), gerando uma grande carga de processos *CPU-Bound*. Os testes ocorreram da seguinte maneira:

Primeiramente, os *scripts* foram executados apenas no nodo 3, a fim de observarmos quanto tempo levaria sua execução em apenas uma máquina. Posteriormente, foi executado o mesmo *script*, sendo adicionados os outros nodos, intercalando as configurações da seguinte maneira: Nodo3 + Nodo2, Nodo3 + Nodo1 e por fim, interligadas todas as máquinas do agrupamento: Nodo3 + Nodo2 + Nodo1. O tempo de execução em relação aos equipamentos utilizados pode ser observado na Figura 7.10 abaixo:

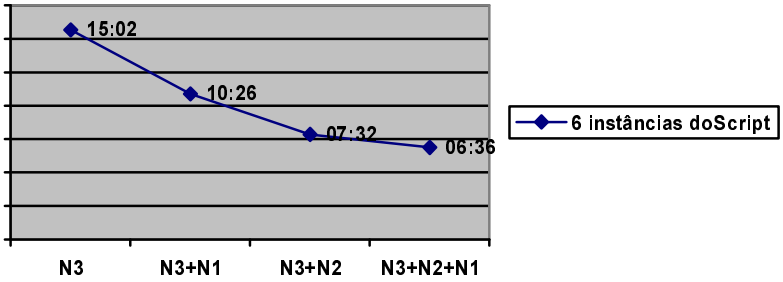

Figura 7.10: Relação de tempo de execução (6 scripts) em relação aos equipamentos

Sendo assim, neste teste observamos que o *Openmosix* reconhece automaticamente a presença de um novo nó na rede e que, tão logo um nó seja adicionado ao *cluster*, o mesmo está apto a receber processos remotos, caso seu processador não esteja ocupado com outra tarefa naquele momento. Assim, nesta situação especifica, o aumento do número de nós, nos dá um considerável ganho de performance, 51,26% (redução do tempo), quando executado o Nodo 3 em conjunto com o Nodo 2, cuja configuração de hardware são similares, e 31,69% quando interligados, os nodos 3 e 1, tendo um ganho menor, mas considerável. Por outro lado, a diferença em relação ao tempo de execução, quando executado com duas máquinas (Nodo 3 + Nodo 2) para a execução com três máquinas não é tão grande, aumentando neste caso, em 13,11% a performance, totalizando 57,66% de ganho de velocidade em relação a primeira máquina testada. Voltamos a lembrar que o *cluster* é heterogêneo, e o Nodo 1, por ser de capacidade menor, não tem o mesmo desempenho que as outras duas máquinas.

Um fato interessante, que deve ser observado, é a maneira na qual o *Openmosix* tira proveito das aplicações. O *Openmosix* é um tipo de *cluster* que não requer que suas aplicações tenham que ter sido previamente desenvolvida para um ambiente paralelo. Por exemplo, uma simples aplicação seqüencial (que não pode dividir seu processamento) não se beneficiaria com a utilização de um *cluster* computacional. O que torna esta tecnologia diferente das demais, é pelo fato que processos inteiros são transferidos entre os nodos do agrupamento, conforme a necessidade, para processamento remoto, trazendo o resultado ao término do mesmo, para a máquina na qual foi solicitada a operação. Isso pode ser observado no *script* anterior, quer requer grande capacidade de processamento, mas não pode ter dividida suas tarefas em particular, ou seja, se somente for executado uma única instância do *script*, o processo em questão só seria beneficiado neste ambiente, se houvesse algum nó na rede com melhor capacidade de processamento, que justificasse esta transferência para outro terminal.

Todavia, ao executar vários scripts simultâneos em uma única máquina (nos experimentos foram executados 6 instâncias), a probabilidade de alguns processos migrarem para outros nós do agrupamento é grande, evitando assim, a sobrecarga de execução em apenas um nó.

Outro fato importante observado é quanto à falha de algum dos nodos, ou seja, se uma tarefa for migrada para algum nó, e o mesmo venha a falhar, todas às informações serão perdidas, o que normalmente ocorre em qualquer máquina, uma vez que o *Openmosix* não possui mecanismos para tolerância de falhas. Por outro lado, se um nó for paralisado através de comandos do linux, como *halt* e *shutdown,* os processos remotos dá máquina em questão serão migrados novamente ao seu nodo de origem ou ainda, para outro nodo que porventura esteja disponível na rede.

7.3.2 Conversão faixas de áudio fora do ambiente de *cluster*

A forma em que um aplicativo é executado pode ser determinante para que o resultado final seja alcançado em um tempo menor de execução. Por exemplo, executando o *software* Bladeenc que será utilizado nos experimentos a seguir, a maneira de chamar sua execução, mesmo em uma única máquina, fora de um ambiente de *cluster*, tem uma notável diferença. Para convertemos os arquivos de áudio para mp3, podemos chamar o *software* de duas maneiras distintas.

Podemos executar o *software* Bladeenc através do script "**`bladeenc –quit *.wav -256 –copy -crc`**". Nesta situação, estamos dizendo que a aplicação devera converter todos os arquivos (wav) do diretório selecionado em seqüência. Logo, não teremos grandes benefícios, pois o cluster vê apenas um grande processo em execução, onde no máximo, todo o processo em questão poderá ser transferido para um nó de melhor capacidade, mas igualmente continuará sendo executado em apenas uma máquina.

Por outro lado, podemos criar um *loop,* através da linha de comando "**`for n in 'ls *.wav`;do bladeenc –quit - quiet $n -256 –copy –crc &`**

done;", que irá chamar para cada música que se deseja converter, uma instância distinta do programa, ou seja, se temos dez músicas para conversão, serão abertos dez programas Bladeenc simultâneos, cada um responsável pela conversão de uma única música, enquanto no primeiro caso, temos um único programa responsável pela conversão de todas as faixas de música. A divisão, neste caso, em vários processos menores, simultaneamente, facilita a divisão das tarefas pelos nodos que compõem o agrupamento, não sobrecarregando apenas um nó.

Para exemplificar como isto pode fazer diferença, mesmo em uma única maquina, utilizamos o decodificador de áudio Bladeenc para converter 16 faixas de músicas iguais. Em um primeiro momento, convertemos as 16 faixas utilizando uma única instância do programa na máquina 3, levando para a conclusão deste processo 18:31min. Já o mesmo número de faixas, sendo convertido por 16 programas simultâneos na mesma máquina, cada qual, encarregado por um único arquivo, obtivemos o resultado final em 10:24min, dando um ganho de performance de 44,07%.

Conforme descrito acima, apenas pelo fato de dividirmos o processo de conversão de áudio entre vários programas simultâneos, mesmo em uma única máquina, obtivemos uma boa diferença em relação ao tempo de execução. Seguindo esta linha, mas agora interligando outros nós a rede, foram feitos diversos experimentos a fim de analisar o desempenho no ambiente de *cluster* computacional.

