UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

CONSTRUÇÃO E CONFIGURAÇÃO DE UM AGLOMERADO COM COMPUTADORES EM LABORATÓRIOS DE INFORMÁTICA

TRABALHO DE GRADUAÇÃO

Carlos Rafael Röhrig da Costa

Santa Maria, Rio Grande do Sul, Brasil
2009

CONSTRUÇÃO E CONFIGURAÇÃO DE UM AGLOMERADO COM COMPUTADORES EM LABORATÓRIOS DE INFORMÁTICA

por

Carlos Rafael Röhrig da Costa

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Prof. Antonio Marcos de Oliveira Candia

Trabalho de Graduação N. 281
Santa Maria, RS, Brasil
2009

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,

aprova o Trabalho de Graduação

CONSTRUÇÃO E CONFIGURAÇÃO DE UM AGLOMERADO COM COMPUTADORES EM LABORATÓRIOS DE INFORMÁTICA

elaborado por

Carlos Rafael Röhrig da Costa

como requisito parcial para obtenção do grau de

Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____

Prof. Antonio Marcos de Oliveira Candia

(Orientador)

_____

Prof. Raul Ceretta Nunes

_____

Profª. Márcia Pasin

Santa Maria, 14 de julho de 2009.

“Dois importantes fatos, nesta vida, saltam aos olhos; primeiro, que cada um de nós sofre inevitavelmente derrotas temporárias, de formas diferentes, nas ocasiões mais diversas. Segundo, que cada adversidade traz consigo a semente de um benefício equivalente. Ainda não encontrei homem algum bem-sucedido na vida que não houvesse antes sofrido derrotas temporárias. Sempre que um homem supera os reveses, torna-se mental e espiritualmente mais forte... É assim que aprendemos o que devemos à grande lição da adversidade. ”

- Andrew Carnegie a Napoleon Hill

AGRADECIMENTOS

Agradeço, primeiramente, a Deus, pela vida, e por todos os eventos nela acontecidos, que me moldaram e tornaram-me exatamente a pessoa que sou.

Agradeço aos meus pais, Antônio e Janete, que amo muito, pelo apoio ininterrupto, por terem me dado uma base forte de educação e bons princípios, além de todo amor e carinho que uma pessoa necessita para se tornar um adulto de bem, correto. Graças a esta base que eu pude chegar onde estou e ser a pessoa que sou hoje. Eles me ajudaram nos primeiros passos e me deram a confiança para caminhar sobre minhas próprias pernas. A minha irmã Maira que, apesar de brigarmos um pouco, eu gosto muito dela. A toda minha família pelo ambiente saudável em que cresci e pelo exemplo de pessoas. E em especial, pelo exemplo de vida, força, superação, bondade e consciência do meu avô Elói.

Agradeço a minha namorada Maíne, pela compreensão, amizade, amor, carinho, apoio, força, conselhos, “puxões de orelha”, broncas, mimos, atenção, companheirismo e por sempre ter as palavras certas no momento certo... Enfim, por ser esta pessoa, exatamente esta pessoa, que me inspira a sempre buscar ser um homem melhor e buscar o melhor. E com certeza, na presença dela, eu tive um amadurecimento e um esclarecimento da visão sobre o futuro. Maíne, eu te amo!

Agradeço ao meu orientador Antonio Marcos de Oliveira Candia pelo apoio e pela disposição a ajudar. Permitindo com que este trabalho fosse realizado.

Agradeço ao Everton e a toda a equipe do LINCE por permitir desenvolver meu trabalho no laboratório, utilizar os recursos e oferecer um ambiente favorável para a elaboração de um bom trabalho. Ao Tiago Badin, por todas as dicas e conhecimento que me passou durante todo o tempo que trabalhamos no LINCE, além do apoio dado neste trabalho, tanto em conselhos quanto em pensamento positivo e reza forte.

Agradeço aos meus colegas por todos os momentos que permitiram adquirir conhecimento e estabelecer amizades que nunca se apagarão. Aos amigos que sempre me apoiaram, ajudaram e aconselharam, Jesse e Henrique, com os quais pude aprender e aos quais devo muito. A todos os meus amigos que conviveram comigo e participaram do meu crescimento como pessoa durante este período da vida chamado *Faculdade.*

Aos mestres que nos ensinaram e nos instigaram a extrairmos de nós o nosso melhor, o meu sincero agradecimento.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

CONSTRUÇÃO E CONFIGURAÇÃO DE UM AGLOMERADO COM COMPUTADORES EM LABORATÓRIO DE INFORMÁTICA

Autor: Carlos Rafael Röhrig da Costa
Orientador: Prof. Antonio Marcos Candia
Local e data da defesa: Santa Maria, 15 de julho de 2009.

Com o passar dos anos, avanços tecnológicos dos componentes dos computadores fazem com que novos computadores mais poderosos que os atuais sejam produzidos. Este avanço nos deixa uma dúvida: O que fazer com os computadores antigos? Como aproveitá-los de forma que ainda possam ser úteis?

Este trabalho tem por objetivo utilizar computadores antigos e ociosos em laboratórios de informática para construção de um aglomerado. Este aglomerado é uma alternativa de utilização de máquinas que não estavam sendo usadas, interligando essas máquinas e obtendo maior poder computacional alocando recursos em rede.

O aglomerado contou com computadores interligados em rede, sistema operacional clusterKNOPPIX, baseado em Debian, OpenMosix como ferramenta de gerenciamento de troca de processos entre os nós e controle de desempenho do sistema. Os resultados deste trabalho poderão auxiliar o desenvolvimento de pesquisas sobre aglomerados, além de possibilitar o uso desses computadores de forma a obter melhor processamento que não é oferecido quando os computadores são utilizados isoladamente.

Palavras-chave: Aglomerados, OpenMosix, migração preemptiva, *Cluster*, Sistemas Distribuídos.

ABSTRACT

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

CONSTRUÇÃO E CONFIGURAÇÃO DE UM AGLOMERADO COM COMPUTADORES EM LABORATÓRIO DE INFORMÁTICA
Author: Carlos Rafael Röhrig da Costa
Advisor: Prof. Antonio Marcos Candia
Place and date of Presentation: Santa Maria, July, 14, 2009.

Along the years, technological advances about computer components make new computers more powerful than the currently produced. This advance bring us doubts: what do about the old computer? How make use of them usefully?

This work aims using old idle computers in informatics laboratories for a cluster construction. This cluster is an alternative of utilization of machines which were not being used, linking them and obtaining a larger computational power, allocating network resources.

The cluster counted on linked network computers, operational system cluster KNOPPIX, based on Debian, OpenMosis as management tool of process changing among the nodes and performance control of the system. This work's results may help the development of researches about cluster besides of enable the use of these computers to obtain a better processing which is not offered when computers are singly used.

Keywords: Clusters, OpenMosix, preemptive migration, Distributed Systems.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

1.1 Contextualização

A necessidade de efetuar operações cada vez mais complexas no menor tempo possível fez com que os desenvolvedores buscassem novas formas de aperfeiçoar suas tecnologias, para obtenção de mais poder computacional. Os avanços dos processadores e dos componentes de computadores pessoais foram evidentes nas duas últimas décadas, fazendo com que PCs (Computadores pessoais, do inglês *Personal Computers*) tenham poder computacional para operações complexas, mas ainda sem ser comparado com os *mainframes*.

A idéia de utilizar vários computadores ou vários processadores para executar uma tarefa que exija grande poder computacional surgiu na década de 1960, quando a IBM (International Business Machines) conectou vários *mainframes* para conseguir uma solução economicamente viável de paralelismo. Este paralelismo era possível devido ao uso de sistemas que forneciam gerenciamento de distribuição de carga entre os *mainframes*, na época era usado o HASP (*Houston Automatic Spooling Priority*) [IBM.com, 2009] e logo após o JES (*Job Entry System*) [ROSENTIEL, 2009]. Hoje em dia a IBM utiliza o sistema *Parallel Sysplex* [IBM.com, 2009] que permite uma melhora no desempenho dos aglomerados *mainframes*.

Uma alternativa de baixo custo para criação de sistemas que suportassem maior carga computacional foi a construção de sistemas distribuídos. Um sistema distribuído comporta vários computadores conectados em rede configurados para trabalhar em conjunto com a finalidade de alta disponibilidade, maior desempenho ou balanceamento de carga.

Através de configurações de *hardware* ou *software*, esse conjunto de computadores obtém maior desempenho, às vezes podendo se equiparar a *mainframes*. A estrutura é transparente para o usuário, que interage com o sistema como se fosse apenas um computador. Essa arquitetura é conhecida como computação em *cluster*, mais comumente chamada de *cluster*, ou também traduzida para aglomerado ou agregado.

Por ser uma solução barata e com bom desempenho, as pesquisas sobre o assunto foram crescendo e muitos aglomerados foram construídos em universidades ou empresas que tinham grande carga computacional. Como mostra a figura 1, a utilização de aglomerados cresceu com grande força nos últimos anos e ganha

destaque entre as arquiteturas consideradas as mais poderosas pelo site *top500.org* e no ano de 2008 garantiu 82% de presença dentre os quinhentos computadores listados, conforme mostra a figura 2.

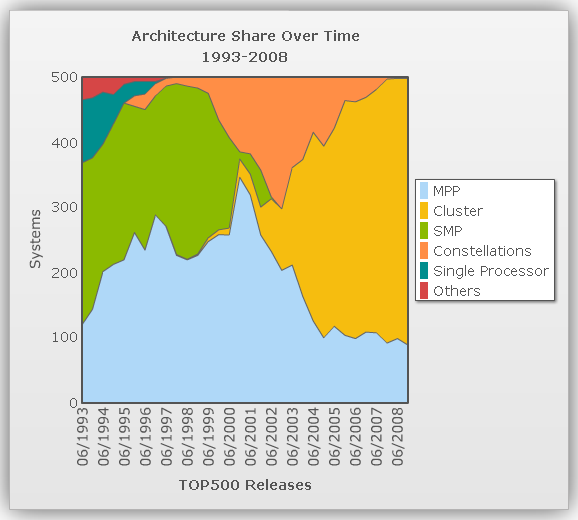


FIGURA 1. Distribuição das arquiteturas no TOP 500 durante os anos
FONTE: top500.org

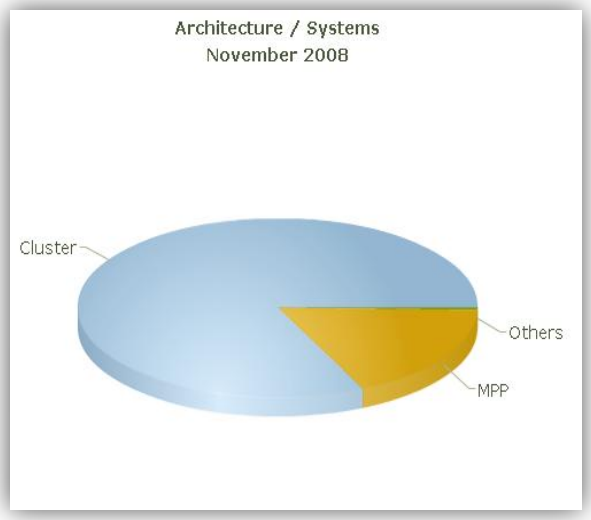


FIGURA 2. Porcentagem de aglomerados presentes na lista do TOP 500
FONTE: top500.org

Vários tipos de aglomerados foram desenvolvidos. Alguns merecem destaque, dentre eles está o aglomerado chamado Beowulf [BEOWULF.org, 2009], que tem como idéia principal utilizar componentes “de prateleira”. Componentes “de prateleira” são componentes com características e especificações normais aos padrões de mercado para constituir o sistema e utilização de Sistema Operacional livre (Linux ou UNIX). Este modelo ganhou destaque pela facilidade de configuração de *software*, por se tratar de um sistema operacional livre e por possibilitar gerenciar o *hardware* de forma mais transparente. Também garante facilidade tanto para trocar componentes quanto para inserir novas máquinas, alem de ser uma solução barata de processamento paralelo que permite uma boa razão entre custo e benefício.

