



Fundação Educacional do Município de Assis
IMESA - Instituto Municipal de Ensino Superior de Assis

FERNANDO HENRIQUE DOS SANTOS

**REVISÃO BIBLIOGRÁFICA SOBRE ARQUITETURAS PARA O
PROCESSAMENTO PARALELO E COMPUTAÇÃO EM CLUSTER**

Assis

2013

Av. Getúlio Vargas, 1200 – Vila Nova Santana – Assis – SP – 19807-634

Fone/Fax: (0XX18) 3302 1055 homepage: www.fema.edu.br

FERNANDO HENRIQUE DOS SANTOS

**Revisão Bibliográfica sobre Arquiteturas para o Processamento
paralelo e Computação em Cluster**

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis, como
requisito do Curso de Graduação.

Orientador: **Me. Douglas Sanches da Cunha**

Área de Concentração: _____

Assis

2013

FICHA CATALOGRÁFICA

SANTOS, Fernando

Revisão Bibliográfica sobre Arquiteturas para o Processamento paralelo e Computação em Cluster/ Fernando Henrique dos Santos. Fundação Educacional do Município de Assis – FEMA – Assis, 2013.

58p.

Orientador: Douglas Sanches da Cunha

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA.

1.Arquitetura 2.Processamento 3.Cluster

CCID: 001.6

Biblioteca da FEMA

Revisão Bibliográfica sobre Arquiteturas para o Processamento paralelo e Computação em Cluster

FERNANDO HENRIQUE DOS SANTOS

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado, pela seguinte comissão examinadora:

Orientador: **Me. Douglas Sanches da Cunha**

Analisador: **Esp. Célio Desiró**

Assis

2013

RESUMO

Com a evolução da tecnologia, a popularização do conceito em nuvens, as áreas de pesquisas aumentando a necessidade de um maior processamento, cresce a necessidade da programação paralela e surge fortemente o conceito do *cluster*, um aglomerado de computadores organizados e configurados de forma como se fosse um único computador com vários processadores. O objetivo deste trabalho é fazer o levantamento bibliográfico sobre a arquitetura da computação em *cluster*, mostrando as principais bibliotecas de paralelismo, os tipos de redes, o sistema *Beowulf*, mostrando a importância de sua estrutura e do estudo que deve ser feito para tomar as melhores decisões, conseguindo obter um sistema ideal para suprir a necessidade. Suas características e as configurações necessárias para implementar um *cluster* utilizando o sistema operacional *FreeBSD*.

Palavras-chave: Arquitetura. Processamento. Cluster

ABSTRACT

With the evolution of technology, the popularization of the concept in clouds, areas of research increasing the need for further processing, the need of parallel programming and strongly emerges the concept of cluster, a cluster of computers arranged and configured so as were a single computer with multiple processors. The object of this work is to lift the literature about the architecture of cluster computing, showing the main libraries of parallelism, types of networks, the Beowulf system, showing the importance of this structure and the study should be done to make the best decisions, obtaining the ideal system to meet the need. Its features and functions necessary to implement a cluster using the FreeBSD operating system settings.

Keywords: Architecture. Processing. Cluster.

Lista de Figuras

Figura 2.1 - Cluster de Processamento.....	15
Figura 2.2 - Ilustração do conceito de Von Neumann	17
Figura 2.3 - Ciclo de Von Neumann	18
Figura 2.4 – Arquitetura MIMD Fonte	19
Figura 2.5 – Cluster da NASA	21
Figura 2.6 - Funcionamento PVM	23
Figura 3.1 – Funções MPI	25
Figura 3.2 – Comando para compilar C++ utilizando MPI	26
Figura 3.3 – Comandos para iniciar o LAM	26
Figura 3.4 – Código de Exemplo utilizando MPI	27
Figura 3.5 – Ilustração da biblioteca PVM	28
Figura 3.6 – Código Exemplo utilizando PVM	30
Figura 4.1 - Tela de menu do FreeBSD	33
Figura 4.2 – Exemplo do arquivo rc.conf	34
Figura 5.1 Cluster Beowulf	38
Figura 5.2 Topologia em estrela	39
Figura 5.3 Topologia em malha	39
Figura 5.4 Topologia Fat Tree	40
Figura 5.5 Interface bWatch	41
Figura 6.1 – Exemplo de configuração para instalação	43
Figura 6.2 – Comandos de instalação do DHCP-SERVER	44
Figura 6.3 – Exemplo do arquivo de configuração do DHCP	44
Figura 6.4 – Arquivo de configuração /etc/rc.conf	46

Figura 6.5 – Arquivo <code>/etc/hosts:allow</code>	46
Figura 6.6 – Arquivo <code>/etc/inetd.conf</code>	46
Figura 6.7 – Comando para copiar o arquivo <code>/boot/pxeboot</code>	46
Figura 6.8 – Arquivos Exportados	47
Figura 6.9 – Arquivo <code>/etc/exports</code>	47
Figura 6.10 – Arquivo <code>/etc/rc.conf</code>	47
Figura 6.11 – Comandos para adicionar no arquivo <code>DISKLESS</code>	48
Figura 6.12 – Exemplo de arquivo de configuração do kernel	48
Figura 6.13 – Comandos para compilar o kernel	51
Figura 6.14 – Comando para copiar os arquivos do <code>/cluster/etc/</code>	51
Figura 6.15 – Exemplo do <code>fstab</code>	51
Figura 6.16 Exemplo do arquivo <code>/cluster/conf/default/etc/rc.conf</code>	52
Figura 6.17 – Comando para replicar os usuários	52
Figura 6.18 – Comandos para instalar o <code>MPICH</code>	53
Figura 6.19 – Comando para adicionar a variável <code>PATH</code>	53
Figura 6.20 – Arquivo <code>/cluster/conf/default/etc/inetd.conf</code>	54
Figura 6.21 – Exemplo do arquivo <code>/cluster/conf/default/etc/hosts.equiv</code>	54
Figura 6.22 – Exemplo do arquivo <code>/etc/hosts</code>	55
Figura 6.23 – Arquivo <code>/etc/csh.cshrc</code>	55
Figura 6.24 – Comandos de configuração da <code>PVM</code>	56

SUMÁRIO

1.INTRODUÇÃO	11
1.1. OBJETIVOS.....	12
1.2. JUSTIFICATIVAS	12
1.3. MOTIVAÇÃO	13
1.4. PERSPECTIVA DE CONTRIBUIÇÃO	13
1.5. METODOLOGIA DE PESQUISA.....	14
2. CLUSTER	15
2.1 ARQUITETURA	15
2.2 TIPOS DE CLUSTERS	19
2.3 USO DO CLUSTER	20
2.4 FERRAMENTAS.....	22
3. BIBLIOTECAS MPI E PVM.....	24
3.1 BIBLIOTECA MPI	24
3.2 BIBLIOTECA PVM.....	27
4. SISTEMA OPERACIONAL	32
4.1 SISTEMA OPERACIONAL FREEBSD	32
5.BEOWULF	36
5.1.ARQUITETURA	36
5.2.TIPOS DE REDE.....	36
5.3.MONITORAMENTO.....	41
6. IMPLEMENTAÇÃO.....	42
6.1 INSTALAÇÃO DO FREEBSD	42

6.2 CONFIGURAÇÃO DO NÓ MASTER	43
6.3 CONFIGURAÇÃO DOS NÓS ESCRAVOS	47
7. CONCLUSÃO	57

1. INTRODUÇÃO

Em 1994 um projeto da CESDIS subsidiária da NASA consistia em colocar 16 computadores ligados pela rede ethernet e organizados de forma que trabalhassem como se fossem apenas um supercomputador. Esse projeto foi um sucesso, logo ganhou destaque na NASA, nas universidades e outras empresas, (NETO 2008).

A idéia é pegar vários computadores e interligá-los fazendo um aglomerado de computadores e foi esse o nome escolhido em inglês, *cluster* (aglomerado). Esse *cluster* da NASA foi chamado de *cluster Beowulf*.

Segundo Pitanga, afirmou em 2003 que o *cluster Beowulf* é de alto desempenho, tem uma maquina mestre que divide um grande problema em vários pequenos problemas e esses pequenos problemas são passados para os nós escravos que processam e devolvem ao nó mestre, dessa forma o resultado final do processamento é muito mais rápido.

Existem outros tipos, como de alta disponibilidade, a prioridade é a disponibilidade do link, são configurados mais de um mestre, ou até mesmo mais de um *cluster*, configurados de forma que se o principal “cair” o outro nó mestre assume sem que o serviço fique indisponível.

Também tem o cluster de balanceamento, neste caso o cluster faz o controle de carga do processamento, fazendo com que tenha um equilíbrio na distribuição dos processos. Exige um constante monitoramento na comunicação e seus mecanismos de redundância, para evitar erros e falha do serviço.

Este trabalho fará uma pesquisa sobre como funciona o *cluster beowulf* utilizando o S.O.(sistema operacional) *freeBSD* que é um dos melhores S.O., para este tipo de serviço, os sistemas *netBSD* e *Red Hat*, também estão entre os mais recomendados.

Com esse grande crescimento da tecnologia, da internet, cresce também a

necessidade de processamento, como por exemplo, um site de busca precisa ser rápido na busca no banco de dados ou renderizar uma imagem dos relevos de uma determinada região. O cluster vem para resolver ou pelo menos diminuir esse problema, sendo mais eficiente e mais barato, pois não precisa de supercomputadores para se montar um cluster.

Porém não são todos os programas que executam no *cluster*, o *cluster* segue o conceito de paralelismo, ou seja, o programa tem que ser escrito de forma paralela para que possa ser executado.