Iniciamos convertendo 16 faixas de músicas distintas (correspondente a todas as faixas de áudio de um CD, com média de 38,01Mb cada faixa, totalizando 608,2Mb para conversão), individualmente em cada máquina, ainda sem a interligação em rede, como forma de avaliarmos inicialmente, a diferença de tempo que cada nodo levaria para o término do processo. Executamos também, o mesmo procedimento para converter outro grupo de músicas, porém, neste caso, 16 faixas de áudio iguais (35,5Mb cada faixa, totalizando 568Mb). Para estas tarefas, conforme podemos observar no Figura 7.11, tivemos uma insignificante diferença entre o tempo de execução da Máquina 2 para a Máquina 3, apesar desta última possuir maior capacidade de memória RAM que a anterior. Por outro lado, ficou claro que a Máquina 1 tem um desempenho muito inferior em relação as outras máquinas do agrupamento, tendo levado em média 20 minutos a mais para a conclusão das conversões para

o grupo de músicas distintas e 18:50 minutos para o grupo de faixas iguais em relação as máquinas 2 e 3. Também fizermos um teste individual na máquina 4 (*Notebook*), onde, como era de se esperar, teve um melhor desempenho, se comparado com as demais estações, em virtude de ser um equipamento de maior capacidade computacional que as demais máquinas.

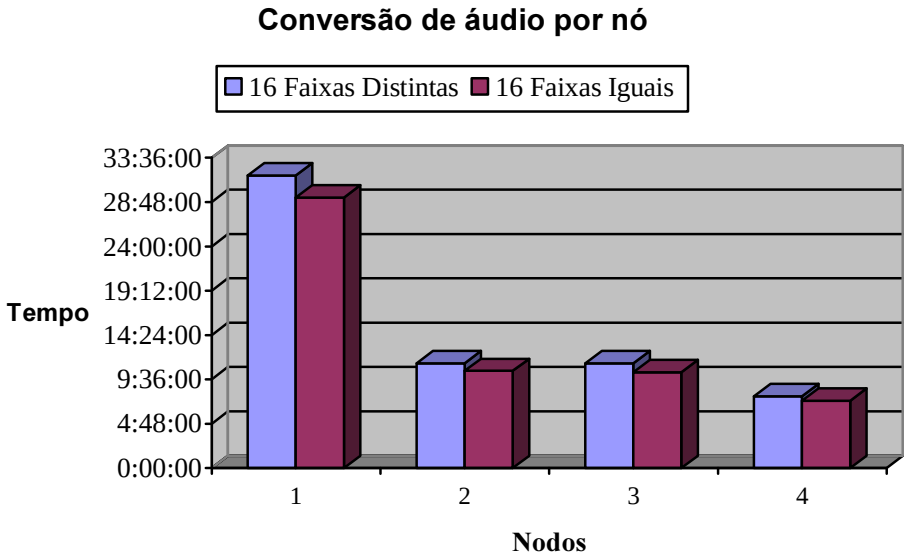

Figura 7.11 Conversão de áudio por nodos

Agora, entrando no ambiente de *cluster* computacional, foram feitos diversos experimentos, intercalando o número de máquinas, a fim de verificar o que isto influência no desempenho das aplicações.

7.3.3 Experimentos executados com número intercalado de máquinas

Iniciamos interligando os nodos 1 e 2, executando para a conversão de áudio, em ambos os testes, o mesmo número de faixas de músicas que a situação anterior, tanto para as faixas iguais, como para as distintas, onde podemos observar algumas diferenças no consumo

de tempo para o término das tarefas solicitadas, principalmente quando relacionando à maquina 1 com as demais.

Enquanto rodando a conversão do mesmo grupo de faixas iguais, apenas no Nodo 1, tivemos os trabalhos concluídos em 29:16min. Já, formando um *cluster Openmosix*, com duas máquinas, Nodo 1 + Nodo 2, ganhamos 63,75%, ou seja, os trabalhos nesta situação foram concluídos em 10:57min. Todavia, ao ser acrescentado mais um nodo, (Máquina 3), ao contrário do que esperávamos, tivemos uma pequena perda de performance, onde os trabalhos acabaram sendo concluídos em 11:10min, perdendo mais ou menos 5% de desempenho em relação as duas máquinas anteriores (Nodo 1 + Nodo 2). Podemos observar isto pela Figura 7.12

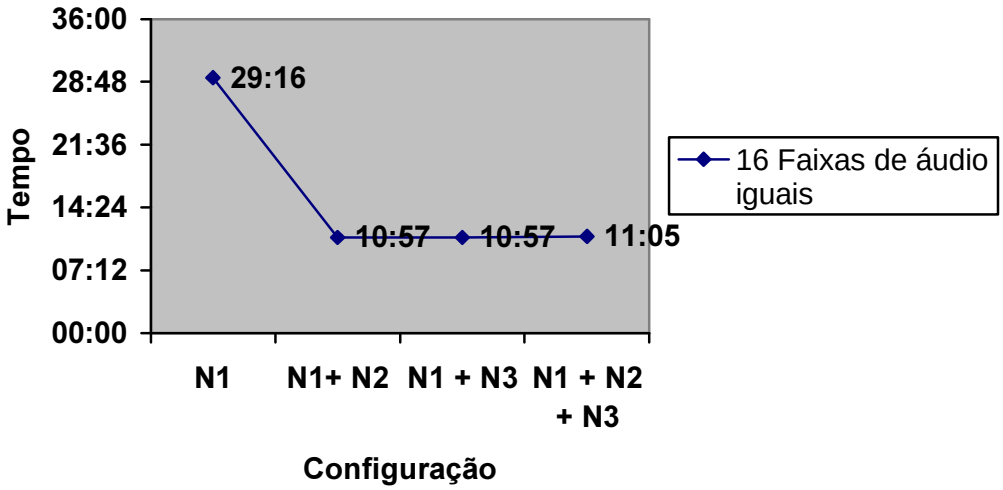

Figura 7.12: Conversão de 16 faixas de áudio iguais a partir do Nodo 1

Justifica-se esta perda de performance aqui ao fato da Máquina 1 ser inferior as outras máquinas e os processos terem sidos migrados na grande maioria para os outros nodos da rede, tendo praticamente todos os processos sendo executados remotamente.