Este trabalho visa poder utilizar melhor os recursos do laboratório, pois os computadores mais velhos e com menor poder computacional estão sem uso e isto é visto como um desperdício. Os computadores velhos estavam sendo utilizados como terminais BSD conectados em nossos servidores, porém, nossos servidores já estavam sobrecarregados, e alguns terminais estavam parados. E uma possibilidade de utilizá-los, de forma que maior desempenho para aplicações executadas no laboratório é combiná-los em um aglomerado de forma a utilizar não só o poder

individual de cada máquina, mas sim o poder computacional resultante da união das mesmas.

Ferramentas que auxiliam o gerenciamento e utilização do aglomerado também foram estudadas, dentre elas o MOSIX e o OpenMosix, que são ferramentas que monitoram a carga em cada um dos nós do aglomerado e fazem balanceamento das tarefas entre eles utilizando migração preemptiva através de algoritmos que calculam o custo de migração e a disponibilidade de recursos em cada nó, fazendo com que as tarefas sejam executadas no nó mais propício.

1.2 Objetivos do trabalho

O objetivo deste trabalho é encontrar uma solução eficiente para construção e configuração de um aglomerado aproveitando máquinas antigas que não estavam sendo utilizadas no LINCE (Laboratório de Informática do Centro de Educação) e se tornariam obsoletas. Para construir o aglomerado, é necessário escolher qual a ferramenta de gerenciamento de recursos será utilizada. Os modelos de aglomerados utilizam ferramentas para obter maior desempenho utilizando as máquinas da forma mais otimizada. Nos próximos capítulos trataremos destas ferramentas, referente a detalhes de instalação, configuração e características de cada uma das ferramentas pesquisadas.

O aglomerado deve utilizar táticas de balanceamento de carga, sendo utilizado um computador para gerenciar o aglomerado e outros computadores como trabalhadores.

Os computadores isolados tem baixo poder computacional, mas combinados em um aglomerado podem obter resultados que permitem equiparar nosso aglomerado com um computador com configurações consideradas atuais. Este aglomerado não ambiciona ser comparado com o poder computacional de mainframes ou supercomputadores, mas ter um ganho de desempenho em comparação com uma máquina trabalhando isoladamente.

1.3 Estrutura do texto

Este trabalho está organizado em cinco capítulos. No segundo capítulo, apresenta-se a revisão bibliográfica referente aos termos constantes no decorrer do texto. O terceiro capítulo descreve o projeto do aglomerado, as principais características de implementação e as ferramentas utilizadas para o desenvolvimento do aglomerado. No quarto capítulo, encontram-se detalhes de implementação e configuração do sistema, descrição dos testes de desempenho e respectivos resultados. Por fim, o quinto capítulo apresenta as conclusões, a análise sobre o trabalho implementado e as possíveis complementações sobre a estrutura montada para prestação de serviços para o laboratório.

2 FUNDAMENTOS TEÓRICOS

2.1 Sistema Distribuído

Sistema distribuído [COULOURIS, 2007] é caracterizado como um conjunto de computadores interligados entre si em uma rede onde seus componentes se comunicam e coordenam suas ações através de troca de mensagens.

Um dos motivos para construção de sistemas distribuídos é o compartilhamento de recursos. Os recursos podem ser gerenciados por servidores e acessados por clientes ou encapsulados como objetos e acessados por outros objetos. Recurso pode ser entendido como componentes de *hardware*, como discos e impressoras, ou também por *software*, como arquivos, bancos de dados e objetos de dados de diversos tipos.

Quando se constrói um sistema distribuído, existem muitos problemas que devem ser superados para que a evolução e o desenvolvimento seja mais fácil e o sistema seja mais abrangente. Segundo COULOURIS (2007) podemos destacar:

Heterogeneidade: Permitir que usuários acessem recursos através de um conjunto heterogêneo de computadores. Heterogeneidade se aplica sobre redes, *hardware* de computador, sistemas operacional, linguagem de programação e programas feitos por diferentes desenvolvedores.

Sistemas Abertos: Um sistema computacional é aberto quando ele pode ser ampliado e reimplementado de diversas maneiras. Para se obter sistemas abertos deve-se ter especificação e documentação das principais interfaces de *software* dos componentes do sistema para que os desenvolvedores possam construí-las com uma padronização.

Segurança: Muitas informações contidas em um sistema distribuído são muito importantes para seus usuários, portanto deve-se ter o cuidado para que essas informações tenham segurança nos seguintes conceitos: *confidencialidade* (proteção contra o acesso de pessoas não autorizadas), *integridade* (proteção contra alterações ou danos) e *disponibilidade* (proteção contra interferência nos meios de acesso ao recurso).

Escalabilidade: O sistema permanece eficiente quando existe um aumento significativo no número de recursos e no número de usuários.

Tratamento de falhas: Quando um componente do sistema falha, os resultados podem ser incorretos ou os programas podem parar antes de terem

concluído sua execução. As técnicas de tratamento de falhas podem ser divididas em: *Detecção de falhas* (consiste em fazer a verificação para detectar dados corrompidos em uma mensagem ou um arquivo); *Mascaramento de falhas* (falhas detectadas podem ser ocultas sem que os resultados sejam afetados); *Tolerância a falhas* (detectar e mascarar falhas em sistemas muito grandes pode ter custos altos, portanto existem sistemas que são projetados para tolerar falhas, como por exemplo, comunicar ao usuário sobre o erro, deixando-o livre para tentar novamente); *Recuperação de falhas* (consiste em criar programas que permitam que o estado dos dados permanentes possa ser recuperado ou retrocedido após uma falha); *Redundância* (uma forma de contornar possíveis falhas utilizar componentes redundantes, ou seja, várias formas de acessar recursos ou vários locais para armazenar recursos).

Os sistemas distribuídos visam fornecer alta disponibilidade perante falhas de *hardware*. A disponibilidade de um sistema é medida pela proporção que o sistema se mantém ativo. Quanto maior a disponibilidade, maior a transparência das falhas para os usuários.

Concorrência: Os recursos fornecidos pelo sistema podem ser compartilhados pelos clientes em um sistema distribuído. É possível que um recurso seja requisitado por vários clientes ao mesmo tempo. Para que haja escalabilidade é preciso tratar essas requisições de acesso e processá-las concorrentemente. Esse procedimento deve ser sincronizado de forma que a consistência dos dados não seja afetada.

Transparência: Usuário ou programadores de aplicativos para o sistema distribuído devem vê-lo como um todo, ao invés de uma coleção de componentes independentes, através da ocultação da separação destes componentes.

2.2 Aglomerados

Aglomerado é a definição para a união de dois ou mais computadores ligados em rede e cooperando na execução de tarefas, trocando mensagens entre si. Dentro do aglomerado cada computador é chamado de nó ou nodo, pois dentro da rede assume a posição de parte integrante do sistema. A construção de aglomerados visa utilizar melhor os recursos computacionais e disponibilizar um serviço mais estável.

Segundo PITANGA (2003), os aglomerados podem ser divididos em categorias conforme sua finalidade (vide Tabela 1):

- *Aglomerados de Alta Disponibilidade* – São aglomerados que tem por objetivo manter um serviço disponível o maior tempo possível, utilizando a redundância dos dados transparente ao usuário do sistema. Consiste em ter o serviço ou os dados em vários nós, e quando um nó falhar, imediatamente outro nó passa a substituí-lo;
- *Aglomerados de Balanceamento de Carga* – Este tipo tem como característica possuir vários computadores interligados que possuam técnica para gerenciar o consumo do processador e prover uma distribuição de trabalho entre os processadores dentro do aglomerado, de forma que processadores sobrecarregados possam distribuir suas tarefas entre processadores ociosos ou com menos trabalho;
- *Aglomerados Híbridos* – São os aglomerados que mesclam características entre os dois citados anteriormente. Tem como características principais a redundância de dados e serviços e o balanceamento de carga entre os nós escravos. A redundância pode ser aplicada aos nós gerentes, de forma que se o nó gerente ativo falhar, um nó gerente reserva assume o trabalho, evitando a indisposição do serviço por falha de gerência.
- *Processamento Paralelo* – Este tipo de computação permite maior desempenho e disponibilidade para as aplicações. Consiste em dividir grandes tarefas em partes menores e distribuí-las entre várias máquinas interligadas ou entre vários processadores.

Outras classificações podem ser adotadas levando em conta certas características. Dentre elas podemos citar quanto a utilização dos nós, quanto ao hardware utilizado e quanto a configuração dos nós.

Quanto a utilização dos nós, podemos definir o aglomerado como:

- *Dedicado* – Quando todos os recursos do nó são alocados para executar a computação requisitada;
- *Não-Dedicado* – Quando os recursos podem ser compartilhados, de forma que estejam disponíveis para outras finalidades além da utilização em processamento do aglomerado.

Classificação	Tipos	Características
Finalidade	*Alta disponibilidade*	Manter serviço disponível maior tempo possível, através de replicação de recursos.
	Balanceamento de Carga	Utilizar os recursos disponíveis de forma que não haja sobrecarga em uma máquina e ociosidade em outras, distribuindo as tarefas entre os nós.
	Híbridos	Visa combinar as características dos dois anteriormente citados, distribuindo carga e replicando os recursos gerenciadores do sistema.
	Processamento Paralelo	Consiste em dividir as tarefas em tarefas menores e distribuí-las entre os nós do aglomerado.
Utilização dos nós	*Dedicado*	Todos os recursos alocados para a computação requisitada.
	Não-dedicado	Quando os recursos são compartilhados e podem ser utilizados para outras finalidades.
Hardware Utilizado	*CoPCs*	Os nós que constituem o aglomerado são computadores pessoais.
	CoW	Os nós do aglomerado são estações de trabalho, que possuem maior poder computacional que os computadores pessoais.
Configuração dos nós	*Homogêneos*	Todos os nós constituintes do aglomerado possuem mesmo sistema operacional e mesma arquitetura.
	Heterogêneos	Aglomerado pode ser composto por nós com diferentes sistemas operacionais e distintas arquiteturas.

TABELA 1. Classificações dos aglomerados segundo PITANGA

Quanto ao *hardware* utilizado, o aglomerado pode ser dividido em:

- *CoPCs, (cluster of PCs)* – Onde os nós que compõem o aglomerado são computadores pessoais;

- *CoW, (cluster of Workstation)* – Onde os nós são estações de trabalho.

E quanto a configuração dos nós, existem dois tipos principais de aglomerados:

- *Homogêneos* – Quando todos os nós do aglomerado possuem a mesma arquitetura e o mesmo sistema operacional;
- *Heterogêneos* – Quando o aglomerado pode ser composto de nós com diversas arquiteturas e sistemas operacionais diferentes.

Para construir aglomerados, deve-se realizar modificações no sistema das máquinas, com o intuito que todas interajam e se comuniquem e identifiquem as outras integrantes do aglomerado e considerem os recursos compartilhados como parte de um todo e que possam acessá-los e utilizá-los. Para isso, é necessário, além de instalar ferramentas de gerenciamento, também alterar o núcleo do sistema operacional. Este núcleo é chamado de *kernel* e é o responsável por toda a ligação das instruções com a máquina.

2.3 *Kernel*

Kernel é o núcleo do sistema operacional. É a camada de *software* que se comunica com o *hardware*, gerenciando os recursos do sistema. O *kernel* é usado para gerar uma interface, padronizando a comunicação dos processos com o *hardware* de forma concorrente e segura. O *kernel* cria, e finaliza processos, aloca e libera memória para eles, além de efetuar a comunicação com os periféricos de entrada e saída.

Segundo TANEMBAUM (1999), conforme a arquitetura dos *kernels*, pode-se separá-los em três categorias:

- *Kernel monolítico ou mono-bloco* – é uma arquitetura onde todo o núcleo é executado no espaço do *kernel* no modo de supervisão;
- *Micro-kernel* – é um termo usado para caracterizar a arquitetura cujas funcionalidades saíram do *kernel* e foram para servidores, que se comunicam com um núcleo mínimo, usando o mínimo possível o espaço do sistema e deixando o máximo de recursos rodando no espaço do usuário;
- *Kernel híbrido* – é baseado em *microkernel* no qual módulos externos podem executar operações em modo protegido, para evitar trocas

de contexto e melhorar o desempenho do sistema. No entanto, sendo híbrido, pode adicionar ou remover funcionalidades.

2.4 Beowulf

Alguns modelos de aglomerados ganharam destaque por características marcantes em seus projetos e seus resultados. Um deles é o aglomerado Beowulf [STERLING, BECKER, SALMON, SAVARESE, 1999] que ficou famoso por ser uma solução de baixo custo de paralelismo.