1.1. OBJETIVOS

A crescente necessidade de processamento, exigindo processadores cada vez mais rápidos para atender a aplicações que tem uma elevada taxa de computação, a construção de um super computador com centenas ou até milhares de processadores, tem um custo muito alto, por isso, este trabalho tem por objetivo desenvolver uma pesquisa sobre a arquitetura do *cluster*, adquirindo novos conhecimentos desse aglomerado de computadores que pode ter um alto nível de processamento a um custo bem mais baixo do que uma super máquina com vários processadores.

1.2. JUSTIFICATIVAS

A grande necessidade de processamento, como renderizar uma imagem para um filme, *Titanic*, *Shrek*, *Final Fantasy*, filmes que utilizaram cluster para renderizar os efeitos especiais, ou até mesmo para renderizar o filme inteiro, como é o caso das animações. (MORIMOTO 2005).

O autor Marcos Pitanga faz recomendações em seu livro em 2008, ele recomenda utilizar cluster para buscas no banco de dados que contém um numero muito grande de informações como é o caso do *Google*. Simulações e cálculos realizados por

grandes empresas, de petróleo, de estudo espacial, na medicina, na química, em várias áreas de pesquisa que exigem grandes cálculos.

Podemos pegar um grande problema e transformá-los em pequenos problemas, por exemplo, a multiplicação de matriz de 1000 por 1000, ao invés de um computador realizar todo o cálculo, pegamos 10 computadores e dividimos a matriz em 10 matrizes de 100 por 1000, dessa forma conseguimos obter os resultados muito mais rápidos.

1.3. MOTIVAÇÃO

Cluster é um conceito que ganhou forças e destaque recentemente, por isso ainda este muita pesquisa para serem realizadas, porém já é muito forte esse conceito dentro das universidades e de grandes empresas.

Novas ferramentas estão surgindo para melhorar a programação, monitoramento, configuração e até mesmo novos tipos de cluster.

Um conceito forte que estamos ouvindo falar muito nesses últimos anos é a computação em nuvem e o cluster cresce junto com este conceito, já que o processamento e a disponibilidade ficam por conta da internet, o usuário só precisa de um equipamento que conecta na internet e os programas, até mesmo o S.O., ficando por conta da empresa que tem o serviço de nuvem.

Existem universidades fazendo pesquisas com videogames, esses estudos apontam que o videogame é mais estável e consomem menos energia. Como se pode observar, essa é uma área que ainda tem muito campo de pesquisa e que vem crescendo significativamente.

1.4. PERSPECTIVA DE CONTRIBUIÇÃO

O objetivo é mostrar um pouco do poder da computação distribuída, do poder do paralelismo e entender como o cluster pode ser útil.

Apesar de parecer complicado, não é difícil construir um cluster, pode ser feito com computadores antigos, sem a necessidade de HD(*hard disk*) nos nós escravos.

Divulgar o trabalho em outras universidades, para afins de novas pesquisas, levando até elas os conhecimentos obtidos nas pesquisas e também para empresas que necessitem de grande poder de processamento, para estarem aderindo a essa tecnologia.

1.5. METODOLOGIA DE PESQUISA

Para este trabalho estamos utilizando pesquisa através de livros, artigos, pesquisas feitas na internet, anotações feita em sala de aula e entrevistas com pessoas de outras universidades.

A estrutura do trabalho está dividido da seguinte forma, capítulo 2, será tratado o assunto teórico sobre a arquitetura do *cluster*. Capítulo 3, apresenta as 2 duas principais bibliotecas de paralelismo, que são MPI e PVM. Capítulo 4, é apresentado o sistema operacional *FreeBSD*. No capítulo 6 é apresentado o sistema de *cluster* *Beowulf*, sua arquitetura, os tipos de rede e o monitoramento. O capítulo 6 apresenta as configurações necessárias para se construir um *cluster* utilizando o sistema operacional *FreeBSD*. O capítulo 7 é conclusão da pesquisa realizada.

2. CLUSTER

Neste capítulo será apresentado toda teoria e arquitetura para a construção de um *cluster*, os tipos, sua utilização e algumas ferramentas.

Cluster é um aglomerado de computadores, organizados e configurados como se fossem um só, os processos chegam ao servidor e o mesmo distribui para os nós. Dessa forma é possível conseguir um alto poder de processamento, como disse Bruno Fins em 2011.

A imagem 1 mostra um cluster:



Figura 2.1 - Cluster de Processamento Fonte
(<http://www.vivaolinux.com.br/artigo/Funcionamento-de-um-cluster-Linux/>)

2.1 ARQUITETURA

De acordo com a arquitetura de Von Neumann, os computadores digitais precisam de uma ULA(unidade lógica e aritmética), uma UC(unidade de controle) e uma memória.

A ULA é responsável por todos os cálculos que serão realizados pela máquina, é composto por milhares de registradores organizados de forma a realizar os cálculos e devolver o resultado.

A memória armazena os programas que estão em execução e também auxilia a ULA guardando valores de variáveis. Quando o programa é finalizado, ele é retirado da memória liberando espaço, se o computador for desligado todos os programas são retirados da memória.

A interação entre a ULA e a memória precisa de um gerenciamento, esse gerenciamento é feito pela UC, é a UC que é responsável por decidir se a informação vai para memória ou para ULA. A imagem 2 ilustra esse conceito de Von Neumann.

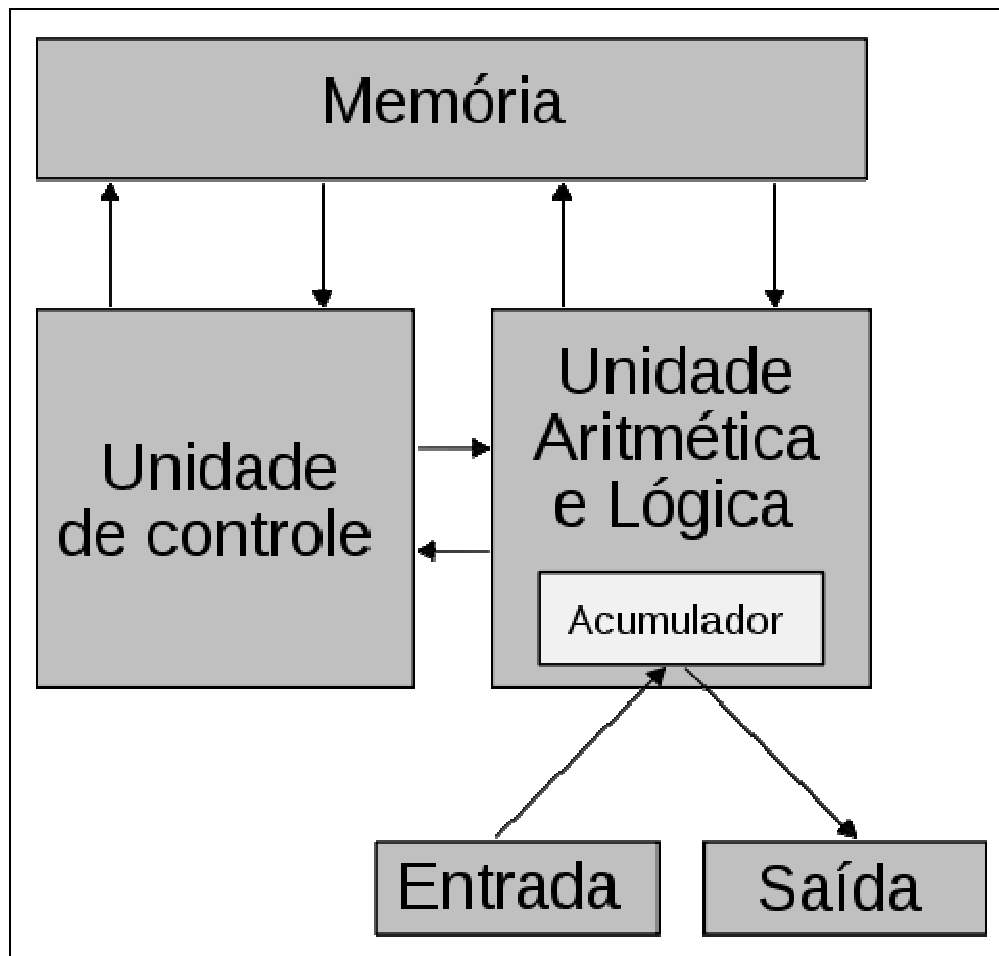


Figura 2.2 - Ilustração do conceito de Von Neumann

Fonte

(http://pt.wikipedia.org/wiki/Ficheiro:Arquitetura_de_von_Neumann.svg)

Com essa arquitetura Neumann desenvolveu um conceito que usado até os dias de hoje que é o de buscar instrução, decodificar e executar. É a entrada, execução e saída da instrução. Segue a imagem 2.1 com o ciclo de Von Neumann:

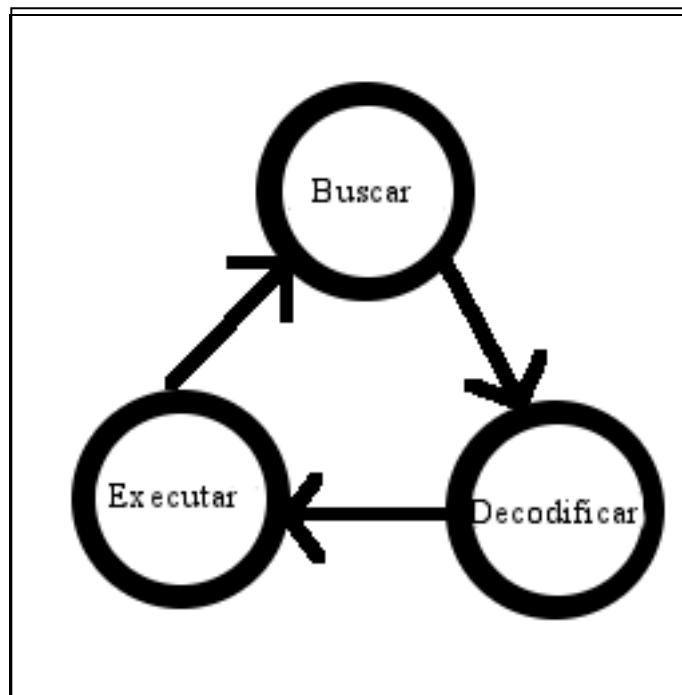


Figura 2.3 - Ciclo de Von Neumann Fonte
(http://pt.wikipedia.org/wiki/Ficheiro:Von_Neumann_Cyclus.png)

De acordo com esse ciclo, os computadores precisam buscar, decodificar e executar a instrução. Com esse conceito foi possível desenvolver o cluster, de forma que o servidor busca uma instrução, interpreta e passa para o nó fazer a execução.