Enquanto foram transferidos em média 10 processos para execução remota, quando interligado duas máquinas, tivemos 15 processos sendo executados remotamente com as três máquinas no *cluster*, intercalados entre os nodos 2 e 3. A migração destes processos pode ser

observada através da ferramenta *Openmosixmigmon*, visualizada através da Figura 7.13 abaixo:

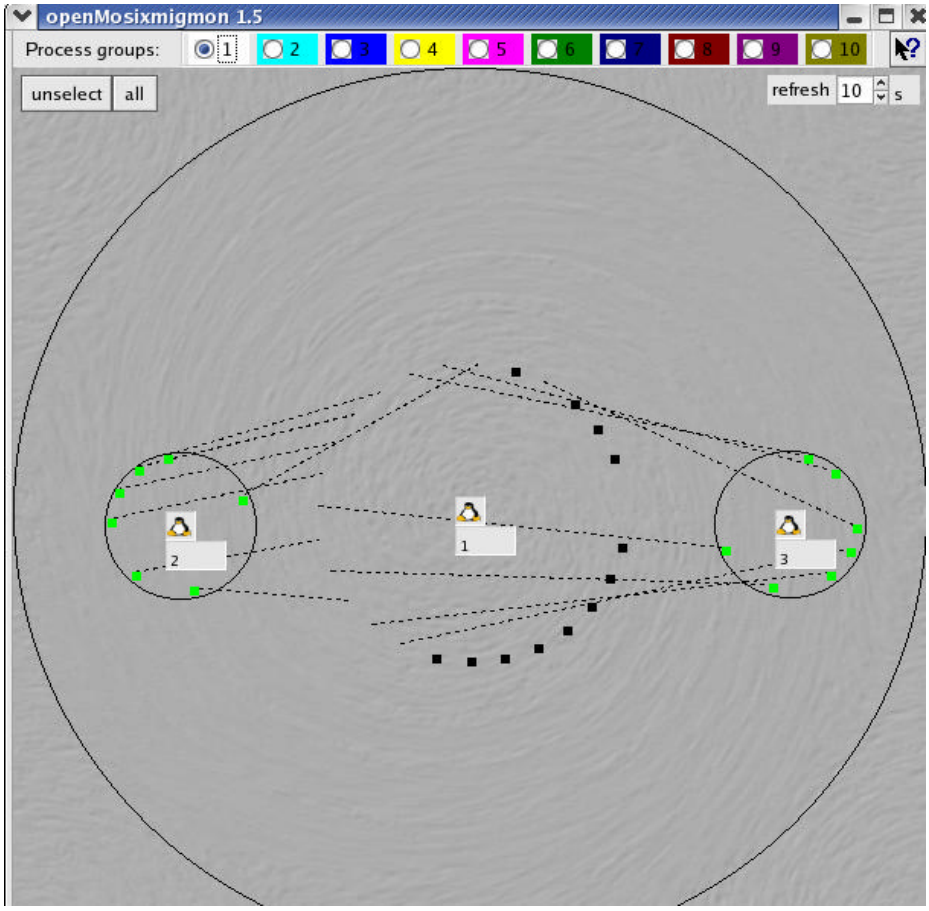

Figura 7.13: Migração intercalada de processos com três máquinas em conjunto

O mesmo processo, utilizando ás três máquinas só que iniciando o pedido de conversão pelo Nodo 2, ganhamos um pequeno desempenho com a utilização de três máquinas. Aqui, os trabalhos foram concluídos em 8:01min, cerca de três minutos a menos do que quando iniciado pelo nodo 1, trazendo a tona que em *cluster* heterogêneos, um mesmo processo, iniciado em nodos diferentes, pode dar uma diferença no tempo de processamento.

Um fato que pode justificar esta diferença de tempos, é que um processo iniciado em um nó de menor capacidade, acaba sendo transferido para um nó com maior capacidade de processamento disponível naquele momento, esbarrando em atrasos devido ao desempenho da rede de comunicação. Atrasos estes que podem ser compensados em situações que utilizam

longas tarefas de *CPU-Bound*, ao contrário dos testes aqui executados, que tiveram pequenas tarefas deste tipo.

Na Figura 7.14, através da ferramenta *Openmosixmigmon*, podemos observar uma situação com duas máquinas (Nodo 1 + Nodo 3), onde um processo sendo executado a partir nó de capacidade inferior, teve praticamente todos os seus trabalhos transferidos para execução remota, sobrecarregando assim, apenas um nó. Apesar desta transferência, mesmo que, quase na sua totalidade ainda ser compensada, pois se fosse executado todos os processos, apenas no nodo inferior (Nó 1), teríamos um tempo maior para a conclusão dos trabalhos. O ideal seria em casos como este, ter uma melhor divisão das tarefas, ou seja, se tivéssemos por exemplo, do grupo de 16 faixas de áudio para conversão, uns 5 processos sendo executados na máquina de origem (inferior) e os demais remotamente, os invés de termos 14 tarefas executadas remotamente e somente 2 no nó local, em tese poderíamos ganhar um pouco mais de desempenho.

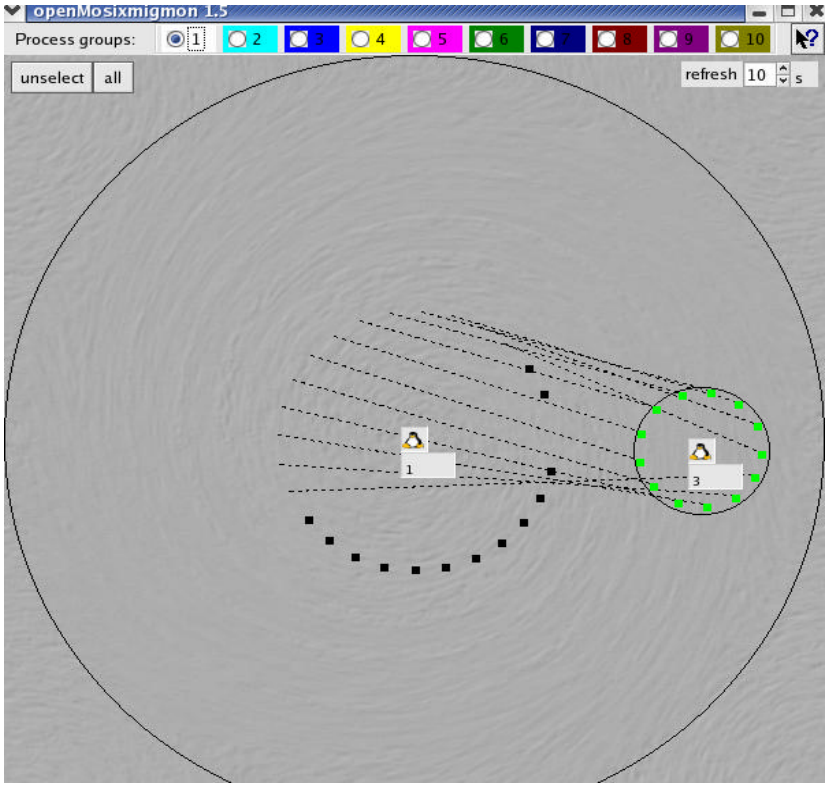

Figura 7.14: Migração de processos a partir de um nó inferior

Observamos também, que tanto para processos iniciados a partir do Nodo 2 como no Nodo 3, quando interligados apenas com o Nodo 1, ou que o mesmo fizesse parte do agrupamento, tivemos uma queda de desempenho na maioria dos casos. O Figura 7.15 abaixo deixa claro isso, onde os processos foram convertidos apenas na máquina 2 em 10:31min, levando 1,26min a mais na combinação das máquinas 1 e 2 , correspondendo numa queda de desempenho de 12,22%, e aumentando este percentual para 23,10% na relação dos nodos 2 e 3 ao serem adicionados com o Nodo 1