A necessidade de maior poder computacional e operações processadas em paralelo fizeram com que se buscassem soluções cada vez mais sofisticadas.

Devido a popularização do PC e a retração do mercado de mainframes, o preço dos PCs teve uma queda considerável e os avanços de suas tecnologias fizeram com que seu desempenho fosse cada vez maior. Essas características favoreceram o interesse nas pesquisas de aglomerado de PCs e estas alcançaram resultados interessantes. A construção de aglomerados fez com que o aproveitamento de processadores fosse beneficiado utilizando balanceamento de carga e alta disponibilidade através da replicação de recursos.

No início de 1994, Donald Becker e Thomas Sterling criaram o projeto BEOWULF, considerado um sistema de baixo custo com grande poder computacional. Trabalhavam no CESDIS (Centro de Excelência em Dados Espaciais e Ciências Informáticas) sob o patrocínio da HPCC (Comunidade de Aglomerados de Alto Desempenho) e apoiados pelo projeto ESS (*Earth and Space Sciences*) da NASA. [BEOWULF.org, 2009]

O aglomerado do tipo Beowulf surgiu com o intuito de concorrer com os altos valores dos *mainframes* sendo uma solução de desempenho com baixo custo. É constituído por máquinas com configurações normais ao padrão comercial, interligados por uma rede privada utilizando um sistema operacional baseado em Linux e ferramentas gratuitas. O aglomerado possui um computador que é utilizado como gerente do aglomerado interligado em rede com computadores que são os trabalhadores, que recebem a carga e fazem o processamento das tarefas (figura 3). Ainda hoje é um modelo adotado em várias entidades acadêmicas ou empresas que precisam de maior poder computacional sem dispor de alta quantia de dinheiro.

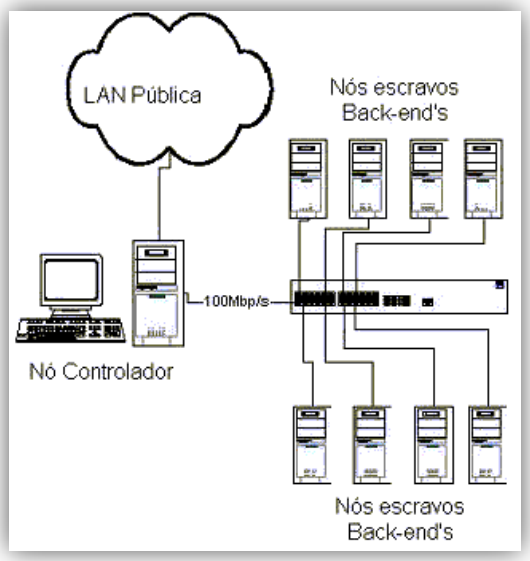


FIGURA 3. Organização de um típico aglomerado Beowulf
FONTE: Pitanga, 2003.

Muitas instituições constroem aglomerados do tipo Beowulf para pesquisa ou para processamento de dados que exigem grande poder computacional, como simulações, renderização e cálculos com matrizes ou grande quantidade de informações.

O Beowulf ganhou destaque devido a facilidade de configuração e manutenção, por se tratar de *software* livre. Além disso, por ser um sistema operacional de código aberto, pode-se adaptá-lo para corresponder as características mais relevantes ao projeto.

O modelo do Beowulf permite usar tecnologia barata de *hardware* e *software*. No entanto existem muitas ferramentas de gerenciamento de aglomerados. A nossa pesquisa focou ferramentas que sejam de fácil utilização e bastante intuitivas. Essa é a característica das ferramentas que compõem o projeto MOSIX. Este projeto visa garantir que as tarefas sejam transferidas entre as máquinas transparentemente.

2.5 MOSIX

O MOSIX (Multicomputer Operating System Unix) [BARAK, SHILOH, 2009] consiste em um sistema distribuído que provê uma transparência do sistema, também chamado de Sistema de Imagem Simples (Single-System Image – SSI) [BRAUN, 2006]. O aglomerado é constituído de vários nós. Através de troca de

mensagens entre os nós e migração preemtiva dos nós, o sistema tem uma abstração e o usuário tem a visão de ser apenas uma única máquina.

MOSIX é implementado como uma camada de *software* que permite que aplicações executem em nós remotos, como se fossem executados localmente. Os usuários podem iniciar aplicações regulares (seqüenciais e paralelas) em um nó, enquanto as ferramentas de gerenciamento MOSIX buscam automaticamente recursos e migram processos para outros nós de forma transparente. Os usuários não precisam alterar aplicações com bibliotecas, efetuar login remoto nos nós, ou mesmo copiar arquivos para nós. As migrações são supervisionadas por um conjunto abrangente de algoritmos que acompanham o estado dos recursos e procuram melhorar o desempenho global pela dinâmica na utilização dos recursos, por exemplo, balanceamento de carga.

Um comparativo entre as características do projeto Beowulf e do projeto MOSIX é demonstrado na tabela 2:

Propriedades	**Beowulf**	**MOSIX**
Sistema Operacional	Linux	Linux
Tipo	Processamento distribuído	Balanceamento carga
Servidor	Central	Distribuído
Instalação	Necessita compilação de kernel para componentes	Necessita compilação de kernel
Portabilidade	Boa	Boa
Escalabilidade	Ruim, deve-se alterar arquivos de configuração de todos os nós	Completamente transparente, basta inserir o nó no sistema
Adaptabilidade	--	Independe da arquitetura, somente da versão do MOSIX
Tarefas suportadas	Processos paralelizados em código	Qualquer tarefa é suportada, porém algumas não podem ser migradas

TABELA 2. Comparativo entre Beowulf e MOSIX
Fonte: Viva o Linux, 2007

O MOSIX foi elaborado pela equipe do professor Amnon Barak, na Universidade Hebrew de Jerusalém, Israel. O projeto fez parte da construção de um aglomerado chamado PDP11/45 [PDP11.org, 2009] feito pela Força Aérea Americana na década de 1980.

A execução do projeto ocorreu em várias etapas para diversas versões do UNIX e diversas arquiteturas. Este projeto já possui uma segunda versão (MOSIX 2) que é utilizado em aglomerados de alto desempenho e grades computacionais. A segunda versão possui recursos de descoberta dinâmica e distribuição de carga de trabalho. A primeira versão é utilizada em aglomerados comuns.

A grande vantagem do MOSIX perante aglomerados Beowulf é a característica de não ser dedicado. Enquanto o Beowulf precisa que os programas sejam projetados para serem divididos em várias *threads* para distribuí-las entre os nós, os aglomerados MOSIX atuam verificando disponibilidade de carga (processador e memória) e migrando processos dos nós mais ocupados para os nós mais ociosos, não precisando, necessariamente, que os processos sejam projetados para computação distribuída.

2.6 OpenMosix

O OpenMosix [BUYTAERT, STEVENSON, 2004] é uma ramificação do MOSIX. Este projeto foi criado porque o MOSIX estava se tornando ferramenta comercial e alguns pesquisadores acreditavam que o projeto deveria ter o código aberto. Moshe Bar esteve envolvido durante anos com o projeto MOSIX (www.mosix.com), era co-gerente de projetos e gerente geral da companhia MOSIX. Após divergência de opiniões a respeito do futuro comercial do projeto, Moshe Bar criou uma nova empresa de *clusters* (*Qlusters*, Inc.) e decidiu continuar desenvolvendo e sustentando o projeto MOSIX sob um novo nome e nova licença: openMosix com licença GPLv2.

O OpenMosix é uma extensão do núcleo do Linux para computação na forma de um Sistema de Imagem Única. Faz com que uma rede de computadores funcione como um grande e único supercomputador, através da migração preemptiva de processos e do balanceamento dinâmico de carga.

A utilização de migração preemptiva faz com que os processos migrem em qualquer momento, de forma transparente, através de algoritmos que calculam balanceamento de carga e visam prevenir a falta de memória. O sistema gerencia todos os recursos e quando determinado componente se encontra com excesso de trabalho, seus processos migram e são executados remotamente em componentes com menos trabalho, aumentando, assim, o desempenho do sistema. A figura 4 demonstra como é realizada a migração preemptiva entre os nós do aglomerado.

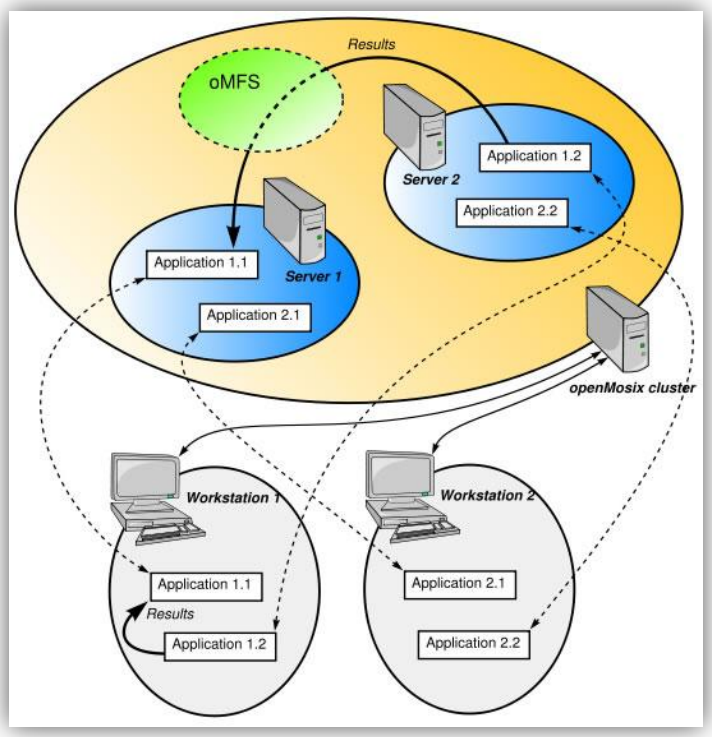


FIGURA 4. Migração de processos entre os computadores em um aglomerado com OpenMosix
FONTE: openmosix.sourceforge.net

A descentralização do controle dos nós faz com que o sistema tenha escalabilidade, permitindo que se possa adicionar ou remover máquinas a qualquer momento. Todos os nós possuem algoritmo de monitoramento de carga da CPU, velocidade de cada nó, a comunicação entre processos, memória disponível e a velocidade de acesso de cada processo.

Não é necessário saber em qual nó os processos estão sendo executados, pois os algoritmos de balanceamento se encarregam de verificar qual as condições do aglomerado. Logo que os processos são criados, eles são encaminhados para o nó mais propício de forma a otimizar o desempenho do sistema. Durante a execução dos algoritmos, as condições do sistema são analisadas. Caso seja necessário migrar um processo, o custo de migração é calculado. A migração somente ocorre se ela favorecer o desempenho do sistema.

Dentre as características do OpenMosix, pode-se destacar:

Escalabilidade: É possível adicionar ou remover nós sem a necessidade de nova configuração de todo o sistema. O nó apenas deve preencher os requisitos do sistema, que é ter o OpenMosix instalado para que seja adaptado pelo aglomerado;

Adaptabilidade: Não é necessário que haja uma homogeneidade no sistema. O aglomerado pode ser constituído por diversas máquinas com diferentes

arquiteturas cooperando em tarefas do aglomerado. Deve-se apenas ter a mesma versão do OpenMosix instalado nas máquinas;

Não necessita recompilação das aplicações: Não é necessário que as aplicações sejam especialmente projetadas e implementadas para sistemas paralelos. Qualquer aplicação pode ter benefícios no aglomerado. Uma aplicação pode ser inteiramente migrada para outro nó, caso seja necessário, sem afetar sua execução.

2.7 *Single-System Image* (SSI)

Single-System Image (SSI) [PITANGA, 2003][BRAUN, 2006] é o termo em inglês associado a Sistema de Imagem Única, ou seja, o sistema é visto como uma única unidade computacional, mesmo sendo composto por vários computadores ligados em rede, fazendo com que se tenha a ilusão de que todo o processamento esteja sendo feito em uma única máquina. O interessante deste tipo de sistema é que é mais fácil de usar e gerenciar que aglomerados especializados.