O nó mestre(servidor) recebe a instrução, decodifica e distribui entre os nós, os nós do cluster precisam apenas de processador, memória e dispositivos de entrada/saída, para fazer a execução e devolver o resultado para o nó mestre. Um grande processo pode ser dividido em vários pequenos processos e divididos entre os nós escravos do cluster, tendo assim um resultado final mais rápido, como já disse o autor Pitanga em 2008.

Essa arquitetura de cada nó ter uma memória local e não compartilharem da mesma, é chamada de arquitetura MIMD(*Multiple Instruction, Multiple Data*) de memória distribuída. Existem computadores com 2 ou mais processadores que compartilham de uma mesma memória, porém são mais caros. Na imagem 2.2 podemos ver uma ilustração de memória distribuída.

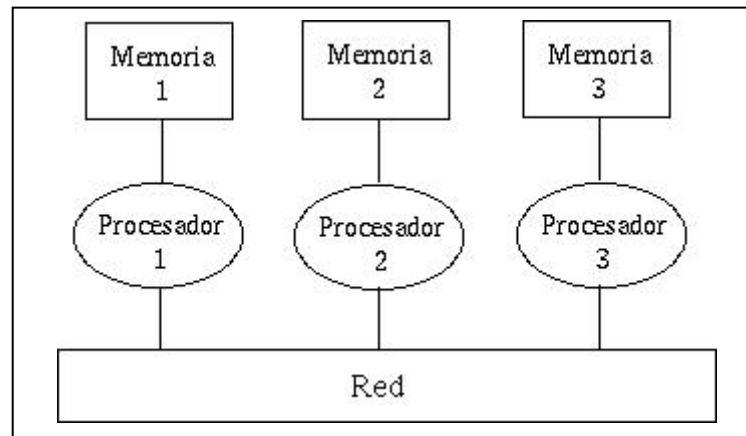


Figura 2.4 – Arquitetura MIMD Fonte
<http://telematica.cicese.mx/computo/super/cicese2000/paralelo/Part3.html>

Neste modelo Pitanga, cita 2(duas) desvantagens em seu livro que são:

- A programação é mais complicada e o paralelismo não é tão intuitivo. Isto porque a tarefa tem que enviar e receber mensagens de outros processadores de forma explícita.
- Se exigir muita troca de mensagens, o desempenho acaba sendo comprometido, pois existe um alto custo nessa comunicação.

O cluster pode ter mais de um nó mestre, este tipo de *cluster* é utilizado quando se precisa de uma alta disponibilidade, pois se um nó mestre parar o outro assume e o serviço não fica indisponível.

2.2 TIPOS DE CLUSTERS

De acordo com Fins (2001)

As tecnologias de *Clustering* possibilitam a solução de diversos problemas que envolvem grande volume de processamento. As aplicações que um cluster pode ter são diversas, indo desde a simples melhora no desempenho de um determinado sistema ou a hospedagem de um site, até o processo de pesquisas científicas complexas.

Cluster de Alto Desempenho: utilizando um sistema operacional gratuito, ele suporta uma grande carga de processamento, atingindo um alto volume de *gigaflops* (um *gigaflop* equivale a 1 bilhão de instruções de ponto flutuante executadas por segundo) em computadores comuns.

Cluster de Alta Disponibilidade: São clusters configurados com mais de 1 (um) nó mestre, se um nó mestre apresentar algum defeito, o outro assume, sendo assim, esses clusters podem garantir até 99,99% de disponibilidade.

Cluster de Balanceamento de Carga: Este cluster faz a distribuição de carga equilibrada, sendo assim, requer um monitoramento constante na comunicação entre os nós, pois a falha de um pode ocorrer interrupção do sistema.

Cluster Combo: Além de ter uma alta disponibilidade, ele faz também um balanceamento de carga. Os clusters podem ter combinações de características para atender uma necessidade.

2.3 USO DO CLUSTER

A utilização do cluster é para áreas que exigem um grande cálculo, um grande poder de processamento. Na astronomia, por exemplo, segundo Alecrim (2013), fala sobre o cluster que a NASA (*National Aeronautics and Space*) batizou de “*beowulf*”, “na verdade, este nome faz referência a um padrão de *clustering* disponibilizado pela NASA (*National Aeronautics and Space*) em 1994 e amplamente adotado desde então”.

Segue a imagem 2.3 um dos clusters da NASA:



Figura 2.5 – Cluster da NASA Fonte
(<http://pt.wikipedia.org/wiki/Ficheiro:Us-nasa-columbia.jpg>)

Segundo o site <http://pt.wikipedia.org/wiki/Cluster> esse cluster da NASA é constituído por 10.240 processadores e é chamado de 20 Altix Clusters.

O Google, entre outros grandes sites de pesquisas também utilizam *clusters* para garantir uma alta disponibilidade dos sites no ar e também para realizar as pesquisas mais rapidamente.

Com o grande crescimento da *cloud computing* (computação em nuvem) e virtualização, também cresce a utilização de clusters. “É este aspecto que pode se tornar ainda mais atraente quando a idéia de cluster é associada a conceitos mais recentes, como *cloud computing* e virtualização” Alecrim (2013).

Para renderizar imagens, animações, filmes, também é utilizado cluster, no filme Titanic foi utilizado cluster com 1200 nós para renderizar as imagens do navio afundando (Pitanga, 2008).

2.4 FERRAMENTAS

O S.O. (Sistema Operacional) é muito importante para a construção de um cluster. Segundo Pintanga (2008, p.54) “Entre os sistemas operacionais mais adequados para *cluster* de computadores estão às versões do *UNIX: Linux, FreeBSD* ou *NetBSD*”.

Para a programação paralela existem algumas bibliotecas, uma delas é a PVM (*Parallel Virtual Machine*) que foi produzida pelo *Heterogeneous Network Project* em conjunto da *Oak Ridge National Laboratory, University of Tennessee, Emory University* e *Carnegie*, em 1989, como já dizia Pintanga (2008).

A biblioteca PVM tem um processo chamado *pvmd3*, que é executado em todas as máquinas do *cluster*, formando uma máquina virtual paralela. Tem também as bibliotecas de rotinas (*libpvm3.a, libfpvm3.a* e *libgpvm3.a*), através dessa biblioteca é possível codificar, enviar e receber mensagens, criar e encerrar processos e sincronização de tarefas.

A imagem 2.4 a seguir ilustra o funcionamento da PVM:

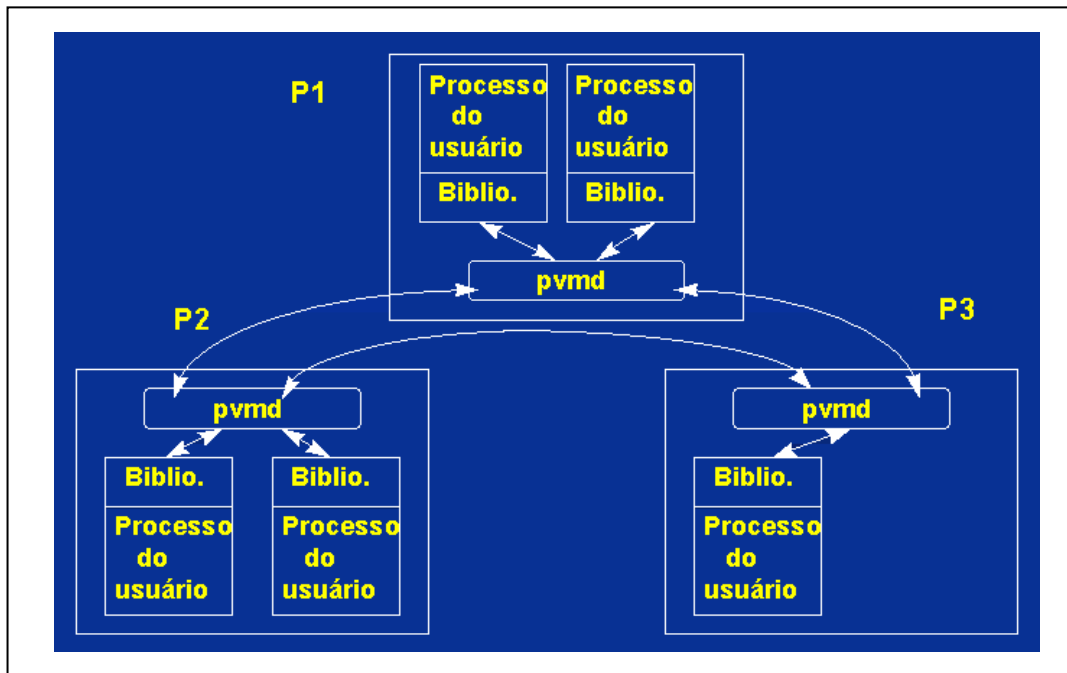


Figura 2.6 - Funcionamento PVM Fonte
 (<http://www.inf.puc-rio.br/~noemi/victal/pvm.html>)

O MPI (*Message Passing Interface*) é outra biblioteca de troca de mensagens. “O principal objetivo do MPI é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas que utilizem troca de mensagens” Pintanga (2008, p. 171).