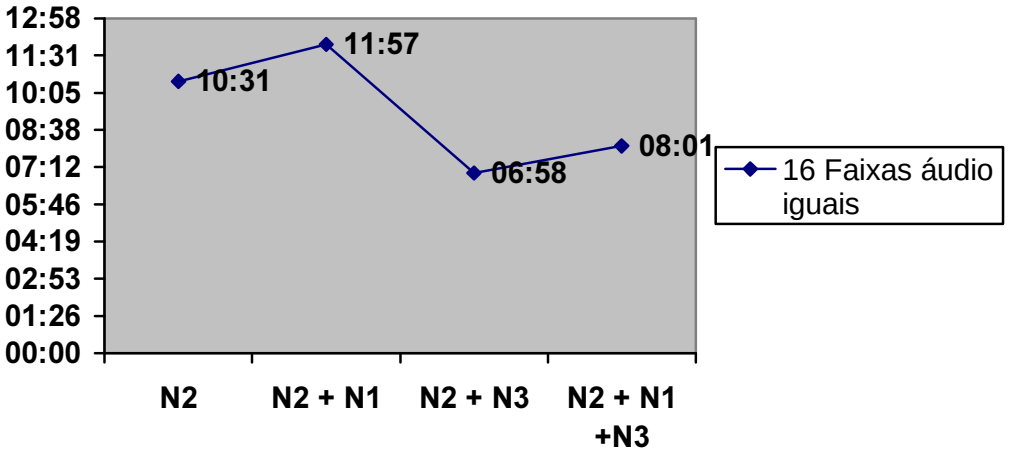

Figura 7.15: Conversão de 16 faixas de áudio iguais a partir do Nodo 2

Os mesmos testes feitos anteriormente, só que agora com um grupo de faixas de áudio distintas (16 arquivos de música diferentes), também foram executados. Nestes testes, como nos testes com faixas de áudio iguais, tivemos praticamente o mesmo tipo de resultados, variando aqui o tempo para a conclusão da conversão em virtude dos arquivos a serem convertidos serem diferentes. Por exemplo, quando convertemos ás músicas apenas no Nodo 3, tivemos os trabalhos concluídos em 11:21min, 7,27% mais rápido do que quando utilizado às maquinas 1 e 3, que em conjunto levaram 12:09min. Já, com o acréscimo de mais uma máquina, neste caso, a máquina 2, tivemos uma melhora de 23,9% em relação a máquina 3 apenas. Todavia, o tempo gasto para a conclusão dos trabalhos com os três nodos interligados (Máquina 1 + Máquina 2 + Máquina 3) é maior que quando utilizado apenas os nodos 2 e 3. Detalhes dos experimentos executados com músicas distintas a partir do nodo 3, pode ser visualizado na Figura 7.16 a seguir.

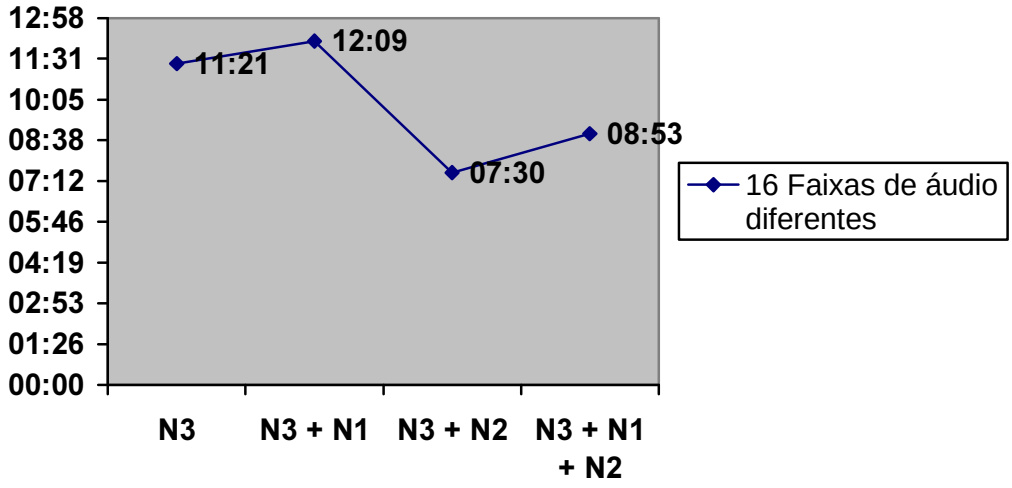

Figura 7.16: Conversão de 16 faixas de áudio distintas a partir do Nodo 3

Outra diferença levantada, que era de se esperar, é em relação à conversão dos arquivos distintos, onde um mesmo processo, ao ser executado em instantes diferentes, pode dar uma diferença no tempo de conclusão dos trabalhos. Ao contrário de quando convertemos faixas iguais, que o tempo de resposta é praticamente o mesmo em ambos os casos. Isso, se deve ao fato, de como temos arquivos distintos entre si, ao ser executado um processo duas vezes em instantes diferentes, nada garante que o mesmo processo será sempre executado na mesma máquina, o que modifica o tempo para o resultado final.

7.3.4 Experimentos executados intercalado à carga de trabalho de cada nó

Como forma de testarmos se a carga de trabalho (*speed*) configurada individualmente nos parâmetros do *cluster*, influência no desempenho dos agrupamentos, foram feitos inúmeros experimentos, também convertendo faixas de áudio, inclusive com os mesmos arquivos, simulando diversas situações distintas de configuração da carga de trabalho. Estes testes foram executados, utilizando os três nodos, alterando os arquivos de configuração do *Openmosix* (openmosix.config), para avaliar se o desempenho do agrupamento sofre

alterações conforme os valores de velocidade (*speed*) configurados manualmente nos parâmetros do *cluster*. Na Figura 7.17 abaixo, podemos visualizar a velocidade determinada pelo *cluster Openmosix* através da ferramenta *Openmosixview*, na qual está indicando (circulado em vermelho) a configuração de velocidade inicial determinada automaticamente pelo *Openmosix*. Conforme comentamos no inicio deste capítulo, as velocidades pré-determinadas bateram de acordo com os equipamentos dispostos, ou seja, tivemos a menor velocidade determinada para o Nodo 1, iguais para os nodos 2 e 3, que tem o mesmo processador e o valor mais elevado para o Nodo 4, cuja configuração é a melhor de todas entre os equipamentos dispostos..