Sistemas SSI permitem que sejam gerados *checkpoints* dos estados de cada nó do aglomerado e das migrações de processos entre eles. Isso permite que se consiga manter um estado consistente e possibilita recuperação caso o sistema entre em um estado instável.

A maioria dos sistemas SSI fornece uma visão única do sistema de arquivos através de um servidor NFS, discos compartilhados ou replicação de dados. Isso garante que os processos possam ser executados em qualquer nó e o acesso aos arquivos necessários seja efetuado sem precisar de qualquer tipo de precaução.

3 PROJETO E PLANEJAMENTO DO AGLOMERADO

A idéia de construir um aglomerado no LINCE (Laboratório de Informática do Centro de Educação) surgiu da necessidade de utilização dos computadores que se encontravam ociosos no laboratório e que, apesar de antigos, ainda poderiam ser aproveitados de forma correta. Estes computadores eram utilizados como terminais BSD ligados a um servidor Debian. No entanto, devido à comunicação entre terminal e servidor pela rede, para os usuários que utilizavam os computadores para acesso a internet, o tempo de espera era visível, e os computadores eram considerados ruins. Outro fator que contribuiu para que os computadores ficassem sem uso foi a falta de pontos de rede e espaço físico no laboratório. Com isso, a forma mais viável de utilizá-los foi em um aglomerado.

Depois de tomada esta decisão, faltava decidir:

- Qual seria o sistema operacional;
- Qual organização do aglomerado a ser implementado;
- Quais ferramentas seriam utilizadas para gerenciamento do aglomerado;
- Qual a estrutura da rede a ser adotada;
- Quantas máquinas e quais as suas respectivas configurações.

O sistema operacional escolhido para testes foi o Debian GNU/Linux, pois este é o sistema utilizado nos servidores presentes no laboratório, e por ser um sistema bastante estudado e pesquisado nos projetos de Software Livre do LINCE.

3.1 Constituição física do aglomerado

A estrutura física do aglomerado foi determinada pelos componentes disponíveis no laboratório. Algumas máquinas estavam danificadas e não puderam ser utilizadas no projeto. A falta de pontos de rede fez com que fosse necessária a utilização de equipamento extra para roteamento, no caso um *switch* com capacidade para oito computadores. Existe a possibilidade de ser instalado um *switch* de trinta e duas portas no laboratório, que permitirá utilizar mais máquinas no sistema, mas enquanto isso não acontece o aglomerado é composto por um *Switch* 10/100Mbits para interligar oito computadores (figura 5): um Pentium III 866Mhz (com HD de 10,2GB e memória RAM de 256MB), um AMD K6/2 300Mhz (com HD

de 10,2GB e memória RAM de 320MB) e seis Celeron 500Mhz (com HD de 10,2GB e memória RAM de 128MB).

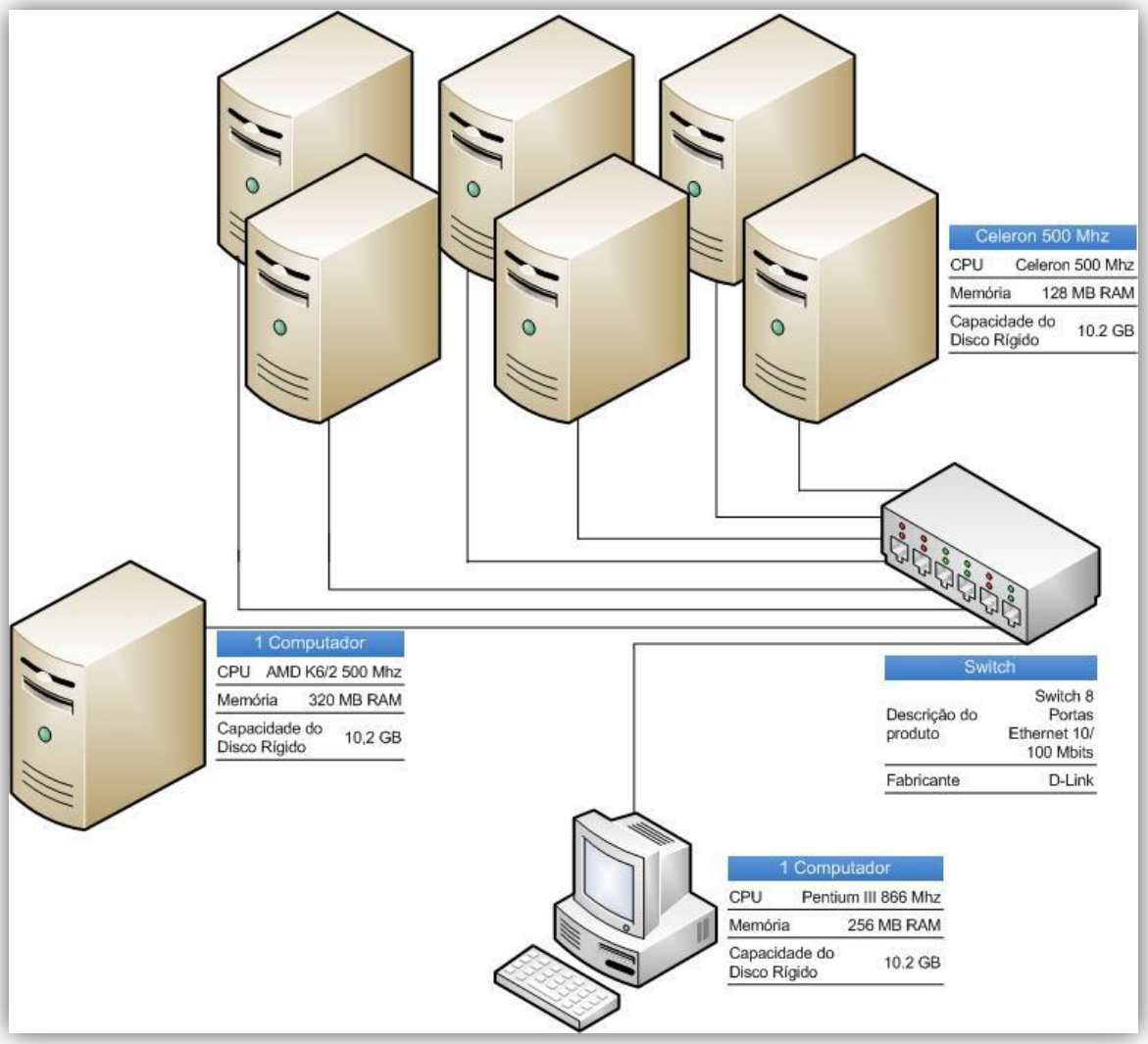


FIGURA 5. Organização física das máquinas no aglomerado.

3.2 Ferramentas testadas para gerenciamento do aglomerado

Foram pesquisados três modelos de aglomerados: Beowulf, MOSIX e OpenMosix. O primeiro tipo de aglomerado a ser testado nas máquinas foi o Beowulf, com o código fonte para o *kernel* versão 2.6.24 e os pacotes *perfcrt* e PAPI que visam controle e testes de desempenho. Também foi testado o MOSIX para o código fonte para o *kernel* versões 2.6.18, 2.6.24 e 2.6.29.2. Outra opção foi testar o OpenMosix para o *kernel* versão 2.6.29.2. Além desses modelos também foram testadas duas distribuições de GNU/Linux especialmente voltadas para construção de aglomerados: Dynebolic versão 1.4 e ClusterKNOPPIX versão 3.6, ambas

possuem os pacotes OpenMosix já inseridos no sistema e ambos são versões que inicializam pelo CD, ou seja, Live CD.

Durante as pesquisas, foram encontradas várias fontes sobre projetos de aglomerados que utilizavam Linux, como Debian, Fedora, FreeBSD, Gentoo, Red Hat, entre outros. No laboratório, o sistema mais utilizado é o Debian, por isso, a idéia inicial foi testar este sistema. A utilização deste sistema consistia em compilar seu *kernel* com as ferramentas de gerenciamento para implementação de um aglomerado.

Tanto os testes de compilação de *kernel* para construção de um *kernel* com os pacotes *perfctr* [Feldt, 2002] e *PAPI* [http://icl.cs.utk.edu/papi/, 2009] quanto a compilação de *kernel* para construção de um *kernel* com o pacote MOSIX tiveram problemas na compilação gerando erros na inicialização dos módulos ou na montagem da partição principal. Depois de várias tentativas de configuração e compilações do *kernel*, estas duas alternativas foram descartadas. Conforme surgiam novas descobertas com as outras ferramentas encontradas, como a utilização do OpenMosix e de distribuições que já possuíam o *kernel* compilado contendo os pacotes para aglomerados.

Conforme a pesquisa evoluía, mais informações eram agregadas no projeto. Análises foram feitas e constatações foram realizadas, entre as constatações estão:

- O Beowulf necessitava que os programas fossem projetados para programação distribuída, que não é o nosso caso, que teremos a execução de aplicações normalmente utilizadas em escritório ou laboratório de informática;
- O OpenMosix possui ótimas ferramentas de gerenciamento e visualização do desempenho do sistema, com interfaces gráficas e bastante intuitivas, além de possibilitar bastante controle sobre a migração de processos;
- As distribuições GNU/Linux pesquisadas agregam no seu *kernel* o pacote do OpenMosix e suas respectivas ferramentas para gerenciamento e monitoramento do sistema. Sua característica principal é a escalabilidade, pois para agregar nós no sistema não é necessário configurações avançadas.

3.3 Modelo adotado no projeto

Considerando todas as informações adquiridas, o projeto seguiu a diretriz de utilizar as distribuições com *kernel* compilado com OpenMosix. Como são distribuições Live CD, ou seja, que rodam diretamente do CD, tivemos que fazer modificações para instalar no HD das máquinas, configurar a rede e outras opções para funcionamento do aglomerado.

O modelo terá como base a distribuição *clusterKNOPPIX* instalada no HD e serão testados os resultados da distribuição *Dynebolic* rodando pelo CD. No próximo capítulo serão discutidos testes com as ferramentas pesquisadas, problemas encontrados e solução adotada e resultados obtidos.

A utilização destas distribuições visa favorecer a escalabilidade do sistema, pois não será necessária a compilação do *kernel* para cada máquina, independente de sua arquitetura e suas limitações de recursos. O modelo adotado será útil posteriormente quando for necessário adicionar novas máquinas no aglomerado. Não é necessário conhecimento avançado sobre compilação, funcionamento do *kernel*. Apenas noções de configurações de rede e instalação do sistema no disco rígido do computador.

4 DESENVOLVIMENTO E RESULTADOS

Após pesquisar três modelos de aglomerado, escolhemos testar ferramentas para o modelo Beowulf, pacotes do MOSIX e com as distribuições de GNU/Linux adaptadas para aglomerados Dynebolic e ClusterKNOPPIX que possuem OpenMosix pré-instalado no kernel.

4.1 Compilação de *kernel* para o modelo Beowulf

Após pesquisa bibliográfica sobre os diversos modelos de aglomerados, decidimos tomar como base um aglomerado Beowulf com sistema operacional Debian 4.0 Etch. No entanto, seguindo diversos tutoriais, após a compilação para adicionar os pacotes do *perfctr* e *PAPI* no kernel, nos deparávamos sempre com o mesmo erro: os módulos não eram carregados e o sistema não era inicializado, atestando o erro: *'Kernel panic - Not Syncing: Attempted to kill init!'*. Este erro pode ser gerado pela incompatibilidade do kernel com o equipamento ou por falha no momento de compilação. Devido a várias tentativas frustradas, a idéia de seguir este modelo foi deixada de lado. Passamos, então, a pesquisar novas soluções: o MOSIX e o OpenMosix.

4.2 Compilação de *kernel* com os pacotes de MOSIX

Após pesquisa sobre o MOSIX foram iniciadas as tentativas com compilação de *kernel* com os pacotes de MOSIX. Para esta compilação, foram utilizadas várias fontes de configuração e instalação do MOSIX no *kernel* do Linux que estávamos utilizando, ainda o Debian 4.0 Etch. Também efetuamos diversas tentativas, seguindo os passos para compilar o *kernel* e inserir o MOSIX no mesmo. A compilação do *kernel* sem os pacotes do MOSIX eram realizadas com êxito. No entanto, quando eram adicionados os pacotes, o resultado era erro de inicialização. O problema pode ser uma possível falha ao criar o arquivo ***initrd***, que é o arquivo que contém as informações sobre quais dispositivos serão inicializados, sem este arquivo, periféricos como adaptadores de rede ou mesmo discos rígidos não funcionarão neste kernel. O erro era o seguinte: *'Kernel panic – Not Syncing: No init found.'*.