Podem-se utilizar programas em FORTRAN, C, C++, entre outras linguagens. O fabricante de computador é responsável desenvolver uma biblioteca do seu equipamento para o ambiente paralelo.

Neste capítulo foi apresentada a teoria sobre arquitetura, tipos e algumas ferramentas sobre cluster, paralelismo e também sobre seu uso.

Pode-se concluir que na maioria dos casos em que se necessita de um grande poder de processamento, alta disponibilidade ou um sistema estável, pode-se utilizar cluster. Não se recomenda a utilização quando existe a necessidade de uma grande carga de comunicação entre os nós.

3. BIBLIOTECAS MPI E PVM

Neste capítulo será apresentado as principais bibliotecas de paralelismo, que são MPI e PVM. Com essas bibliotecas é possível estabelecer conexão entre o nó mestre com os nós escravos, garantindo uma sincronia para obter os resultados.

3.1 BIBLIOTECA MPI

Aproximadamente 40 instituições, 60 pessoas contribuíram para o projeto do MPI, a maioria dos EUA e Europa. Ferreira escreveu em 2002, “O início do processo de padronização aconteceu no seminário sobre Padronização para Troca de Mensagens em ambiente de memória distribuída, realizado pelo *Center for Research on Parallel Computing*, em abril de 1992”. Esse seminário serviu para definir as ferramentas básicas e o grupo para dar continuidade à padronização.

Em novembro de 1992 foi adotado o procedimento e a organização do HPF (*the High Performance Fortran Forum*). Neste mesmo ano o projeto foi apresentado na conferência *Supercomputing 93*, no qual deu origem a versão oficial do MPI em 5 de maio de 1994, (FERREIRA, 2002).

O MPI é voltado para os *clusters*, pois o *cluster* não faz uso do compartilhamento de memória, o Kunigami escreveu em 2010, “Já o MPI é voltado para memória distribuída, como é o caso de clusters, que são basicamente um conjunto de máquinas rodando um sistema operacional distribuído”.

Pitanga escreveu em 2008, “As funcionalidades que o MPI designa são baseadas em praticas comuns de paralelismo e outras bibliotecas de passagem de mensagens similares, como *Express*, *NX/2*, *Vertex*, *Parmacs* e *P4*”.

Segundo Pitanga, 2008, o MPI implementa um mecanismo de portabilidade e independência de plataforma, um código escrito para uma arquitetura e S.O.

específico, pode ser portado para uma arquitetura e S.O. diferentes com quase nenhuma alteração no código-fonte da aplicação.

“A paralelização de código consiste em quebrá-lo em pedaços que podem ser executados independentemente e distribuir para vários processadores”. (KUNIGAMI, 2010)

Mesmo sendo complexo, o MPI pode resolver muitos problemas com apenas 6 funções, que servem para: iniciar, terminar, executar e identificar processos, enviando e recebendo mensagens. Como ilustra a imagem 3.3 abaixo:

MPI_INIT	Inicia uma execução MPI
MPI_FINALIZE	Finaliza a execução
MPI_COMM_SIZE	Determina o número de processos
MPI_COMM_RANK	Determina a identificação de processos
MPI_SEND	Envia a mensagens
MPI_RECV	Receber mensagens

Figura 3.1 – Funções MPI Fonte
 (http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382002000100007)

Ferreira, 2002, “Os algoritmos paralelos, freqüentemente, precisam de operações coordenadas envolvendo múltiplos processos. Por exemplo, todos os processos podem precisar transpor uma matriz distribuída ou somar um conjunto de números distribuídos em cada um dos processos.”

O MPI fornece algumas funções prontas, que são:

- Barreira (*Barrier*): sincroniza todos os processos.
- Difusão (*Broadcast*): envia dado de um processo a todos os demais processos.
- Juntar (*Gather*): junta dados de todos os processos em um único processador.

- Espalhar (*Scatters*): distribui um conjunto de dados de um processo para todos os processos.
- Operação de redução (*Reduction operations*): soma, multiplicação, etc., de dados distribuídos.

O programas escritos utilizando as função do MPI, segue alguns passos e de forma genérica, os principais passos são(Ferreira, 2002):

- 1)Utilizar compiladores que reconheçam automaticamente os procedimentos MPI, ou incluir manualmente as bibliotecas MPI no processo de compilação;
- 2) Verificar quais são os nós disponíveis no ambiente paralelo;
- 3) Fixar os parâmetros para o ambiente paralelo: por exemplo, escolher os nós a serem usados e o protocolo de comunicação;
- 4) Executar o programa (que deverá ser executado em cada um dos nós escolhidos).

Kunigami já explicou em 2010 como compilar uma aplicação. Para compilar um programa C++ utilizando MPI no *linux*, segue a figura com o comando:

```
mpic++ programa.cpp -o programa
```

Figura 3.2 – Comando para compilar C++ utilizando MPI

Em seguida, é preciso editar um arquivo de texto chamado '*hostfile*' contendo o endereço dos nós do cluster (no caso de uma máquina, basta colocar *localhost*). Após isso é preciso inicializar o LAM (*Local Area Multicomputer*). Segue a figura 3.3 com os comandos para inicializar o LAM

```
lamboot -v hostfile
```

```
mpirun -np <numero de threads> ./programa <argumentos do programa>
```

Figura 3.3 – Comandos para iniciar o LAM

O programa iniciará com o número especificado de threads. Como queremos paralelizar o código, esse número pode ser a quantidade de processadores disponíveis.

Segue um código de exemplo apresentado pelo Pitanga em 2008:

```
#include "mpi.h"
#include <stdio.h>

Int main(int argc, char *argv[]){
Int meu_id, numero_processos;

// rotinas de inicialização

MPI_Init(&argc, &argv);

MPI_Comm_size(MPI_COMM_WORLD, &numero_processos);

MPI_Comm_rank(MPI_COMM_WORLD, &meu_id);

fprintf(stdout, "Olá mundo! Sou o processo %i de %i criados", meu_id,
meu_numero_processos);

// Rotinas de finalização

MPI_Finalize();

return 0;

}
```

Figura 3.4 – Código de Exemplo utilizando MPI

3.2 BIBLIOTECA PVM

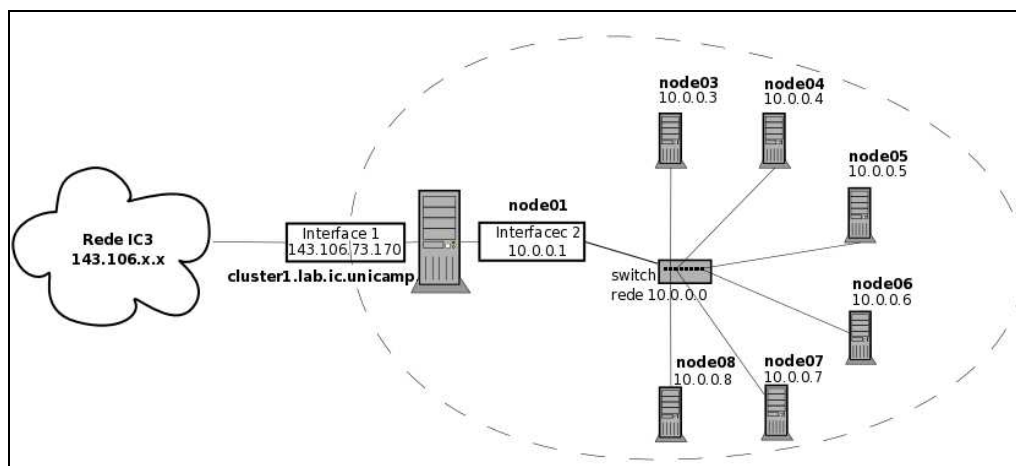
Como Pessoa, 2003, escreveu, "Originalmente criado no final da década de 80 o PVM tem uma filosofia bastante interessante, pois visa principalmente o funcionamento em ambientes heterogêneos, afinal, ela foi criada pelo *Heterogeneous Network Project*".

Segundo Pina, 1998, o projeto PVM teve início em 1989 com a construção de seu protótipo no *Oak Ridge National Laboratory*, esse protótipo foi usado apenas internamente. Em 1991 foi desenvolvida a versão 2.0 pela Universidade *Tennessee*, depois alterada para 2.1 até a 2.4. Em 1993 foi concluída a versão 3.3 que foi distribuída livremente e utilizada em vários projetos pelo mundo.

“O PVM usa o modelo de passagem de mensagens como plataforma para a utilização da computação distribuída numa vasta gama de computadores, incluindo MPP(Processamento Paralelo Massivo), que do ponto de vista do utilizador são apenas uma enorme máquina virtual.”(Pina, 1998).

“Quando a transmissão for feita entre arquiteturas diferentes, é feita uma conversão de dados pelo formato XDR(External Data Representation) automaticamente conforme RFC 1832”(Pitanga, 2008).

Segue imagem da estrutura de um cluster com a biblioteca PVM no servidor(node 1):



**Figura 3.5 – Ilustração da biblioteca PVM Fonte
(<http://www.students.ic.unicamp.br/cluster.shtml>)**

“Em PVM todas as tarefas são identificadas por um número inteiro TID, único em toda a máquina virtual, que é criado automaticamente pelo pvmd local sempre que uma nova tarefa é gerada” (Pina, 1998).

Ao paradigma geral para o desenvolvimento de uma aplicação em PVM correspondem os seguintes passos (Pina, 1998):

- escrita de programas: em C, C++ ou Fortran que contém chamadas às bibliotecas do PVM, correspondendo cada programa a uma tarefa da aplicação;
- compilação: dos programas para cada arquitetura e respectiva instalação em localização acessível.
- execução: das aplicações iniciando (manualmente) uma cópia da tarefa inicial, mestre, em qualquer um dos computadores que constituem a máquina virtual.
- geração de novas tarefas: iniciadas pela tarefas mestre que eventualmente realizam cálculos e transferem mensagens entre si.