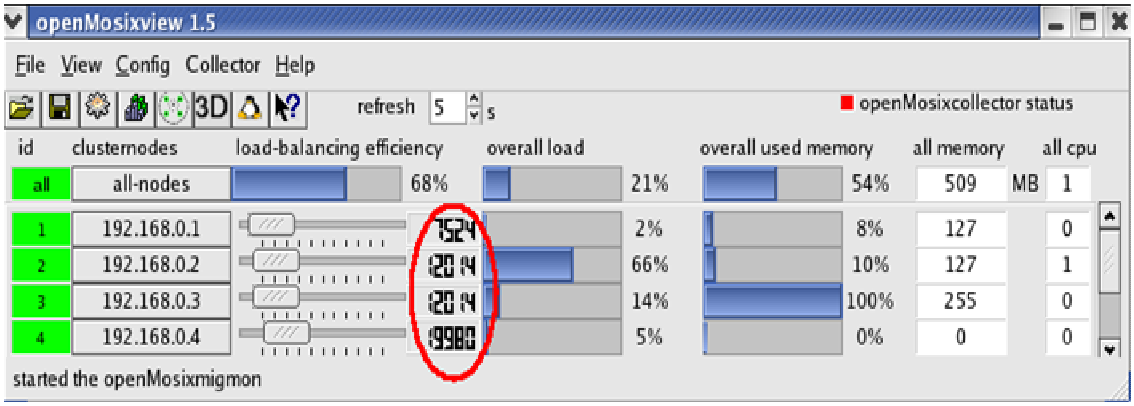

Figura 7.17: Interface da ferramenta *Openmosixview* indicando a carga de trabalho pré-determinada pelo *Openmosix*

Tomando como base os experimentos efetivados com músicas iguais, partindo os testes do Nodo 2, na qual foram efetivados anteriormente com a carga de trabalho definida automaticamente pelo *Openmosix*, podemos levantar algumas informações com relação aos testes executados em que alteramos a carga de trabalho manualmente.

Relembrando, quando executamos os testes nos três nodos (1,2 e 3) para converter as faixas de áudio com a carga de trabalho definida automaticamente pelo *Openmosix*, tivemos as tarefas concluídas em 8,01min. Alterando esta carga de trabalho, deixando todos os nodos do *cluster* definidos com a mesma carga de trabalho, neste caso, as três máquinas definidas com carga igual a 10000, tivemos os trabalhos concluídos em 8:10min. Quando aumentamos a carga de trabalho da máquina 1 para 15000, que em tese, é a mais inferior de todas, deixando os nodos 2 e 3 configurados com carga igual a 10000, tivemos um queda de 5% de

desempenho, ou seja, levou-se 8,41min para o término dos trabalhos, 40 segundos a mais do que quando executado com os parâmetros iniciais. Já fazendo o processo inverso, diminuindo o valor da carga de trabalho da máquina de menor velocidade (Máquina 1) para 5000, deixando ainda os nodos 2 e 3 com a mesma carga do teste anterior, em 10000, ganhamos 5,49% no tempo de execução.

Agora, aumentando a carga de trabalho conforme a configuração de *hardware*, deixando uma carga maior para a máquina de maior capacidade de processamento e uma carga menor para o nodo de *hardware* inferior, levando em consideração processador e memória disponível, configuramos os parâmetros da seguinte maneira: o Nodo 1 ficou definido com uma carga de 5000, enquanto os nodos 2 e 3 foram configurados com 10000 e 15000 respectivamente. Com esta configuração, ganhamos 7,74% de desempenho, ou seja, levou-se 7,39min para a conversão dos arquivos Ao ver que modificar a carga de trabalho, temos uma influência no desempenho das tarefas, diminuímos ainda mais a carga do Nodo 1, evitando que o mesmo venha a receber muitos trabalhos para processamento, uma vez que, este nó tem afetado o desempenho quando configurado com uma carga maior. Assim, re-configuramos novamente as cargas de trabalho, deixando o Nodo 1 com menos carga que as configurações anteriores, neste caso em 3000 e aumentamos um pouco mais as cargas do Nodo 2 e 3, para 12000 e 16000 respectivamente. Isso fez com que a performance aumentasse ainda mais, tendo as tarefas executadas em 7,16min, 10,61% a menos quando comparado com a configuração pré-determinada pelo *Openmosix*, o que deixa claro que a tarefa de definir a carga de trabalho em relação ao equipamento disponível interfere no desempenho global do *cluster*. Maiores detalhes desta configuração podem ser vistos no Figura 7.20 a seguir.

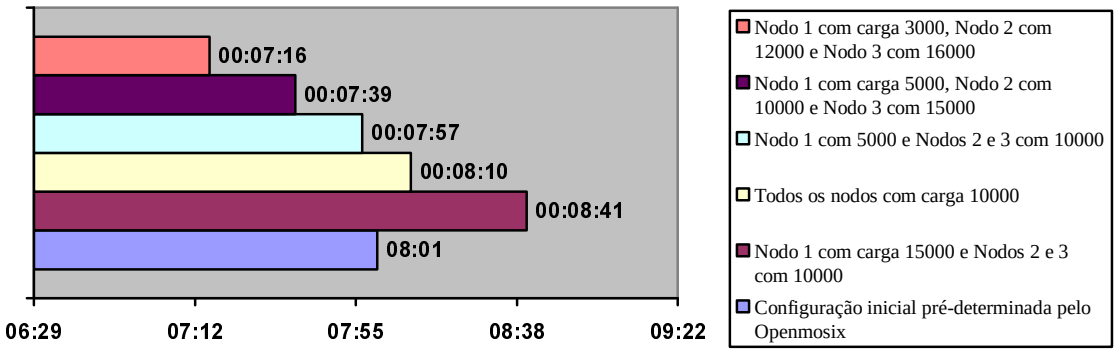

Figura 7.18: Testes com cargas de trabalho distintas

7.3.5 Experimento com máquinas interligadas ponto a ponto

Como descrito no item “Configuração de *Hardware*” anteriormente, foi montado um segundo *cluster,* este composto por duas máquinas (Máquina 4 + Máquina 5), interligadas através de uma rede *ethernet* ponto-a-ponto, como forma de verificarmos o desempenho do *Openmosix* em uma situação em que, além de termos máquinas melhores, temos uma conexão de rede mais rápida e eficiente. Estes testes seguiram o padrão dos testes anteriores, onde convertemos um grupo de faixas de áudio, distintas e iguais, verificando o tempo gasto para a conclusão dos trabalhos.