4.3 Testes de distribuições adaptadas para aglomerado com a ferramenta de gerenciamento OpenMosix

Após testes de compilação que resultavam erros, vimos que a configuração e compilação de kernel para construção do aglomerado seriam uma barreira para a escalabilidade do sistema, pois exigiriam conhecimento específico destas ações para adicionar novas máquinas. Uma solução para este problema foi a utilização de distribuições que já eram compiladas com os pacotes e ferramentas utilizadas para configuração e gerenciamento do aglomerado. Foram encontradas duas distribuições que possuíam características interessantes para o projeto como facilidade de configuração, interface e ferramentas de fácil aprendizado e utilização, *clusterKNOPPIX* e *Dynebolic,* que já possuíam o OpenMosix agregado no sistema.

Após ser escolhido utilizar distribuições preparadas para aglomerados no projeto, e tendo todas as configurações que permitam o funcionamento do sistema, o modelo estava pronto para ser submetido a testes e simulações.

Estes testes tem como objetivo analisar a adaptabilidade e o desempenho alcançado tanto em programas paralelos (que são adaptados em código para executarem em aglomerados) quanto em programas habitualmente utilizados por usuários comuns (que são programas de edição de texto, edição e visualização de imagens, navegadores *Web*), através de séries de execuções variando o número de nós utilizados. Outro objetivo é verificar a compatibilidade com outros sistemas operacionais para facilitar a escalabilidade do sistema e comparar os resultados entre as duas distribuições escolhidas.

Inicialmente, foram efetuados testes no aglomerado para verificar a adaptabilidade. Onde foi testado utilizar computadores com sistemas operacionais distintos. Metade dos nós do aglomerado utilizaram Dynebolic rodando diretamente do CD e a outra metade com clusterKnoppix instalados no disco rígido. Os computadores se encontram na rede e migram tarefas entre eles. No entanto, deve-se tomar cuidado quanto a versão do OpenMosix utilizado. Todos os nós possuam a mesma versão para que o aglomerado funcione corretamente.

Para instalar o clusterKnoppix basta abrir o terminal e digitar o comando *knoppix-installer* como *root*. Este comando abre um menu com opções de confgurações para a instalação (figura 7). Primeiramente, formatamos o HD na opção 3, no nosso caso preparamos o disco rígido para ter 90% de partição primária e 10% de swap. Após isso, escolhemos a configuração de instalação na opção 1,

que é a padrão conforme o CD. Então podemos iniciar a instalação na opção 2 do menu.

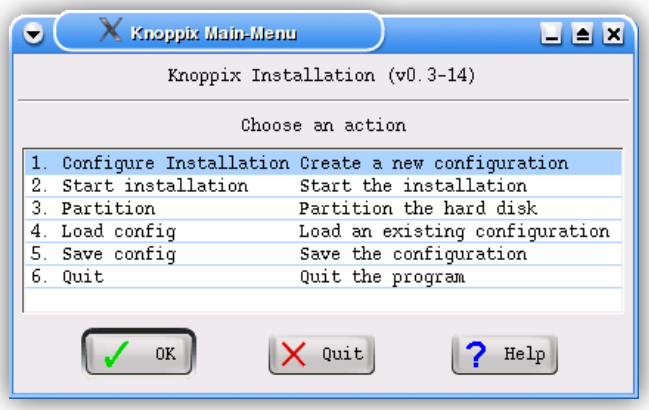


FIGURA 6. Menu de instalação do clusterKnoppix

Seguindo os passos de instalação o sistema irá concluir o processo e mostrar uma mensagem de operação finalizada com sucesso.

Após instalado o sistema clusterKnoppix em todos os nós do aglomerado, foram efetuados testes de desempenho do sistema rodando uma grande quantidade de tarefas e verificando o quão eficiente são as migrações de tarefas e se haveria balanceamento de carga entre os nós das tarefas lançadas no computador utilizado para interação com o usuário.

Não houve nenhuma alteração nas configurações do OpenMosix. Então foram rodadas várias chamadas a aplicativos utilizados nos laboratórios, como editores de texto, *browsers*, editores de imagens, etc. Com esse teste constatou-se que nenhum processo migrou para os outros nós, todos foram executados no computador onde foram criados.

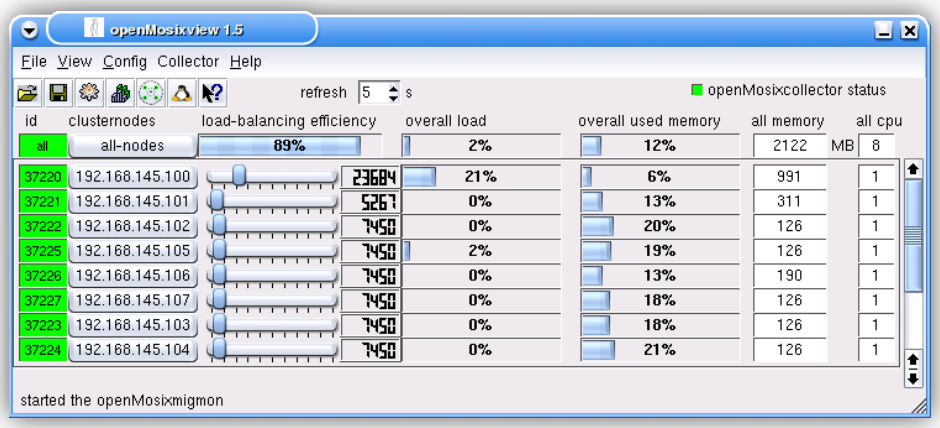


FIGURA 7. Ferramenta openMosixview mostrando valores dos nós sem alterações nas configurações

Como pode-se notar na figura 8, o nó 37220 que é o nó utilizado pelo usuário possui 21% de carga, isso mostra que não houve distribuição para os outros nós e o balanceamento de carga é 89% (esse valor ocorre pois não houve uma sobrecarga de tarefas).

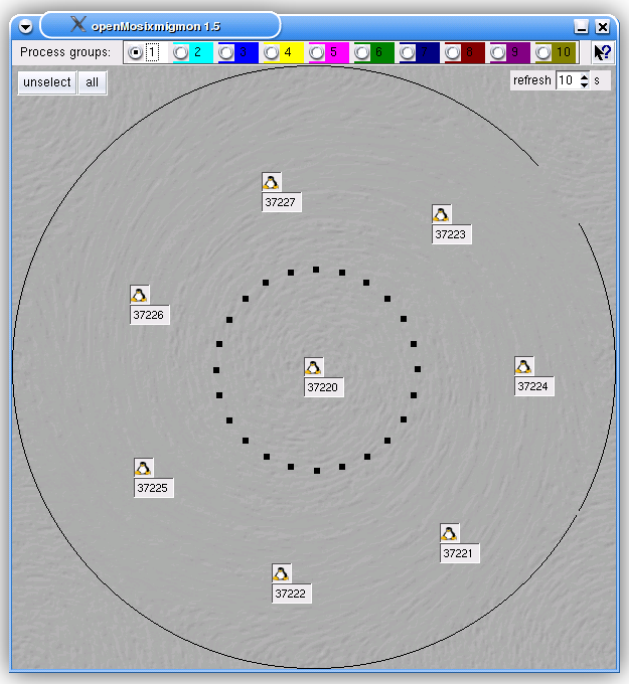


FIGURA 8. Ferramenta openMosix migmon mostrando que não houve migração de processos

A figura 9 representa a constituição do aglomerado. Cada imagem de pingüim representa um nó do aglomerado. A imagem do meio representa o computador utilizado pelo nó gerente. Cada ponto ao redor do nó gerente representa um processo criado no computador. Esta figura mostra que não houve nenhuma migração entre os processos pois os nós ainda estão próximos ao nó gerente. Isso acontece pois o OpenMosix analisa todos os recursos de cada nó e gera um valor que caracteriza o poder computacional do nó e guarda em uma variável chamada *SPEED*.

No aglomerado usado durante os testes, o computador utilizado pelo usuário tinha maior poder computacional que os outros nós. Então o algoritmo de balanceamento calculava a viabilidade de migração de processos, e como o valor era muito alto, o resultado mostrava que seria melhor executar no próprio computador ao invés de migrar. Então devemos mudar o valor desta variável para forçar a migração de processos para os outros nós com o comando *mosctl setspeed 5000.*

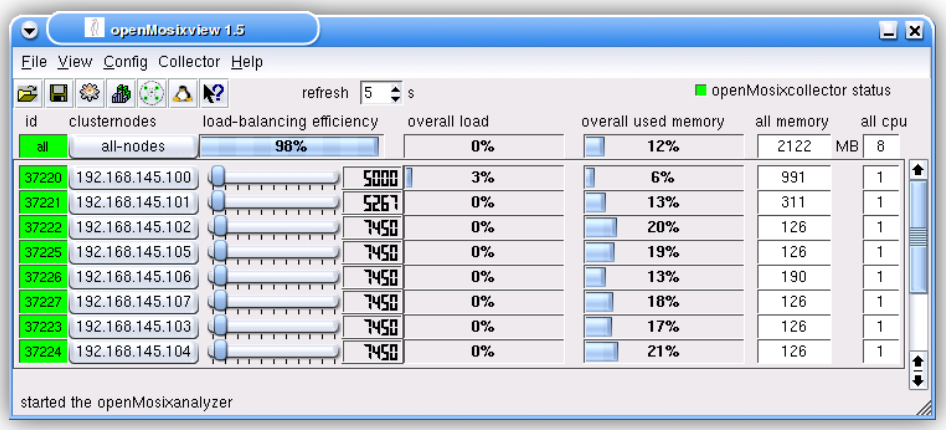


FIGURA 9. Ferramenta openMosixview mostrando valores dos nós após alteração da variável *SPEED*

Após alterar o valor da variável, fizemos novamente o mesmo teste, executar várias chamadas de aplicativos utilizados no laboratório. Como pode-se notar na figura 11, com o valor da variável *SPEED* alterado para 5000 (menor que nos outros nós), o balanceamento de carga é mais eficaz, chegando ao nível de 98% e a sobrecarga do nó gerente é apenas 3%. Desta vez tivemos resultados de migração de processos e balanceamento de carga. Como o computador acessado pelo usuário tem o menor valor da variável *SPEED*, o algoritmo chega a conclusão que a maioria de seus processos tem mais eficiência de execução se migrados para os outros nós. Esta é uma forma de induzirmos o programa gerenciador a utilizar mais as máquinas com menor poder computacional que compõem o aglomerado e liberam recursos da máquina utilizada pelo usuário, dando a impressão de menos carga sobre o sistema. A figura 12 mostra migrações de processos entre os nós. Os pontos verdes são processos que migraram do nó central para o nó onde estão. A linha pontilhada representa a migração. Os pontos pretos são os processos que não migraram e continuam no computador que foram criados. O algoritmo responsável pela migração considera o computador que é acessado pelo usuário como o pior nó devido ao valor da variável *SPEED*, então garante que muitos processos sejam enviados para outros nós.

Periodicamente, o programa gerenciador do aglomerado verifica a situação de todos os nós, e calcula qual a melhor solução de migração. Processos especiais não podem ser migrados para outros nós, entre eles as tarefas de inicialização de sistema que se comunicam diretamente com o *kernel*. Processos ociosos podem ser migrados automaticamente para os nós com menor poder computacional. Além de

migração automática, a ferramenta de gerenciamento OpenMosix migmon permite a migração de um processo para outro nó, através de solicitação do usuário em uma interface gráfica bastante intuitiva. A figura 13 mostra o resultado da migração de processos manualmente para um único nó, onde vários processos são enviados para o nó 37222.