Segundo Pitanga, 2008, a principal idéia por trás do PVM é uma estrutura heterogênea interconectados como um recurso virtualmente único. Sendo que cada computador dessa rede pode ser utilizado como um nó. Utilizando as bibliotecas de rotinas em C e Fortran, podendo escolher o protocolo que precisa utilizar TCP ou UDP, para realizar a troca de mensagens. O empacotamento e desempacotamento das mensagens geram uma sobrecarga extra, porém tem uma maior facilidade, comodidade de resolver o problema de uma arquitetura heterogênea na “grande máquina virtual”.

Vejamos algumas características do PVM citado por Pitanga, 2008:

- Quanto à interoperabilidade: além da portabilidade, os programas podem executar em arquiteturas completamente diferentes;
- Abstração completa: o PVM permite que a rede seja totalmente heterogênea, administrada como uma única máquina virtual;
- Controle de processos: capacidade de iniciar, interromper, e controlar processos em tempo de execução;
- Controle de recursos: totalmente abstrato, graças ao conceito de máquina virtual paralela;
- Tolerância a falhas: comporta esquemas básicos de notificação de falhas para alguns casos. Porém permite flexibilidade, de forma que, ainda em certas

situações onde não existe resposta de um outro nó, uma aplicação recebe os resultados das outras;

- Fácil de instalar e usar;
- *Software* de domínio publico;
- Grande aceitação e utilização no mundo.

A seguir um código de exemplo de uma aplicação em PVM:

```
#include "pvm3.h"

void main()
{
    Int nt, tid, msgtag =1;
    char buf[100];
    printf("Hello world!\n");
    nt = pvm_spawn ("hello_cnt", NULL, PvmTaskDefault, NULL, 1, &tid);
    pvm_recv (tid, msgtag);
    pvm_upkstr (buf);
    printf("%s da TID=%x\n", buf, tid);
    pvm_exit();
}

#include "pvm3.h"

void main()
{
    Int p_tid, msgtag=1;
    Char buf[100] = "Hello, PVM! da ";
    p_tid = pvm_parent ();
    strcat (buf, getenv("HOST")); /* nome da maquina */
    pvm_initsend (PvmDataDefault);
```

```
pvm_pkstr (buf);  
pvm_send (p_tid, msgtag);  
pvm_exit ();  
}
```

Figura 3.6 – Código Exemplo utilizando PVM

Neste capítulo foi apresentado as principais bibliotecas de paralelismo que são utilizadas para construção do *cluster*.

Neste capítulo conclui-se que a biblioteca MPI tem mais recursos, visa a *performance*, porém recomenda-se utilizar em arquitetura homogênea.

A biblioteca utilizando um sistema de maquina virtual, visa uma arquitetura heterogênea, podendo construir *cluster* com diferentes tipos de hardware.

4. SISTEMA OPERACIONAL

Neste capítulo será apresentado a história, a arquitetura e funcionamento do sistema operacional *FreeBSD*, que é considerado como um dos melhores sistemas para se construir um *cluster*.

4.1 SISTEMA OPERACIONAL FREEBSD

O projeto do sistema operacional *FreeBSD* teve seu início pelos 3 coordenadores: Nate Williams, Rod Grimes e Jordan Hubbard, o projeto teve como base o sistema 386BSD, como disse Jordan Hubbard em 1995.

O Hubbard disse em 1995, “Nosso objetivo original era produzir um *snapshot* intermediário do 386BSD, de forma a poder corrigir uma série de problemas com este sistema, que o mecanismo de manutenção não era capaz de resolver”. Assim nascendo o projeto *FreeBSD*, o nome foi sugerido por David Greenman.

A primeira distribuição do *FreeBSD* foi em Dezembro de 1993, foi distribuído em CDROM e via internet em parceria com a empresa *Walnut Creek CDROM*, que produziu o *CDROM* e disponibilizou uma máquina dedicada e uma conexão rápida para disponibilizar o sistema pela internet.

“O *FreeBSD* não é o sistema somente *Intel/AMD* que foi um dia. Também é executado em máquinas SPARC64 e tem um histórico relativamente longo em arquiteturas *Alpha*”.(Puhlmann, 2005).

O projeto *FreeBSD* visa oferecer o software e o código para qualquer usuário sem ter que pagar nada, porém pode ser feita uma contribuição financeira(doação). O projeto tem preferência a licença BSD, como o Hubbard falou em 1995, “devido às

complexidades adicionais que podem envolver o uso comercial de software GPL, nós temos preferência pelos programas lançados sob a licença de direito autoral BSD sempre que possível, por ser uma licença consideravelmente mais flexível”.

O sistema operacional FreeBSD não vem com interface gráfica, é tudo feito por linha de comando, segue a imagem 3.1 da tela inicial:



Figura 4.1 - Tela de menu do FreeBSD Fonte
(<http://desconstruindo.eng.br/artigos/02%20-%20Introducao%20Ao%20FreeBSD%20Para%20Estacoes%20E%20Servidores%20BR.html>)

Como Lehey explicou em 2013, hoje existem 3 tipos de desenvolvedores do *FreeBDS* que são:

Contribuidores: são aqueles que escrevem códigos e a documentação, mas não tem a permissão de *committer*(adicionar o código) diretamente na árvore de códigos do *FreeBSD*. Para o código ser inserido no sistema, é necessário que um desenvolvedor registrado avalie e aprove o código, esse desenvolvedor registrado é chamado de *committer*.

Committers: são desenvolvedores que possuem acesso a escrita na árvore do código fonte. É de responsabilidade de cada *committer* a autorização da escrita na árvore de código fonte, se caso ele achar necessário pode passar o código para o grupo central avaliar. O grupo central tem o direito de retirar uma atualização, caso essa esteja com problema. Cada alteração é informada para todos os desenvolvedores via correio eletrônico, sendo assim impossível alguém fazer uma alteração secretamente.

Grupo Central(*Core Team*): esse grupo central que gerencia o projeto. Esse grupo discute e determina a direção do projeto, existem vários projetos acontecendo ao mesmo tempo. Para entrar no grupo central não precisa ser um desenvolvedor, apesar de que, geralmente o caso é esse.

Segundo Pohlmann, 2005, “É possível descrever o *FreeBSD* como um sistema operacional do administrador da rede: é rápido, com capacidade SMP e bem integrado a um grande número de ferramentas de rede”. Além de ser um sistema estável, seguro e gratuito.

Segue a imagem 3.2 do arquivo de configuração `rc.conf`:

```

^I (escape) menu    ^y search prompt  ^k delete line    ^p prev li       ^g prev page
^o ascii code      ^x search         ^l undelete line  ^n next li       ^v next page
^u end of file     ^a begin of line  ^w delete word    ^b back 1 char
^t top of text     ^e end of line    ^r restore word   ^f forward 1 char
^c command         ^d delete char    ^j undelete char  ^z next word
=====line 1 col 0 lines from top 1 =====
hostname=""
keymap="br275.iso.acc.kbd"
ifconfig_em0="DHCP"
sshd_enable="YES"
# Set dumpdev to "AUTO" to enable crash dumps, "NO" to disable
dumpdev="AUTO"
moused_enable="YES"
moused_port="/dev/psm0"
dbus_enable="YES"
hald_enable="YES"
linux_enable="YES"

file "/etc/rc.conf", 11 lines

```

Figura 4.2 – Exemplo do arquivo `rc.conf`

Neste capítulo foi apresentado o sistema operacional *FreeBSD* que é um dos melhores sistemas para sistemas paralelos e distribuídos. Um sistema estável, com um sistema de escalonamento eficiente e eficaz.

5. BEOWULF

Neste capítulo será apresentado o sistema *Beowulf*, mostrando arquitetura, será abordado também os tipos de rede mais utilizados e o monitoramento desse sistema.

5.1. ARQUITETURA

O primeiro *cluster Beowulf* foi criado em 1994 na CESDIS, uma subsidiária da NASA. Este tipo de *cluster* tem por característica o baixo custo, podendo ter resultados equivalentes ou até superiores à super máquinas. “Uma opção mais barata para instituições que precisam de um supercomputador, mas não possuem muito dinheiro disponível, é usar um sistema de processamento distribuído, ou *cluster*” (MORIMOTO, 2005).

A construção de um *cluster* pequeno é simples, porém um projeto grande será mais complexo, pois tem que se tomar decisões como: tamanho dos discos, quantidade e tecnologia de memória empregada, número de CPUs por nó, tipo de processador, número de nós, como estes nós estarão interconectados.

Existe uma maneira de configurar o servidor de arquivos poder entregar todo o sistema de arquivos aos nós de processamento que serão “*diskless*”. “Uma vez que os nós computacionais não conterão nenhuma informação de configuração, a administração será muito mais fácil, pois estará centralizada em um único ponto” (PITANGA, 2008).

Outra maneira é a utilização de discos armazenados em cada nó computacional. Neste caso é preciso a instalação do sistema em cada nó e configurado manualmente. “Neste caso não teremos tráfego em excesso na rede, mas torna o processo de administração mais difícil” (PITANGA, 2008).

A quantidade de memória é muito importante, pois se a área de *swap* em disco for muito utilizada, haverá uma queda de desempenho. Processador e tipo de rede também são muito importantes, pois podemos ter as seguintes situações:

- a) Muito processamento e pouca utilização da rede;
- b) Muito processamento e muita utilização da rede;
- c) Pouco processamento e pouca utilização da rede (será que justifica um *cluster*?);
- d) Pouco processamento e muita utilização da rede.

Cluster homogêneo é aquele em que toda estrutura tem a mesma característica de nó e de rede. Heterogêneo, é o *cluster* em que os nós ou a rede de comunicação possuem características diferentes.