Quando executamos os trabalhos de conversão de faixas iguais apenas no Notebook, levamos em média, 7:18min para que todas as músicas fossem convertidas. O mesmo processo, só que agora trabalhando em conjunto com mais um PC (Atlhon XP1600, 512Mb), ganhamos cerca de 100% de performance, tendo as faixas de áudio processadas em 3,58min ou seja, levou-se 3,19min a menos que quando executado apenas no Notebook. Voltamos a lembrar que, aqui, além dos dispositivos de processamento serem melhores que os testados anteriormente, as placas de comunicação também tem um desempenho melhor que as demais, com taxa de transmissão de 100Mbs.

Outro teste efetivado, nos moldes do anterior, só que agora utilizando os equipamentos de menor capacidade interligados ponto-a-ponto, também tivemos algumas diferenças no desempenho do *cluster*.

Interligando ás máquinas 2 e 3 (as mesmas dos primeiros testes), diretamente uma na outra, e convertendo um grupo de 16 faixas de áudio iguais, a partir do Nodo 3, com as configurações de carga de trabalho definidas automaticamente pelo *Openmosix*, tivemos os trabalhos concluídos em 5:59min, enquanto que o mesmo processo executado com as máquinas interligadas através de um *hub*, foram concluídas em 6,46min, quase 1 minuto de diferença ou em percentuais, 13,46% de ganho de performance, ilustrado através da Figura 7.21.

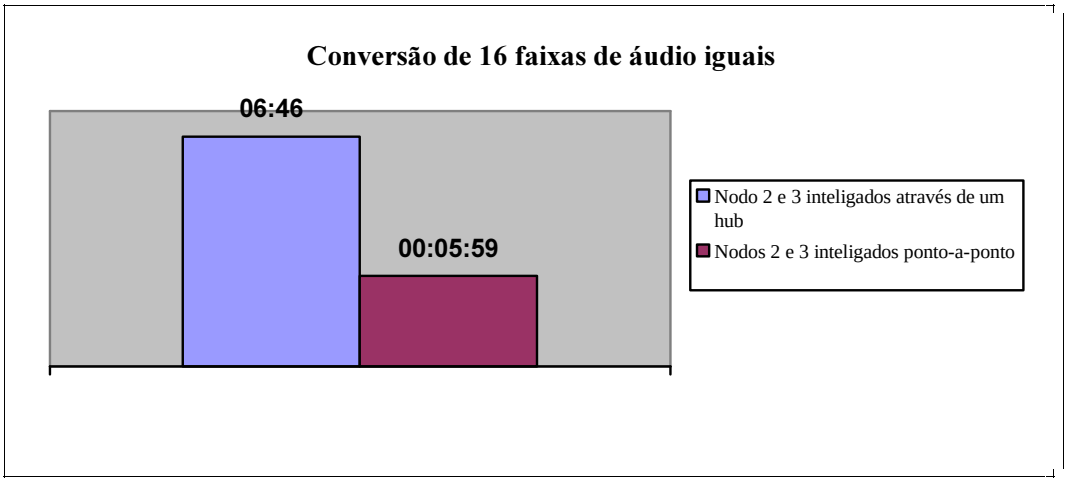

Figura 7.19: Diferenças na execução de tarefas em relação ao tipo de comunicação

Nestes testes interligados ponto-a-ponto, onde os processos de comunicação não sofrem interferência de outros dispositivos, notamos um aumento de desempenho na relação em que comparamos o desempenho dos equipamentos ligados ponto-a-ponto com os mesmos equipamentos interligados através de um *hub*, deixando evidente que toda e qualquer interferência na comunicação, tende a afetar o desempenho global do *cluster*. Neste caso, o fato desta perda de desempenho esta diretamente ligada ao dispositivo de comunicação (*Hub*), uma vez que sua taxa de transmissão era de 10Mps enquanto a ligação ponto-a-ponto a taxa de transmissão era de 100Mps. A simples troca deste dispositivo por um equipamento melhor, por exemplo um *Switch* com transmissão de 100Mps, poderia fazer com que estes dados fossem os mesmos tanto para a ligação ponto-a-ponto como para a conexão com um dispositivo de rede.

7.3.6 Relação entre equipamentos descontinuados com um PC mais moderno

Fazendo um comparativo entre uma máquina de melhor capacidade com relação às máquinas disponibilizadas pela instituição, que são inferiores em termos de *hardware*, podemos observar uma boa diferença na execução dos trabalhos, chegando na maioria das combinações, a ter um resultado melhor no ambiente em *cluster* do que a máquina mais moderna rodando os processos individualmente..

A comparação aqui realizada foi entre um computador com processador AMD *Athlon* 1400Mhz, com 512Mb de memória RAM (Máquina 5) em relação a configuração em *cluster* que obtivemos o melhor resultado anteriormente, ou seja, a máquina mais moderna em relação a duas máquinas com processador AMD *Athlon* 800Mhz, uma com 128Mb e outra com 256Mb de memória RAM, interligadas através de uma rede ponto a ponto.

Ao converter um grupo de músicas iguais (16 faixas), tivemos os trabalhos concluídos numa única máquina (mais moderna) em 8:54 minutos, enquanto que o mesmo processo foi executado no *cluster* com duas máquinas (No 2 + No 3 ponto-a-ponto) em 5:59 minutos, ou seja, 2,54 minutos (34,54%) mais rápido na operação em *cluster*. Ainda que, utilizando as mesmas máquinas interligadas por intermédio de um *hub*, que como vimos anteriormente, aumenta um pouco o tempo de resposta, tivemos um desempenho melhor do que uma única máquina processando. Neste caso, o processo *em* cluster foi concluído em 1,57 minutos mais rápido.

Apesar da Máquina 5 ter em tese a melhor configuração de *hardware* de todas ás máquinas testadas, os melhores resultados rodando os processos de conversões em uma única máquina foram obtidos na Máquina 4 (*Notebook*). Sendo assim, fizemos um comparativo com relação a está máquina também, comparando como anteriormente, com os processos rodando em paralelo. Neste caso, baixou um pouco a diferença no tempo de resposta, mas ainda foi melhor o desempenho no ambiente em *cluster* computacional. Enquanto o melhor resultado individual foi de 7:09min no n*otebook*, o tempo de resposta em *cluster* ainda foi 1,09min mais rápido. Na Figura 7.20 a seguir, temos uma relação entre as máquinas de melhor capacidade de processamento, com ás máquinas mais antigas, já fora de mercado, rodando em *cluster*.