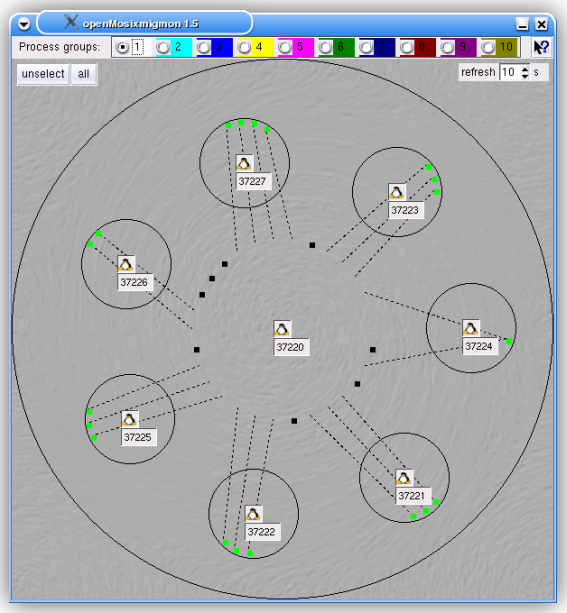


FIGURA 10. Ferramenta openMosix migmon mostrando migração automática de processos entre os nós

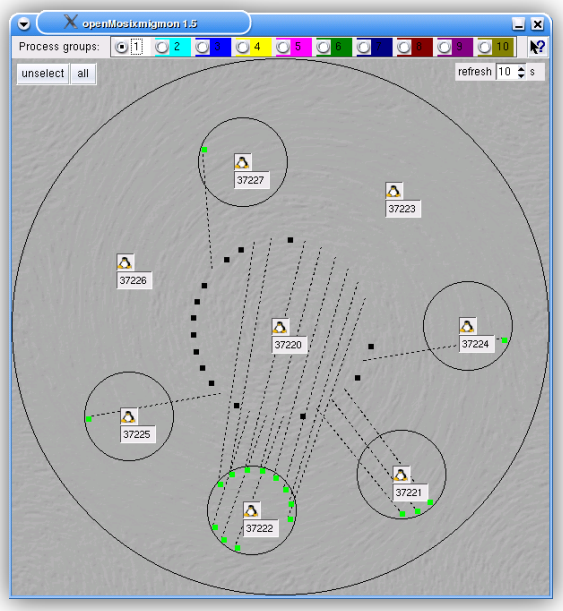


FIGURA 11. Ferramenta openMosix migmon mostrando migração manual de processos entre os nós

Outras ferramentas que auxiliam para acompanhar o estado do aglomerado são openMosix History e openMosix Analyser. A ferramenta openMosix History

(figura 14) contém informações sobre os processos. Como o histórico de migrações. A ferramenta openMosix Analyser (figura 15) contém valores estatísticos mínimo, máximo e médio de carga, balanceamento, utilização de memória e número de nós que compõem o aglomerado.

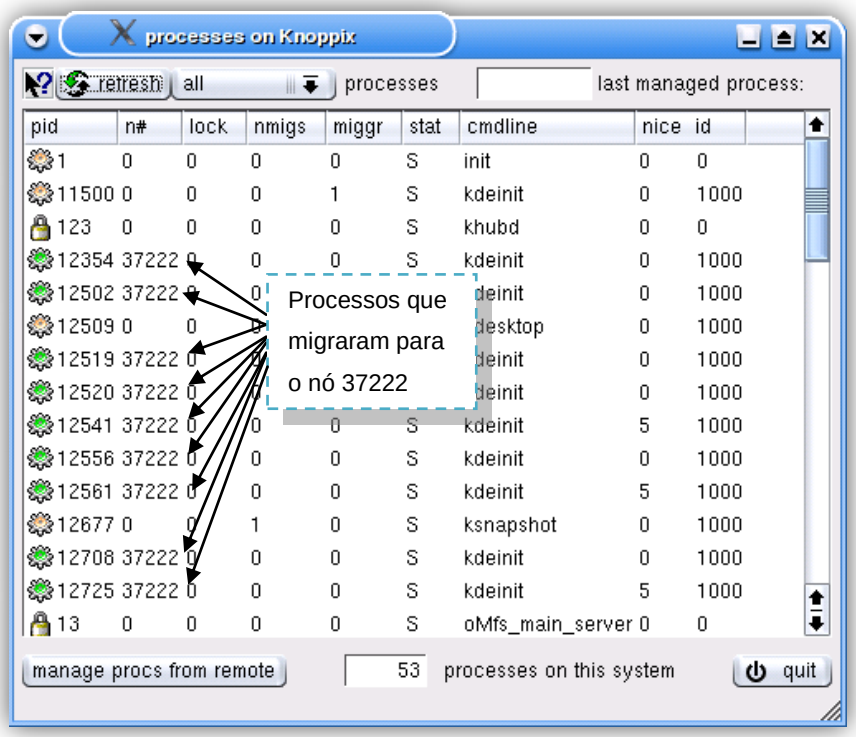


FIGURA 12. Ferramenta openMosix History com dados sobre os processos

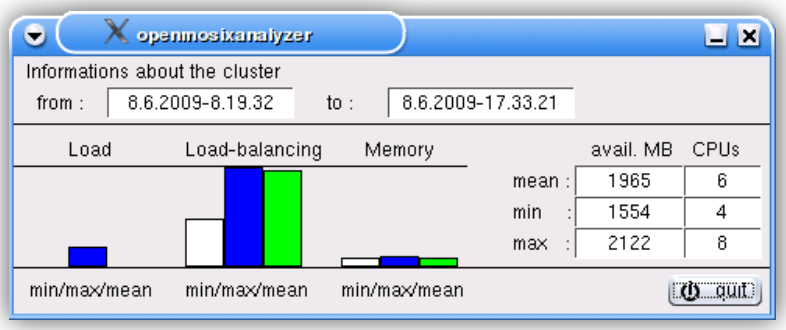


FIGURA 13. Ferramenta openMosix analyser com percentuais de dados do aglomerado

Além de migrar programas inteiramente para outros nós, pode-se também executar paralelamente. Para isso deve-se ter um código-fonte que permita a divisão de tarefas em vários processadores. É possível escolher quantas máquinas participarão da execução. A figura 16 mostra qual opção do menu deve ser escolhida para executar programas através do openMosixview. A figura 17 mostra

todas as opções de execução. Como aquela que permite execução em paralelo. Esta ferramenta permite também escolher executar com ou sem migração, executar no nó mais rápido, no mais lento, ou no nó que criou o processo.

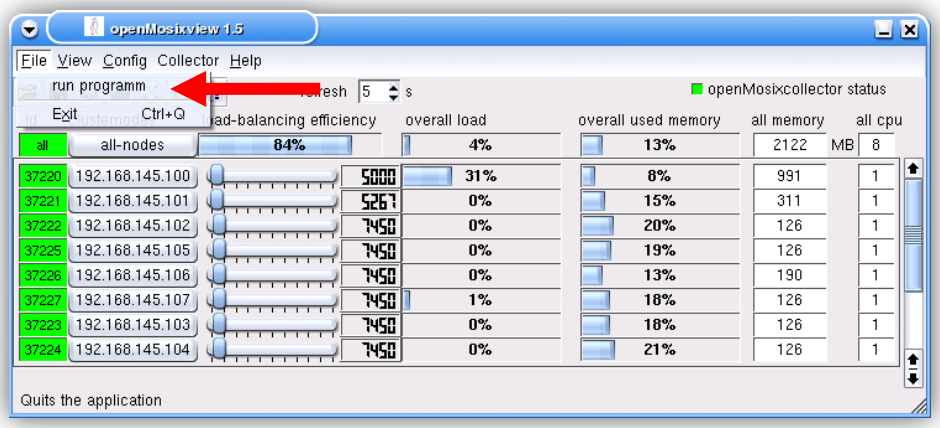


FIGURA 14. Opção do menu para rodar programas

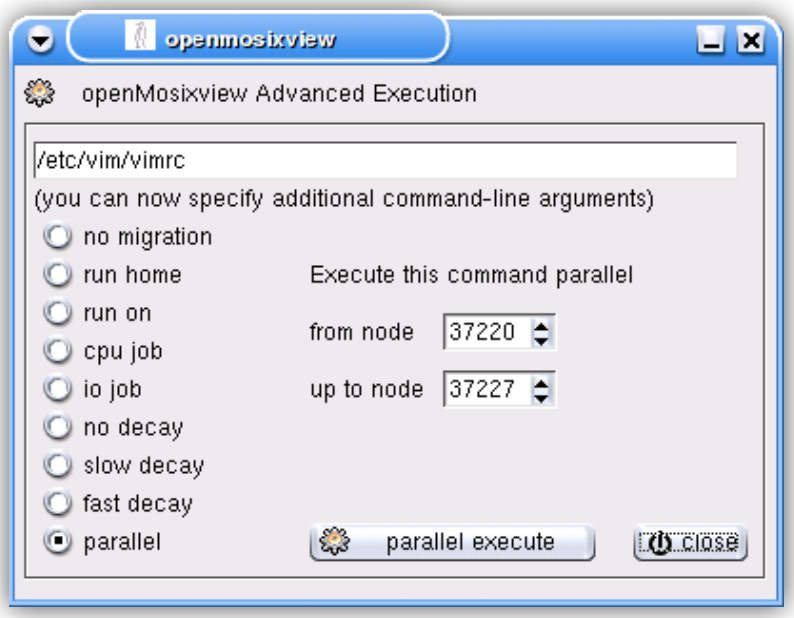


FIGURA 15. Opções para rodar um programa pelo openMosixview

O openMosix oferece ferramentas que possibilitam maior controle sobre o aglomerado. Qualquer usuário está apto a utilizar o sistema sem perceber que tarefas são distribuídas entre os nós.

Alem de testar o aglomerado rodando várias aplicações, executamos um programa dividido em vários processos para verificar qual o ganho de desempenho conforme aumenta o número de nós envolvidos na execução.

Para este teste foi utilizado um programa proposto para exercícios na disciplina de Programação Paralela que utiliza *pthreads*: o de multiplicação de

matrizes. Utilizando *threads,* ele divide as matrizes em partes e distribui o trabalho de multiplicação.

```c
int main(int argc, char **argv)
{
   int n;
   int rc;
   int taskid, ntasks;
   double start_time, end_time;

   if ((argc != 3)) {
      printf("Uso: %s <nthreads> <ordem da matriz quadrada>\n", argv[0]);
      exit(EXIT_FAILURE);
   }

   ntasks = atoi(argv[1]); // ntasks = nthreads
   n = atoi(argv[2]);      // ordem da matriz

   // Cria matrizes
   matrix_t *A = matrix_create(n, n);
   matrix_randfill(A);
   matrix_t *B = matrix_create(n, n);
   matrix_fill(B, 1.);
   matrix_t *C = matrix_create(n, n);

   // Calcula C = A * B em ntasks, medindo o tempo
   start_time = wtime();
   matrix_mult_threads(A, B, C, ntasks);
   end_time = wtime();

   // Mostra estatisticas da execucao
   printf("%d %f\n", ntasks, (end_time - start_time)/ntasks);
   fflush(stdout);

   matrix_destroy(A);
   matrix_destroy(B);
   matrix_destroy(C);

   return EXIT_SUCCESS;
```

FIGURA 16. Função *main()*

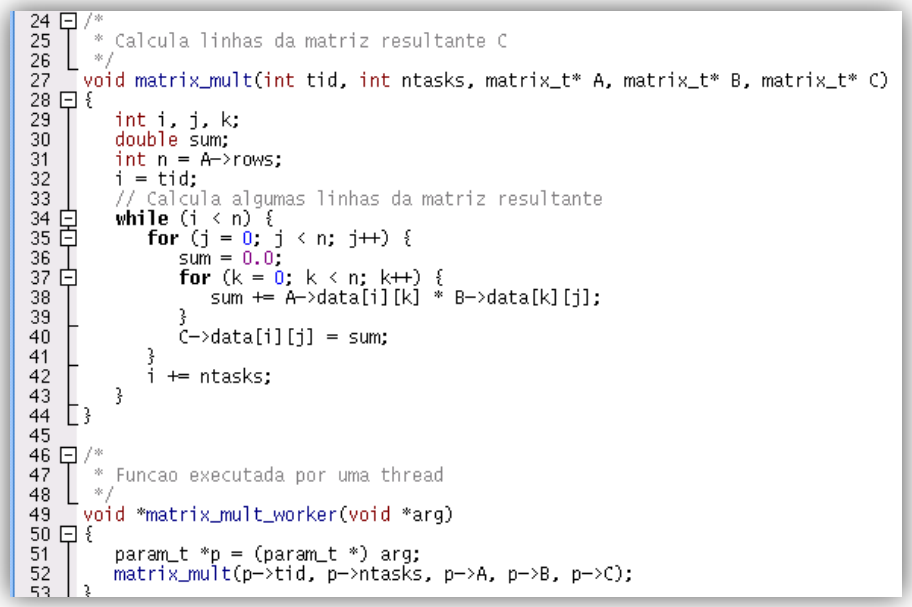
```c
/*
 * Calcula C = A * B, distribuindo o trabalho entre ntasks threads
 */
void matrix_mult_threads(matrix_t *A, matrix_t *B, matrix_t* C, int ntasks)
{
   int i;
   pthread_t *threads;
   param_t *args;

   threads = (pthread_t *) malloc(ntasks * sizeof(pthread_t));
   args = (param_t *) malloc(ntasks * sizeof(param_t));

   for (i = 0; i < ntasks; i++)
   {
      args[i].tid = i;
      args[i].ntasks = ntasks;
      args[i].A = A;
      args[i].B = B;
      args[i].C = C;
      pthread_create(&threads[i], NULL, matrix_mult_worker, (void *) (args+i));
   }
   for (i = 0; i < ntasks; i++)
   {
      pthread_join(threads[i], NULL);
   }
   free(args);
   free(threads);
}
```