Os *clusters* homogêneos por terem as mesmas características pode-se distribuir a mesma carga de processamento para os nós escravos, sendo assim, mais fácil a administração. Já no *cluster* heterogêneo por ter suas características diferentes, deve-se ter uma distribuição de acordo com o poder de processamento do nó para que não haja lentidão do sistema, o que torna a administração mais difícil. “O balanceamento de carga é a distribuição imparcial de carga de trabalho entre os recursos computacionais disponíveis em uma rede de computadores” (PITANGA, 2008).

Segue abaixo a figura com o *cluster Beowulf*:



Figura 5.1 Cluster Beowulf Fonte:

<http://theochem.chem.rug.nl/hardware/medusanodes.jpg>

5.2.TIPOS DE REDE

O tipo de rede é importante, pois a comunicação entre os nós é fundamental para o bom funcionamento do *cluster*. Redes estáticas têm ligações diretas entre dois pontos, ou seja, são conexão fixas. Redes dinâmicas, variam em relação ao tempo, não tem ligação direta entre os elementos, como é o caso da rede *ômega* e *crossbar*.

A seguir imagens dos tipos estrela, malha e *Fat Tree*, que são consideradas as redes mais utilizadas:

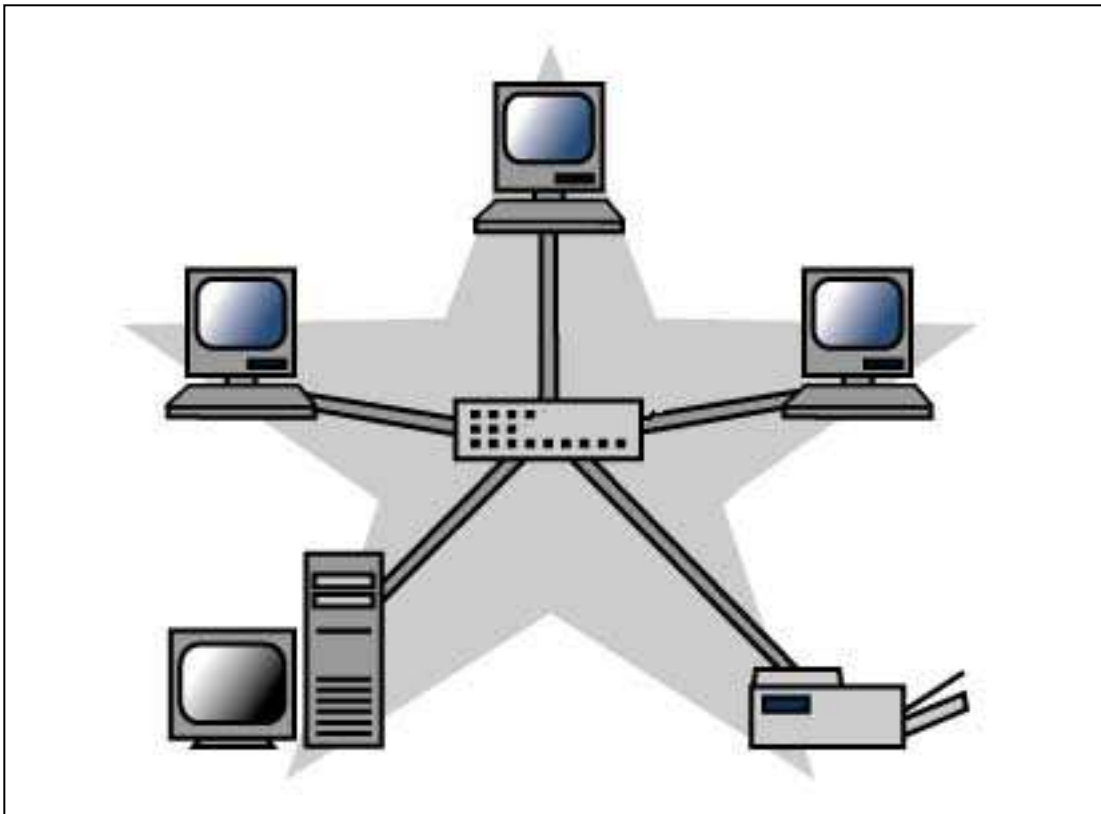


Figura 5.2 Topologia em estrela Fonte: <http://fabrica.ms.senac.br/wp-content/uploads/2013/07/estrela.jpg>

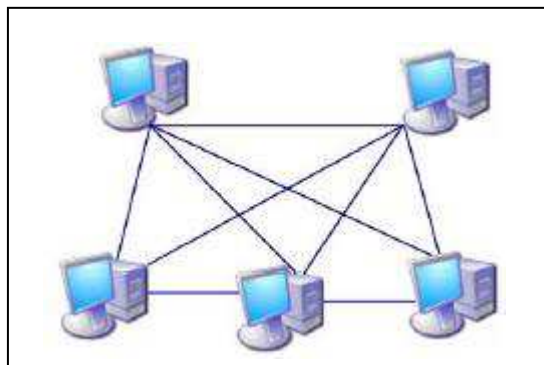


Figura 5.3 Topologia em malha Fonte: http://files.cefiredes10.webnode.pt/200000014-a39bba495b/topologia_malha.jpg

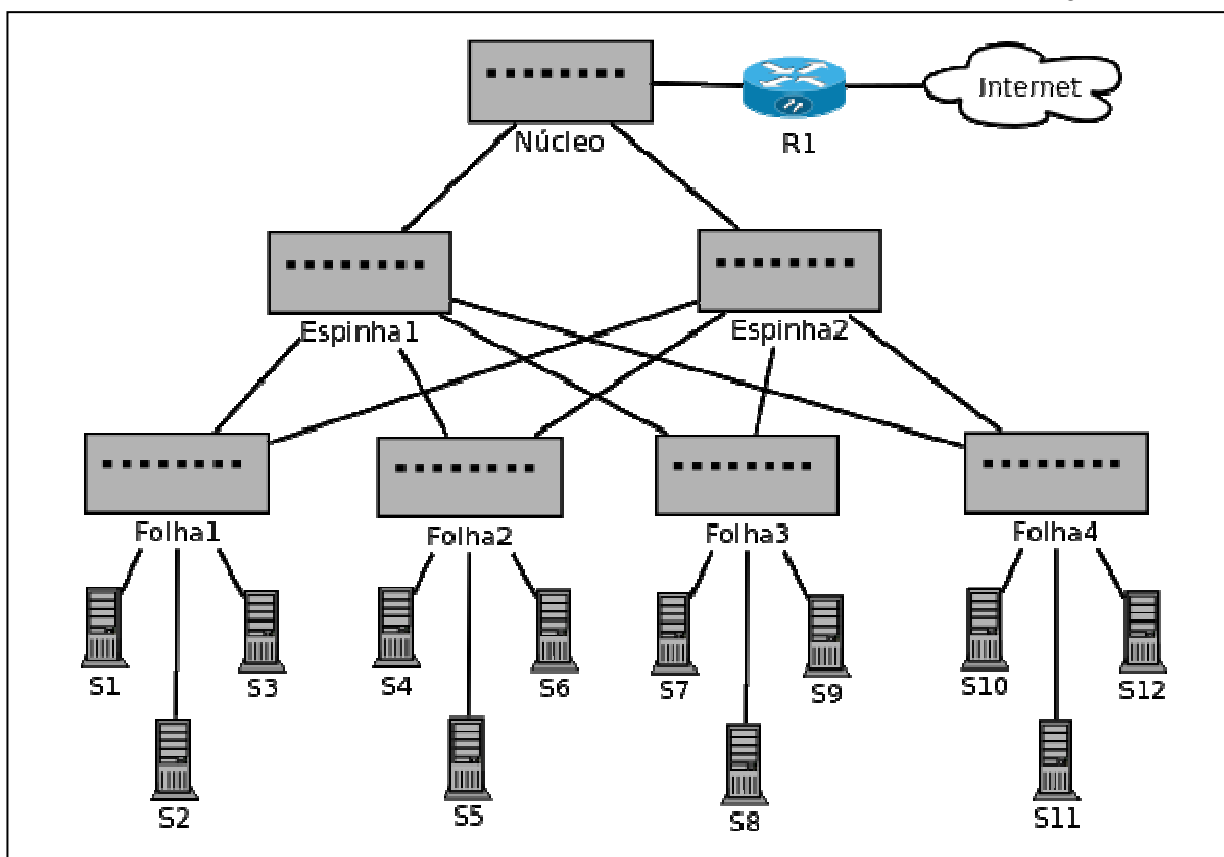


Figura 5.4 Topologia *Fat Tree* Fonte:

<http://wiki.sj.ifsc.edu.br/wiki/images/e/ee/Fat-tree3.png>

A ligação direta entre os nós torna a rede mais rápida, porém mais complexa, com problemas de escalabilidade, ou seja, cada topologia tem vantagens e desvantagens. “Ainda que a topologia totalmente conectada provenha muito mais velocidade em termo de comunicação, esta também possui um alto custo de implementação em muito dos casos e tem problemas de escalabilidade” (PITANGA, 2008).