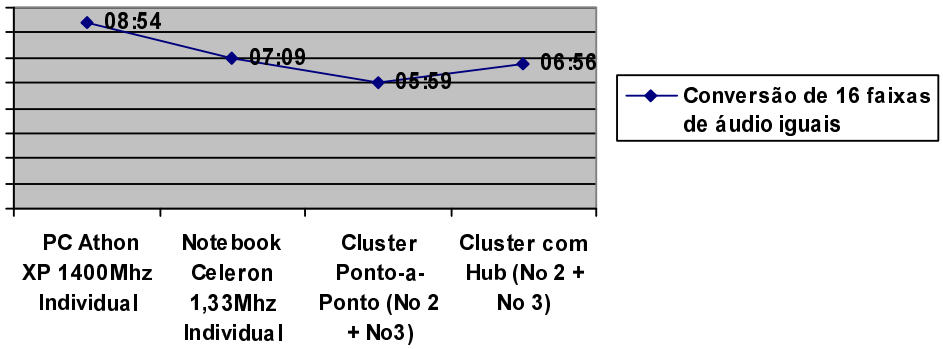

Figura 7.20: Comparativo entre operações em *cluster* e máquinas individuais

Da mesma forma que as músicas iguais, onde o desempenho foi considerado bom, os resultados em relação aos testes efetivados rodando um *script* descrito anteriormente também foram semelhantes. Neste caso, o desempenho do *cluster* em relação a uma única máquina de melhor capacidade, fora do ambiente de *cluster* foi em torno de 21,54% melhor para a execução com duas máquinas interligadas e 31,83% melhor, quando interligados três micros em conjunto, ou seja, enquanto rodando os *scripts* em apenas uma máquina, levou-se 9:33 minutos para o término dos trabalhos, o mesmo processo foi executado em 7:32min e 6:36min respectivamente a duas e três máquinas trabalhando em paralelo.

Sendo assim, podemos observar que utilização de *clusters* nos experimentos descritos acima nos dá uma notável diferença no tempo de resposta, mesmo que utilizando máquinas de menor capacidade computacional. Entretanto, observamos que em situações em que não existem nenhum tipo de comunicação entre processos (neste caso, o script rodado), o ganho de performance foi melhor conforme o eram incluídas máquinas ao agrupamento, mesmo que estas máquinas tivessem capacidade de processamento menor que as demais, ao contrário da conversão de áudio que ao ser incluído um nodo inferior, diminuía um pouco o rendimento do *cluster*.

CONCLUSÃO

Vimos que a evolução dos computadores de alto desempenho foi impulsionada pela necessidade de processar uma grande quantidade de informações em um tempo relativamente pequeno. Durante muitos anos, a computação evoluiu no sentido de desenvolver novas tecnologias de *hardware* em máquinas cada vez mais velozes.

O surgimento das arquiteturas paralelas e sistemas distribuídos trouxeram um novo paradigma em relação ao processamento computacional, melhorando muito o desempenho de processos que exigiam grande poder de processamento, através do principio da divisão de tarefas entre os processadores. Entretanto, este desenvolvimento custava muito caro, sendo restrito a poucas empresas e instituições. Dessa forma, surgiram os *clusters* computacionais, como forma de aproveitar as tecnologias já existentes, utilizando computadores comuns interligados através de uma rede como forma de dividirem as tarefas que até então só eram executadas por supercomputadores.

Ao longo deste trabalho, observamos que a utilização de estações de trabalho na solução de problemas de alta disponibilidade e alta performance de computação, principalmente nos últimos anos, tem demonstrado um crescimento significativo e tem impulsionado o desenvolvimento e estudos de tecnologias nesta área, sendo adotada cada vez mais, como uma solução de alto desempenho com um custo-benefício relativamente baixo.

Quanto aos experimentos executados, podemos levantar como uma das principais vantagens na utilização de um ambiente de *cluster*, neste caso, o *cluster Openmosix*, a sua

facilidade de implementação, onde inclusive uma pessoa sem grandes conhecimentos nesta área pode ser capaz de configurar esta tecnologia, além de ser um ambiente que não requer que as aplicações tenham que ter sido previamente desenvolvidas para um ambiente paralelo, trazendo assim, benefícios na execução de aplicativos comuns aos usuários a um custo relativamente baixo. Entretanto, observamos que a montagem de *clusters* heterogêneos requer um pouco mais de atenção, onde neste caso, deve ser feito um estudo mais detalhado quanto às configurações das máquinas, deixando bem definido a carga de trabalho de cada estação conforme seus recursos de *hardware*, como forma de evitar possíveis perdas de performance.

Em relação ao desempenho obtido, apesar de, em algumas situações termos perdido um pouco de desempenho com a adição ou exclusão de máquinas, na maioria das vezes foi obtido um resultado similar ou melhor do que os experimentos executados por máquinas de maior capacidade de processamento, rodando individualmente, comprovando que é possível obtermos um bom desempenho utilizando equipamentos inferiores (muitos, inclusive que não eram nem se quer mais utilizados) trabalhando em conjunto.

Observamos também, que o acréscimo de mais nodos ao agrupamento não aumenta na mesma proporção o tempo de resposta, ou seja, executando uma tarefa em um *cluster* com duas máquinas não tivemos o dobro de velocidade do que o mesmo processo sendo executado em uma única máquina. O ideal seria que o ganho de velocidade fosse proporcional à quantidade de nós utilizados**,** mas diferenças de hardware e atrasos em comunicações tendem a afetar estes dados, além de existir um limite para a paralelização dos processos, que muda conforme o tipo de tarefa que está sendo executada. Como a maioria dos experimentos realizados foi com equipamentos heterogêneos, não podemos fazer um comparativo maior com relação ao ganho de desempenho conforme o número de máquinas, visto que as diferenças dos equipamentos utilizados nos davam resultados diferentes conforme as máquinas que eram adicionadas.

Se por um lado, temos um bom desempenho em algumas situações, e como ponto forte desta tecnologia a escalabilidade, baixo custo e facilidade de implementação, por outro temos algumas limitações quanto ao uso deste ambiente, como a velocidade de comunicação, a latência gerada pelos dispositivos de rede, softwares impróprios, etc. Assim, dentro deste contexto, existem ainda grandes desafios a serem enfrentados no projeto de um *cluster* como

aplicações que possuem alta comunicação entre processos, o que prejudica a execução de tarefas. Porém, com o grande avanço das redes de interconexões, bem como a evolução rápida de componentes comerciais, fazem com que os *clusters* se tornem cada vez mais eficientes, substituindo aos poucos os supercomputadores e máquinas MPPs, tornando viável sua implementação sem grandes investimentos, como soluções domésticas, acadêmicas e comerciais.