FIGURA 17. Função *matrix_mult_threads()*

A figura 18 mostra a função *main()*. Esta função faz uma chamada a *matrix_mult_threads(),* mostrada na figura 19, que inicializa as *threads*, passando os argumentos das funções para a *matrix_mult_worker()*, mostrada na figura 20, sendo esta o trabalho de cada *thread*. Essa função, por sua vez, chama a multiplicação ilustrada também na figura 20 como *matrix_mult()*.



```
/*
 * Calcula linhas da matriz resultante C
 */
void matrix_mult(int tid, int ntasks, matrix_t* A, matrix_t* B, matrix_t* C)
{
   int i, j, k;
   double sum;
   int n = A->rows;
   i = tid;
   // Calcula algumas linhas da matriz resultante
   while (i < n) {
      for (j = 0; j < n; j++) {
         sum = 0.0;
         for (k = 0; k < n; k++) {
            sum += A->data[i][k] * B->data[k][j];
         }
         C->data[i][j] = sum;
      }
      i += ntasks;
   }
}

/*
 * Funcao executada por uma thread
 */
void *matrix_mult_worker(void *arg)
{
   param_t *p = (param_t *) arg;
   matrix_mult(p->tid, p->ntasks, p->A, p->B, p->C);
}
```

FIGURA 18. Funções *matrix_mult_worker()* e *matrix_mult()*

Os testes com este programa foram aplicados em três situações. Variando o número de *threads* de um a oito, para cada número foram feitos as seguintes ações. Primeiramente o programa foi executado cinqüenta vezes em um único computador antigo, depois o programa foi executado as mesmas cinqüenta vezes no aglomerado com as configurações padrões. Por último, uma seqüência de cinqüenta execuções no aglomerado com alterações nas configurações para melhoramento de desempenho.

Os dados coletados são divididos em menor tempo de execução, tempo médio de execução e maior tempo de execução para cada número de *threads,* em cada uma das três situações. Outro valor calculado foi a aceleração que o sistema obtém conforme aumenta o número de *threads*. Estes dados visam mostrar o desempenho em cada uma das situações e comparar os resultados do aglomerado com alterações das configurações e demonstrar os ganhos neste caso.

Os dados mostram que para os três casos, quanto maior o número de *threads*, menor é o tempo de execução. Os tempos de execução no aglomerado com alterações nas configurações são menores que no aglomerado com as configurações padrões. Estes são menores que os tempos de execução no computador antigo isolado.

A tabela 3 mostra os valores de menor tempo, tempo médio e maior tempo de execução em cada uma das situações testadas.

Número de *Threads*	Tempos de execução		
	Aglomerado com alterações nas configurações	Aglomerado com configurações padrão	Um único computador antigo
1	40,77667	44,17067	46,23867
2	21,254	26,41033	31,21267
3	16,926	19,20567	25,426
4	12,61967	15,24433	15,36467
5	10,738	12,35033	13,78367
6	9,121	11,40067	11,84533
7	8,372667	9,93	10,38767
8	7,482333	9,398667	9,260667

TABELA 3. Tempos de execução para cada um dos sistemas testados

A aceleração (tabela 4) é o resultado da divisão do tempo de execução com uma *thread* pelo tempo de execução com n *threads*. A aceleração demonstra o ganho de desempenho do sistema conforme o número de tarefas em que o trabalho é dividido, aumenta.

Número de *Threads*	Aceleração		
	Aglomerado com alterações nas configurações	Aglomerado com configurações padrão	Um único computador antigo
1	1	1	1
2	1,918541	1,672477	1,481407
3	2,409114	2,299877	1,818558
4	3,2312	2,897514	3,009416
5	3,797417	3,576476	3,354598
6	4,470636	3,874393	3,903534
7	4,870213	4,448204	4,451304
8	5,449726	4,699674	4,993017

TABELA 4. Aceleração ganha nos sistemas conforme aumenta o número de *Threads*

Estes testes mostram que ocorre uma diminuição no tempo de execução do programa se executado no aglomerado com alterações nas configurações. Porém este ganho não é constante: quanto mais máquinas são adicionadas na computação, menor é o ganho. A aceleração no aglomerado com alterações nas configurações é maior que nos outros dois casos. Este aspecto é explicado devido às condições de rede, comunicação entre nós, leitura e escrita, mas também pode ser explicado porque parte do código é serial, e não pode ser paralelizado. Isso acarreta em ter um tempo que é constante e um tempo que varia conforme o número de processadores.

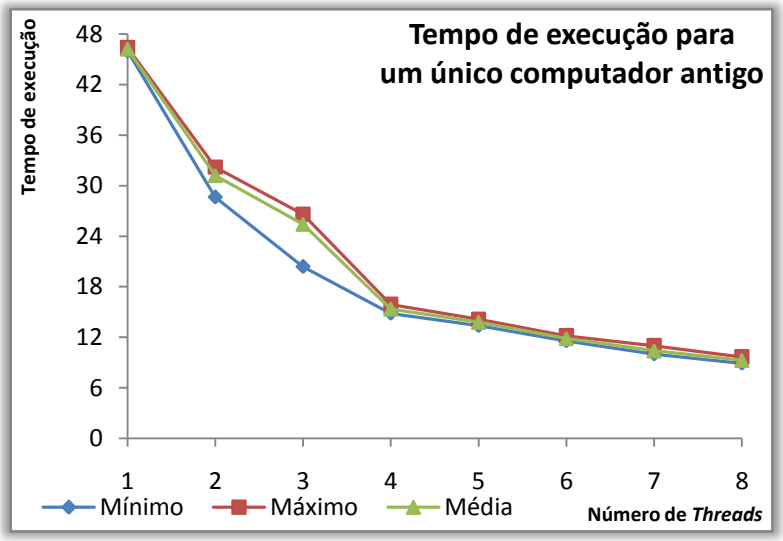


FIGURA 19. Tempos de execução em um único computador antigo

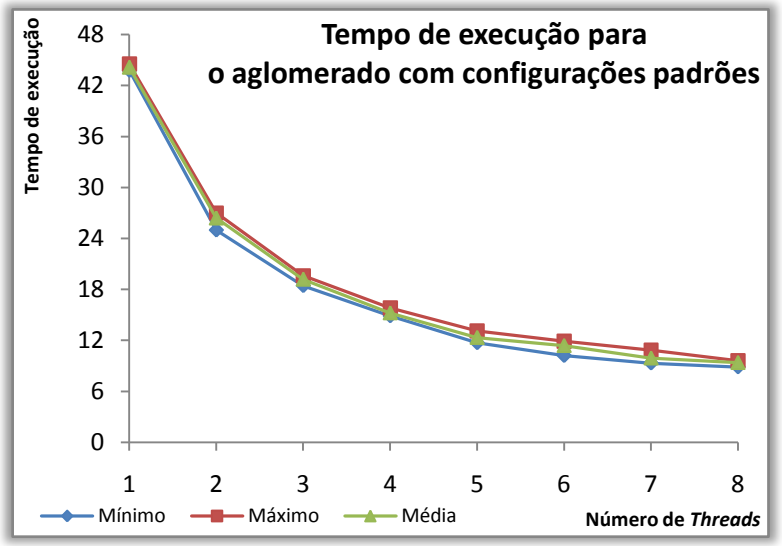


FIGURA 20. Tempos de execução no aglomerado com configurações normais

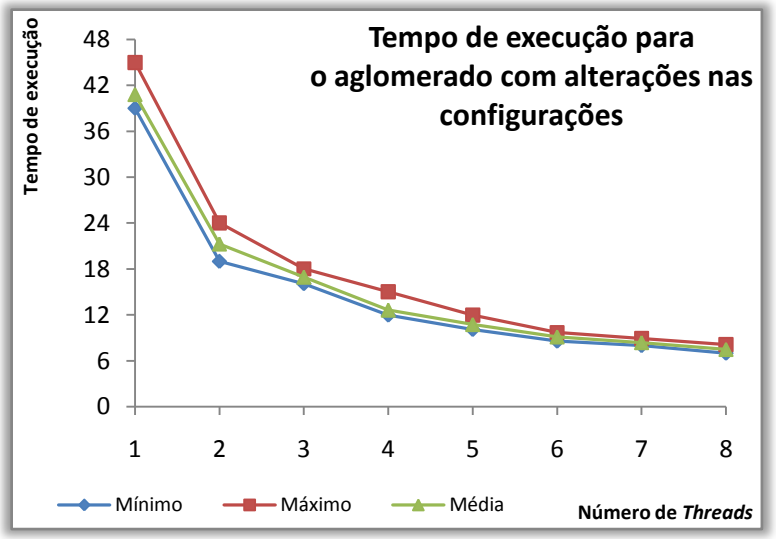


FIGURA 21. Tempos de execução no aglomerado com alterações nas configurações

Um comparativo entre os tempos médios de execução das três situações testadas é representado na figura 22 e demonstra como o aglomerado com alterações nas configurações obtém melhor desempenho.

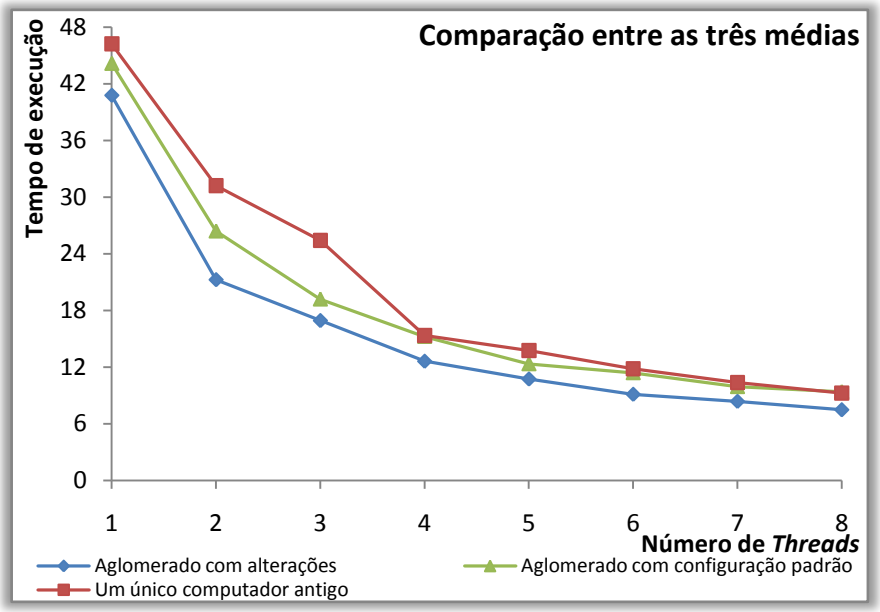


FIGURA 22. Comparativo entre o tempo médio de execução dos três sistemas testados

A figura 23 representa a aceleração para os três sistemas, levando em consideração os tempos médios de execução. A figura mostra que o aglomerado que tem alterações nas configurações para ganho de desempenho possui maior aceleração.