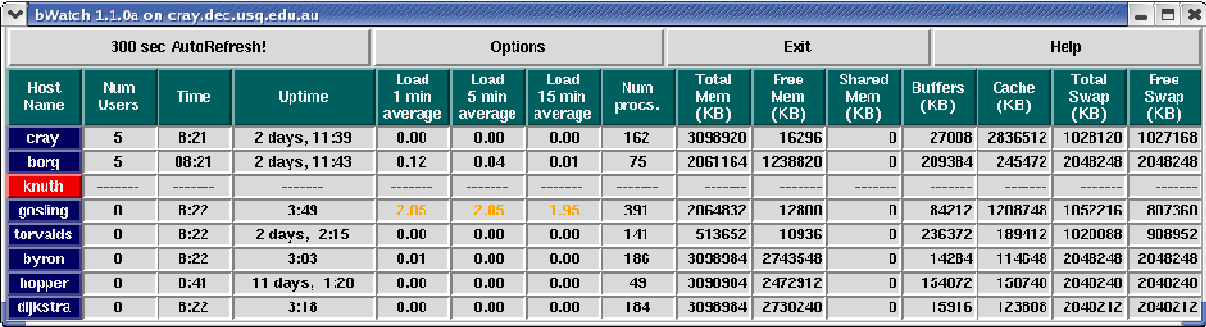
A latência e a largura de banda são os termos de medida de desempenho da rede. O tempo em que software leva para montar e transferir os bits é chamado de latência. O número de bits que podem ser transmitidos pela rede é chamado de largura. “O ideal para o bom funcionamento de uma aplicação é ter uma rede com baixa latência e grande largura de banda” (PITANGA, 2008).

5.3.MONITORAMENTO

O gerenciamento desse aglomerado não é uma tarefa muito fácil, porém existem algumas ferramentas para auxiliar na manutenção do supercomputador. “São ferramentas como bWatch, Ganglia, Nagios, dentre outras, que facilitam a administração do sistema e monitoram a saúde do cluster, assim como permitem descobrir possíveis gargalos de performance” (PITANGA, 2008).

Para monitorar a carga e o uso da memória em todos os nós do *cluster*, existem alguns gerenciadores, como *bWatch*(*beowulf Watch*). Com o *bWatch* consegue-se identificar se um nó parou as atividades, ou se está com uma demora no tempo de resposta. “O bWatch é um script escrito na linguagem interpretada gráfica (Tcl/Tk) designado para monitorar nosso supercomputador desenvolvido por Jacek Radajewski” (PITANGA, 2008).

Segue a figura 5.5 que mostra a interface do bWatch:



300 sec AutoRefresh!				Options				Exit			Help			
Host Name	Num Users	Time	Uptime	Load 1 min average	Load 5 min average	Load 15 min average	Num procs.	Total Mem (KB)	Free Mem (KB)	Shared Mem (KB)	Buffers (KB)	Cache (KB)	Total Swap (KB)	Free Swap (KB)
cray	5	8:21	2 days, 11:39	0.00	0.00	0.00	162	3098920	16296	0	27008	2836512	1028120	1027168
borg	5	08:21	2 days, 11:43	0.12	0.04	0.01	75	2061164	1238820	0	209384	245472	2048248	2048248
knuth	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
gnsing	0	8:27	3:49	2.05	2.05	1.95	391	2064832	12800	0	84212	1208748	1052216	807360
tarvalds	0	8:22	2 days, 2:15	0.00	0.00	0.00	141	513652	10936	0	236372	189412	1020088	908952
byron	0	8:22	3:03	0.01	0.00	0.00	186	3098984	2743548	0	14284	114648	2048248	2048248
hopper	0	0:41	11 days, 1:20	0.00	0.00	0.00	49	3090904	2472312	0	154072	150740	2040240	2040240
uijkstra	0	8:22	3:18	0.00	0.00	0.00	184	3098984	2730240	0	15916	123608	2040212	2040212

Figura 5.5 Interface bWatch Fonte: <http://bwatch.sourceforge.net/>

Neste capítulo foi apresentado o conceito, estrutura do sistema *Beowulf*, assim como, os tipos de redes e o monitoramento. Devendo sempre realizar um estudo antes de tomar qualquer decisão, como escolha de disco, memória, processador, tipo de rede.

6. IMPLEMENTAÇÃO

Neste capítulo será apresentado as configurações necessárias para executar um *cluster* utilizando o sistema operacional *FreeBSD*.

Para poder fazer a inicialização do sistema sem disco, é preciso utilizar placas de rede com uma propriedade chamada PXE(*pre-boot eXecution Enviroment*), essa propriedade permite a inicialização via rede.

O nó mestre tem a configuração dos softwares que são distribuídos para os nós escravos. Basicamente são 3 *softwares* para a inicialização que são:

- DHCP
- TFTP
- NFS

A configuração dos nós escravos também fica no nó mestre, o arquivo raiz que é onde ficam os principais arquivos do sistema, um kernel reduzido e com suporte a inicialização sem disco e configurações de scripts para a inicialização do sistema.

6.1 INSTALAÇÃO DO FREEBSD

A instalação no nó mestre é da versão 9.1. Segue algumas informações da instalação:

Ponto de montagem	Tamanho	Descrição
/	1GB	Sistema de arquivos raiz do sistema
/usr	10GB	Aplicativos do sistema
/home	Indefinido	Compartilhamento de arquivos
/cluster	1GB	Sistema para os nós escravos
Swap	4GB	Memória virtual

Figura 6.1 – Exemplo de configuração para instalação

Os pacotes básicos para a instalação são:

- *Base*
- *Kernel(GENERIC)*
- *Ports*
- *src*

O nó escravo também é instalado no nó mestre, porém numa partição diferente do nó mestre, pois a inicialização do nó escravo é diferente.

A partição */cluster* é onde é instalado o sistema operacional dos nós escravos que tem os seguintes pacotes:

- *Base*
- *Kernel(GENERIC)*

Sem habilitar nenhum *daemon* e sem criar nenhum usuário, a instalação se encerra.

6.2 CONFIGURAÇÃO DO NÓ MASTER

A configuração do nó mestre é a instalação dos serviços de rede que são disponibilizados para os nós escravos, foi configurado na ordem em que os serviços são solicitados durante a inicialização, DHCP, TFTP e NFS.

A instalação do DHCP é feita via *ports*, o caminho do *ports* é: `/usr/ports/net/isc-dhcp4-server/`, para executar a instalação tem que estar logado como usuário *root* e executar os seguintes comandos:

```
# cd /usr/ports/net/isc-dhcp4-server/  
# make install
```

Figura 6.2 – Comandos de instalação do DHCP-SERVER

O arquivo de configuração do DHCP está localizado no diretório `/usr/local/etc/` com o nome `dhcpd.conf`, segue abaixo em trecho do arquivo configurado:

```
default-lease-time 600;  
  
    max-lease-time 7200;  
  
    authoritative;  
  
    ddns-update-style ad-hoc;  
  
    option domain-name "fema.net";  
  
    option routers 192.168.0.1  
  
    subnet 192.168.0.0 netmask 255.255.255.0 {  
        use-host-decl-names on;  
  
        option subnet-mask 255.255.255.0;  
  
        option broadcast-address 192.168.0.255;  
  
        host s2.cluster.fema.net {  
            hardware ethernet 00:0D:60:E0:A8:A4;  
  
            fixed-address 192.168.0.202;  
  
            next-server 192.168.0.200;  
  
            filename "/pxeboot";
```

```
    option root-path "192.168.0.200:/cluster/";
}
host s3.cluster.fema.net {
    hardware ethernet 00:0D:60:E0:84:D4;
    fixed-address 192.168.0.203;
    next-server 192.168.0.200;
    filename "/pxeboot";
    option root-path "192.168.0.200:/cluster/";
}
host s4.cluster.fema.net {
    hardware ethernet 00:0D:60:E0:83:8A;
    fixed-address 192.168.0.204;
    next-server 192.168.0.200;
    filename "/pxeboot";
    option root-path "192.168.0.200:/cluster/";
}
host s5.cluster.fema.net {
    hardware ethernet 00:09:6B:DF:39:3A;
    fixed-address 192.168.0.205;
    next-server 192.168.0.200;
    filename "/pxeboot";
    option root-path "192.168.0.200:/cluster/";
}
host s6.cluster.fema.net {
    hardware ethernet 00:0D:60:E0:A7:CC;
```

```

fixed-address 192.168.0.206;

next-server 192.168.0.200;

filename "/pxeboot";

option root-path "192.168.0.200:/cluster/";

}

```

Figura 6.3 – Exemplo do arquivo de configuração do DHCP

Para que o DHCP seja inicializado juntamente com o sistema operacional é preciso adicionar a seguinte linha ao arquivo `/etc/rc.conf`:

```

dhcpcd_enable="YES"

inetd_enable="YES"

```

Figura 6.4 – Arquivo de configuração `/etc/rc.conf`

O TFTP já vem instalado nativamente, porém para poder alterar o modo somente leitura deve-se editar o arquivo `/etc/hosts:allow`, para que fique como no exemplo a seguir:

```

tftpd: 192.168.0.0/255.255.255.0 :allow

```

Figura 6.5 – Arquivo `/etc/hosts:allow`

É preciso também configurar o arquivo `/etc/inetd.conf` para que serviço seja executado junto a inicialização do sistema. Basta somente remover o comentário da linha referente ao tftp:

```

tftp dgram udp wait root /usr/libexec/tftpd tftpd -l -s /tftpboot

```

Figura 6.6 – Arquivo `/etc/inetd.conf`

Com o TFTP configurado, é preciso copiar o arquivo de boot para o diretório `/tftpboot`:

```

# cp /boot/pxeboot /tftpboot

```

Figura 6.7 – Comando para copiar o arquivo `/boot/pxeboot`

Com o TFTP configurado, agora é a vez do NFS que irá exportar os sistemas de arquivos para os outros nós do cluster. Veja abaixo alguns arquivos exportados e suas propriedades.

Ponto de montagem	Propriedades	Descrição
/cluster nós escravos	Somente Leitura	Sistema de arquivos raiz dos
/usr	Somente Leitura	/usr dos nós escravos
/home compartilhado	Leitura e escrita	/home, Diretório de usuários

Figura 6.8 – Arquivos Exportados

O arquivo de configuração do NFS está em `/etc/exports`, veja abaixo como ficou o arquivo utilizado em nosso *cluster*:

```
/cluster -alldirs -maproot=0:0 -ro -network 192.168.0.200 -mask 255.255.255.0
/usr -alldirs -maproot=0:0 -ro -network 192.168.0.200 -mask 255.255.255.0
/home -alldirs -maproot=0:0 -network 192.168.0.200 -mask 255.255.255.0
```

Figura 6.9 – Arquivo `/etc/exports`

Para que o NFSD seja inicializado juntamente com o sistema operacional, é adicionado as seguintes linhas ao arquivo `/etc/rc.conf`:

```
nfs_server_enable="YES"
rpcbind_enable="YES"
```

Figura 6.10 – Arquivo `/etc/rc.conf`

A configuração do serviço está pronta.