Quanto a trabalhos futuros, fica a sugestão em relação ao uso de ferramentas de *benchmarks*, na qual não foi possível concluir sua utilização neste trabalho. Outra sugestão é quanto ao uso de outros meios de comunicação entre os nodos, o que mudaria bastante os resultados obtidos neste experimento, além da inclusão de mais equipamentos ao agrupamento, fazendo uma análise de escalabilidade com grandes quantidades de máquinas.

REFERÊNCIAS BIBLIOGRÁFICAS

ALECRIM, Emerson. **Cluster:** Principais conceitos. Info Wester, 2004. Disponível em: <www.infowester.com/cluster.php>. Acesso em: 18 mar. 2006.

AMORIM, Cláudio Luiz; BARBOSA, Valmir Carneiro; FERNANDES, Edil Severiano Tarares. **Uma introdução à computação paralela e distribuída**. Campinas: UNICAMP, IMECC, 1988. 256p.

BESSANI, Alysson Neves. **O padrão Umiop como base para comunicação de grupo confiável em sistemas distribuídos de larga escala**. Florianópolis: 2002. 105p. Dissertação (Mestre em engenharia elétrica) – Centro de Pós Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, 2002. Disponível em: <http://www.das.ufsc.br/~neves/arquivos/dissertacao.pdf>. Acesso em 07 mai. 2006.

BEOWULF, 2004. Disponível em: <http://www.beowulf.org>. Acesso em 26 mai. 2006.

BOOKMAN, Charles. **Agrupamento de computadores em Linux**: aprenda a construir e manter grupos de computadores com Linux. Rio de Janeiro: Ciência Moderna, 2003. 240p.

BRASIL ESCOLA. **Revolução do Computador:** Os primeiros computadores. [S.1: Brasil Escola], 2006. Disponível em: <http://www.brasilescola.com/informatica/revolucao-do-computador.htm>. Acesso em: 02 abr. 2006.

BUYYA, Rajkumar: **High Performance Cluster Computing**: Architectures and Systems. Volume 1. New Jersey: Prentice-Hall, 1999. 849p.

CECHIN, Sergio; NETTO, João. Ferramentas de Avaliação de Desempenho de Sistemas Computacionais. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 1ª, 2001. **Anais**... Gramado: SBC/ Instituto de informática da UFRGS/ Faculdade de Informática da PUCRS/UNISINOS, 2001. p.127-150.

CLUSTER Benchmarks Web Page, 2006. Disponível em: <http://www.mfn.unipmn.it/~mino/cluster/benchmarks/>. Acesso em 09 jun. 2006.

CUMULVS, 2005. Disponível em: <http://www.csm.ornl.gov/cs/cumulvs.html>. Acesso em 14 mai. 2006.

DANTAS, Mario. **Ambientes Distribuídos de Alto Desempenho**: Clusters e Grades Computacionais. [S.1: Portal Brasileiro sobre computação de alto desempenho, 2006. Disponível em: <http://www.gridcomputing.com.br/tiki-ndex.php?page=Getting%20Started>. Acesso em 12 mai. 2006.

DANTAS, Mario. **Computação Distribuída de Alto Desempenho**: Redes, Clusters e Grids Computacionais. Rio de Janeiro: Axcel Books, 2005. 262p.

GOULART, Ademir. **Sistemas Distribuídos e Comunicação em Grupo**. Itajaí: 2002. 67p. Disciplina Tópicos Especiais em Computação, UNIVALI, 2002. Disponível em : <http://www.din.uem.br/~cfmoro/download/APOSTILA_AdemirGoulart_SD_CGv10.PDF>. Acesso em 08 mai. 2006.

GUIA de estruturação e Administração do Ambiente de Cluster**.** Brasilia, 2006. Disponível em: <*http://guialivre.governoeletronico.gov.br/guiaonline/guiacluster*>. Acesso em 15 set. 2006.

IEEE, 2006. Disponível em: <http://www.feg.unesp.br/~caeel/about/aboutieee.htm>. Acesso em: 09 jun. 2006.

LINHALIS, Flavia; MAYB, Iara Fiats. **PVM e MPI**. São Paulo: 1998. 47p. Monografia - Departamento de Ciências da Computação e Estatísticas, USP-SP, 1998. Disponível em: <http://black.rc.unesp.br/gpacp/descricao.html>. Acesso em 02 mai. 2006.

MINDCRAFT. **Web Stone**: The Benchmarks for Web Servers. [S.l: Mindcraft], 1998. Disponível em: <http://www.mindcraft.com/benchmarks/webstone/> Acesso em: 28 mai. 2006.

MPI, 2006. Disponível em: <http://www.mpi-forum.org>. Acesso em 16 mai. 2006.

MPIBENCH Home Page, 2006. Disponível em: <http://icl.cs.utk.edu/projects/llcbench/mpbench.html> Acesso em 08 jun. 2006.

OPENMOSIX Project, 2006. Disponível em <openmosix.sourceforge.net>. Acesso em 27 mai. 2006.

PITANGA, Marcos. **Computação em cluster**: o estado da arte da computação. Rio de Janeiro: Brasport Livros e Multimídia Ltda, 2003. 322p.

PITANGA, Marcos. **Construindo supercomputadores com linux**. 2ª edição. Rio de Janeiro: Brasport Livros e Multimídia Ltda, 2004. 292p.

PVM, 2006. Disponível em:<www.csm.ornl.gov/pvm/>. Acesso em 14 mai. 2006.

ROSE, César A.F. De. Arquiteturas Paralelas. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 1ª, 2001. **Anais**... Gramado: SBC/ Instituto de informática da UFRGS/ Faculdade de Informática da PUCRS/UNISINOS, 2001. p.3-33

SKAMPI Home Page, 2006. Disponível em:<http://www.ira.uka.de/~skampi/>. Acesso em 10 jun. 2006.

TANENBAUM, Andrew S. **Organização estruturada de computadores**. 4ª edição. Rio de Janeiro: LTC, 2001. 398p.

TOP500 SuperComputer Site, 2005. Disponível em: <http://www.top500.org/lists/2005/11>. Acesso em: 05 abr. 2006.

WEBER, Raul Fernando. **Fundamentos de arquitetura de computadores**: Série livros didáticos, número 8. Porto Alegre: Instituto de informática da UFRGS, Editora Sagra Luzzatto, 2000. 262 p.

XPVM, 2006. Disponível em: <http://www.netlib.org/utk/icl/xpvm/xpvm.html>. Acesso em 15 mai. 2006.

ZELENOVSKY, Ricardo; MENDONÇA, Alexandre. **Processadores para o próximo milênio**. 2º parte. [S.1: Clube do Hardware], 2001. Disponível em: <http://www.clubedohardware.com.br/printpage/993>. Acesso em: 10 abr. 2006.