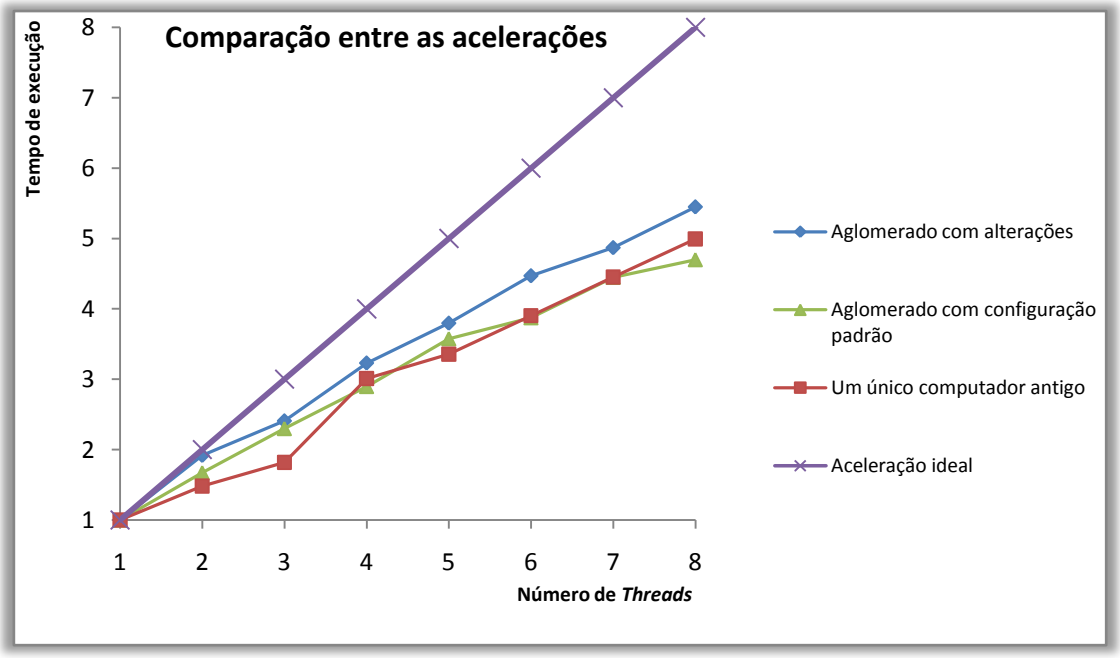


FIGURA 23. Comparação entre as acelerações dos três sistemas

A aceleração ideal mostrada na figura anterior é referente ao tempo esperado para um caso ótimo, em que dividindo as tarefas em n *threads*, o tempo de execução também será dividido por n. Então a aceleração é linearmente crescente.

4.4 Aplicação instalada no aglomerado

O aglomerado desempenha, por enquanto, papel de computador de pesquisa, onde serão feitos os projetos do laboratório. Um projeto em execução, que poderá ser hospedado no aglomerado, é o novo sistema de agendamento de auditórios e equipamentos para professores e funcionários do Centro de Educação. Este sistema utiliza Apache, PHP, MySQL e é uma programa Web, ou seja, é acessado via *browser*.

Outra função que o aglomerado pode desempenhar é um servidor *Web* para hospedar o site do LINCE, que atualmente está hospedado nos servidores da UFSM. Ou, então, poderá hospedar um servidor para ambientes de educação a distância como, por exemplo, o Moodle, para ser utilizado em cursos e aulas realizados no laboratório.

O aglomerado pode ser utilizado por qualquer usuário. E a manutenção não exige grande conhecimento técnico, apenas configurações de rede de um sistema Linux e por ser composto de máquinas velhas, o aglomerado está sujeito a falhas nos nós. Para solucionar o problema é necessário reinstalar o sistema no computador que ocorreu a falha. Os funcionários do laboratório já possuem conhecimento sobre esta área.

5 CONCLUSÕES

5.1 Observações finais

Este trabalho de graduação apresentou uma solução de utilização de máquinas antigas, de forma a se obter desempenho, através de ferramentas de gerenciamento de balanceamento de carga e migração preemptiva, com interfaces gráficas e intuitivas, que possibilitam fácil controle da situação dos processos dentro do sistema.

Com esta proposta procuramos uma forma de facilitar a utilização de aglomerados, buscando ferramentas de fácil instalação e configuração. Assim, não é necessário grande conhecimento técnico sobre o assunto para utilizar e gerenciar o sistema. Para isso foi demonstrado um caso de construção de aglomerado e as alterações que foram feitas sobre este para que possa ser configurado devidamente. É possível, também, sem muita dificuldade, aumentar o número de máquinas do aglomerado.

Ao concluir o trabalho, os objetivos propostos foram satisfatoriamente atingidos, pois foi possível construir o aglomerado e os resultados de testes sobre o sistema mostraram que é possível utilizar máquinas antigas conectadas para aplicações que isoladamente seriam executadas com um desempenho menor.

5.2 Trabalhos Futuros

A proposta apresentada serve como base para pesquisas futuras que possam evoluir cada vez mais o desenvolvimento deste tema, possibilitando que a implementação de um aglomerado seja cada vez mais simples e abrangente. A seguir apresentam-se aperfeiçoamentos que podem ser feitos em áreas abordadas por este trabalho:

1. **Utilização de outros sistemas operacionais:** Pode-se melhorar a adaptabilidade do aglomerado buscando soluções em outros sistemas operacionais, tendo em vista que a única limitação é a versão do OpenMosix. Solucionando o problema de compilação de *kernel* é possível que o aglomerado seja constituído de computadores com sistemas operacionais utilizados no laboratório, como por exemplo Debian e Ubuntu.

2. **Testes de desempenho por escalabilidade:** É muito importante verificar o quanto é aceitável aumentar o número de nós no aglomerado. Como o nosso projeto foi limitado em oito máquinas, não podemos saber, exatamente, até quantas máquinas estamos ganhando desempenho. Pois, a partir de um determinado número, o aglomerado tem um desempenho limitado.

3. **Testes de desempenho por aplicações:** Também é de suma importância saber para quais aplicações o aglomerado tem maior eficiência, podendo, assim, construir aglomerados para finalidades definidas de forma que se obtenha maior desempenho.

4. **Definir hierarquia no aglomerado:** Para conseguir um sistema distribuído mais seguro e mais estável, podemos dividir o aglomerado em níveis hierárquicos, através de atribuições de funções. Pode-se determinar quais nós irão tratar a comunicação com usuário e construir replicação destes recursos para que haja alta disponibilidade. O aglomerado pode ser dividido em níveis, onde cada nível será responsável por uma função. O primeiro nível seria responsável por receber os pedidos de usuários, sendo que os nós deste nível devem ser replicados para que exista alta disponibilidade de serviços. O segundo nível é o nível responsável pela execução de cálculos e processamento de dados. O terceiro nível consiste em nós que são responsáveis pelo armazenamento de dados, nestes nós estarão instalados os bancos de dados do sistema. Também é necessária a replicação de recursos no terceiro nível, para que o serviço esteja disponível o maior tempo possível. Assim o aglomerado se comporta de forma que mantenha o serviço organizado e com tratamento para possíveis falhas que possam suspender o trabalho. Além disso, evita-se que um nó seja o gargalho de desempenho de todo o sistema.

REFERÊNCIAS BIBLIOGRÁFICAS

ALUVINO, José Carlos. **Alto poder de computação com cluster de computadores usando MOSIX.** 2008, Trabalho de Graduação (Curso Superior de Tecnologia em Informática com Ênfase em Redes de Computadores) – Faculdade de Tecnologia de Guaratinguetá, Guaratinguetá, 2008.

AMAR, Lior; BARAK, Amnon; MAOZ, Tal; MEIRI, Ehud; SHILOH, Amnon. **MOSIX2 Tutorial.** 2009. Disponível em: <www.mosix.org/pub/tutorial.pdf>. Acesso em: Junho de 2009.

ÁVILA, Rafael Bohrer. **Uma proposta de Distribuição do Servidor de Arquivos em Cluster.** 2005, Tese (Doutorado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

BARAK, Amnon. **Overview of MOSIX2 for Clusters and Multi-Clusters.** 2009. Disponível em:<www.mosix.org/pub/overview.pdf>. Acesso em: Junho de 2009.

BARAK, Amnon; SHILOH, Amnon. **The MOSIX2 Management System for Linux Clusters and Multi-Clusters – A White Paper.** 2009.
Disponível em:<www.mosix.org/pub/MOSIX2_wp.pdf>. Acessado em: Junho de 2009.

BARBOSA, André da Silva; HASHISUCA, Antônio Marcos; SILVA, Filipe Augusto. **Estudo, projeto e implantação de um *Cluster* de computadores utilizando ferramentas livres.** In: XXI Congresso de Iniciação Científica e Tecnológica em Engenharia, 2006.

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

BRAUN, Leandro Rodrigo. **Avaliação da implementação de clusters de computadores usando tecnologias livres.** 2006, Trabalho de Graduação (Ciência da Computação) – Centro Universitário Feevale, Novo Hamburgo, 2006.

BUYTAERT, Kris; STEVENSON, Scot W. **The OpenMosix HowTO.** 2004. Disponível em: <tldp.org/HOWTO/pdf/openMosix-HOWTO.pdf>. Acessado em: Junho de 2009.

COULOURIS, George; DOLLIMORE, Jean; KINDBERG, Tim. **Sistemas Distribuídos: conceito e projeto**. Porto Alegre: Bookman, 2007.

IBM.com. **What is a JES? – HASP definitions.** 2009.
Disponível em:
<http://publib.boulder.ibm.com/infocenter/zos/v1r9/index.jsp?topic=/com.ibm.zos.r9.hasa800/has2a86046.htm>. Acesso em: Junho de 2009.

FELDT, Robert. **PerfCtr - Ruby Performace Counters Interface.** 2002. Disponível em: <http://aspectr.sourceforge.net/perfctr/> Acesso em: Junho de 2009.

MATOS, Edval P. de. **Técnicas para construção de clusters de alto desempenho com baixo custo.** 2006, Trabalho de Graduação (Ciência da Computação) – Universidade São Francisco, Itatiba, 2006.

PAPI - Performance Application Programming Interface. 2009. Disponível em: < http://icl.cs.utk.edu/papi/> Acesso em: Junho de 2009.

PEREIRA, Tiago Silva. **Um Ambiente de *Middleware* para Melhoria de Desempenho para Sistemas Distribuídos e Paralelos.** 2005, Trabalho de Graduação (Ciência da Computação) – Universidade Federal de Santa Catarina, 2005.

PITANGA, Marcos. **Computação em Cluster: O Estado da Arte da Computação.** Rio de Janeiro: Brasport. 2003.

PITANGA, Marcos. **Construindo supercomputadores com linux**. 2ª edição. Rio de Janeiro: Brasport. 2004.

ROCHA, Johnny M. G. ***Cluster* Beowulf: aspectos de projeto e implementação.** 2003, Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Pará, Belém, 2003.

ROSA, Bruno O. T. **Análise de sistemas de comunicação para computação paralela em clusters.** 2002, Dissertação (Mestrado em Ciências "Física Aplicada – Opção Física Computacional") – Universidade de São Paulo, São Carlos, 2002.

ROSENTIEL, Wolfgang. **Job Entry System – JES.** 2009, Universität Tübingen. Disponível em:
< http://www-ti.informatik.uni-tuebingen.de/os390/os390/subsyst/job02.pdf>.
Acesso em: Junho de 2009.

SCHNORR, Lucas Mello. **Visualização simultânea e multi-nível de informações de monitoramento de *Cluster.*** 2005, Dissertação (Mestrado em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

TANENBAUM, Andrew. **Sistemas operacionais modernos**. Rio de Janeiro: LTC. 1999.

TOP500. **Top500 supercomputer site.** Disponível em: <http://www.top500.org>. Acesso em: Junho de 2009.

VIÉGAS JÚNIOR, Raimundo; YOKOKURA, Alex Yuichi. **Estudo de Viabilidade da implementação de Técnicas de Cluster para a criação de Laboratórios de Informática em Instituições de Ensino Públicas voltada à programação nos cursos de Engenharia e Ciência da Computação** In: World Congress on Computer Science, Engineering and Technology Education, march 2006.

STERLING, Thomas; BECKER, Donald J.; SALMON, John; SAVARESE, Daniel F. **How to Build a Beowulf - A Guide to the Implementation and Application of PC Clusters.** MIT Press. 1999.