6.3 CONFIGURAÇÃO DOS NÓS ESCRAVOS

A Configuração dos nós escravos é feita no nó mestre, pois somente ele terá acesso total ao sistema de arquivo raiz dos nós escravos. Seguindo a tabela 1 o sistema de arquivos raiz dos nós escravos está montado sobre o diretório /cluster no nó mestre.

Construindo um *kernel* com suporte a inicialização em rede.

Um novo *kernel* deve ser compilado com suporte ao *boot* na rede. Os arquivos de configuração do *kernel* estão localizados em /usr/src/sys/i386/conf/ e o arquivo padrão de configuração é nomeado com *GENERIC*. Copie o arquivo *GENERIC* para outro arquivo com o nome *DISKLESS*, e adicione as seguintes linhas:

```
options BOOTP
options BOOTP_NFSROOT
options BOOTP_NFSV3
options BOOTP_COMPAT
```

Figura 6.11 – Comandos para adicionar no arquivo DISKLESS

O arquivo genérico de configuração do *kernel* do *FreeBSD* trás vários módulos compilados estaticamente com o *kernel*, em nosso sistema apenas deixaremos habilitados o necessário para o sistema sem disco, pode-se retirar várias opções, entre elas, suporte a disco, a placas de redes não utilizadas, dispositivos ATA, SCSI, entre outros, veja baixo um exemplo de como pode ser configurado o arquivo.

DISKLESS arquivo de configuração:

```
machine      i386
cpu          i686_CPU
ident        DISKLESS

options      SCHED_4BSD          # 4BSD scheduler
options      PREEMPTION         # Enable kernel thread preemption
```



```

options    INET                # InterNETworking
options    FFS                # Berkeley Fast Filesystem
options    MD_ROOT           # MD is a potential root device
options    NFSCLIENT        # Network Filesystem Client
options    NFS_ROOT         # NFS usable as /, requires NFSCLIENT
options    PSEUDofs         # Pseudo-filesystem framework
options    GEOM_GPT         # GUID Partition Tables.
options    COMPAT_43        # Compatible with BSD 4.3 [KEEP
THIS!]
options    COMPAT_FREEBSD4  # Compatible with FreeBSD4
options    COMPAT_FREEBSD5  # Compatible with FreeBSD5
options    KTRACE           # ktrace(1) support
options    SYSVSHM          # SYSV-style shared memory
options    SYSVMSG          # SYSV-style message queues
options    SYSVSEM          # SYSV-style semaphores
options    _KPOSIX_PRIORITY_SCHEDULING # POSIX P1003_1B real-time
extensions
options    KBD_INSTALL_CDEV # install a CDEV entry in /dev
options    AHC_REG_PRETTY_PRINT # Print register bitfields in debug
options    AHD_REG_PRETTY_PRINT # Print register bitfields in debug
options    ADAPTIVE_GIANT   # Giant mutex is adaptive.
device    apic              # I/O APIC
options    BOOTP            #OPÇÕES DE BOOT
options    BOOTP_NFSROOT    #PELA REDE
options    BOOTP_NFSV3

```

```

options      BOOTP_COMPAT

device      eisa

device      pci

# atkbdc0 controls both the keyboard and the PS/2 mouse

device      atkbdc          # AT keyboard controller

device      atkbd          # AT keyboard

device      vga            # VGA video card driver

device      sc

device      agp            # support several AGP chipsets

device      pmtimer

# Network devices.

device      miibus        # MII bus support

device      fxp           # Intel EtherExpress PRO/100B (82557, 82558)

device      xl

# Pseudo devices.

device      loop          # Network loopback

device      random        # Entropy device

device      ether         # Ethernet support

device      tun           # Packet tunnel.

device      pty           # Pseudo-ttys (telnet etc)

device      md            # Memory "disks" UTILIZADO EM FS DE
MEMÓRIA

```

Figura 6.12 – Exemplo de arquivo de configuração do kernel

Salve o arquivo de configuração do *kernel* com o nome *DISKLESS* e o compile com os seguintes comandos:

```
# cd /usr/src/sys/i386/conf/
# config DISKLESS
# cd ../compile/DISKLESS
# make cleandepend; make depend
# make all
# make DESTDIR=/cluster install
```

Figura 6.13 – Comandos para compilar o kernel

O *kernel* está instalado.

O diretório */etc* é substituído durante a inicialização do sistema sem disco, ele é substituído por um diretório dentro de */conf/\$IP/etc*, onde *\$IP* é o endereço de IP do nó escravo, caso este diretório não existir, ele será substituído por */conf/default/etc*. Neste exemplo será apresentado somente o diretório */conf/default/etc*. Copie todos os arquivos dentro do */etc* para */conf/default/etc*, relativos ao endereço raiz dos nós escravos, neste caso */cluster/etc/* e */cluster/conf/default/etc/*.

```
# cp -rf /cluster/etc/ /cluster/conf/default/etc/
```

Figura 6.14 – Comando para copiar os arquivos do */cluster/etc/*

Agora todos os arquivos necessários para a inicialização estão criados, agora deve ser editado alguns, como o */cluster/conf/default/fstab* e *cluster/conf/default/etc/rc.conf*

O *fstab* terá que ser alterado, para montar os sistemas de arquivo a partir do nó mestre, abaixo veremos o *fstab* de exemplo:

192.168.0.1:/cluster/	/	nfs	ro	0	0
192.168.0.1:/usr/	/usr	nfs	ro	0	0
192.168.0.1:/home/	/home	nfs	rw	0	0
md	/var	mfs	rw,-s32m	0	0

Figura 6.15 – Exemplo do *fstab*

As 3 primeiras linhas representam os sistemas de arquivos importados do nó mestre, a última linha é um detalhe importante; como as partições são exportadas como somente leitura, então os arquivos de *log*, *sockets*, arquivos de *pid*, que são necessários para o sistema não poderão ser escritos. A solução para isto é o *md*, o *md* é um pseudo sistema de arquivos de memória, ele aceita leitura e escrita, seus dados são gravados em memória e serão perdidos após o sistema desligar, como os arquivos que serão gravados sobre */var* tem estado temporários para este caso, o *md* é perfeito para isto.

rc.conf

O *rc.conf* é o arquivo de configuração de inicialização do sistema, por isso são necessários alguns ajustes nele.

Alguns serviços vem habilitados por padrão junto com a inicialização do *FreeBSD*, porém alguns deles não são necessários, pode-se desabilitá-los. Somente um serviço, fora do padrão, deve ser habilitado, o *inetd*, veremos a configuração dele mais a frente. Veja abaixo o exemplo de como pode ser configurado o arquivo */cluster/conf/default/etc/rc.conf*:

```
sendmail_enable=NONE
sendmail_submit_enable=NO
sendmail_msp_queue_enable=NO
inetd_enable=YES
```

Figura 6.16 Exemplo do arquivo */cluster/conf/default/etc/rc.conf*

Os usuários do sistema deverão ser os mesmos do nó mestre, existem varias formas dinâmicas de fazermos isto, porém será apresentado uma forma para poucos usuários, a melhor solução é apenas replicar os arquivos de usuário do nó mestre para os nós escravos. Para isto execute o seguinte comando:

```
# cd /etc
# cp master.passwd passwd pwd.db spwd.db /cluster/conf/default/etc/
```

Figura 6.17 – Comando para replicar os usuários

Agora será exemplificado apenas a instalação e configuração de duas das principais bibliotecas de programação paralela, a MPICH2 e a PVM, para maiores informações leia as páginas dos manuais das bibliotecas.

Integrando os nós via MPICH

A primeira parte já está pronta, já temos um sistema homogêneo e com discos compartilhados, porém isto ainda não é um *cluster*. O cluster, seguindo a definição de *cluster beowulf*, será dedicado a execução de processos paralelos. Para isto é utilizado a biblioteca MPI - *Message Passing Interface*, ela é uma biblioteca de troca de mensagem entre processos paralelos. Cada processo trabalha com um conjunto de dados, e eles a utilizam para acessar e modificar os dados de outros processos, assim criando um ambiente de compartilhamento de memória. Será visto a utilização da MPICH, que é uma implementação livre da MPI. Ela está disponível para instalação no *FreeBSD* via *ports*.

O *ports* da MPICH está no diretório `/usr/ports/net/mpich`, para iniciar a instalação, execute os seguintes comandos:

```
# cd /usr/ports/net/mpich
# make install
```

Figura 6.18 – Comandos para instalar o MPICH

Se nada anormal acontecer, a MPICH estará instalada. Ela é instalada sobre o diretório `/usr/local/mpich`, e seus executáveis ficarão dentro do diretório `/usr/local/mpich/bin`, deve-se adicionar este diretório a variável `PATH` do ambiente, para isto execute o seguinte comando.

```
# setenv PATH $PATH:/usr/local/mpich/bin
```

Figura 6.19 – Comando para adicionar a variável PATH

A MPICH utiliza uma *shell* remota para a execução de processos, atualmente ela pode utilizar o RSH ou o SSH, como o foco não é a segurança será apresentado a configuração do RSH, pois o SSH utiliza criptografia, o que adiciona uma carga maior de processamento.

O rsh é um serviço de *login* remoto, ele será utilizado pela MPICH para a comunicação entre processos. Para habilitá-lo precisa-se apenas remover os comentários das duas linhas do arquivo `/cluster/conf/default/etc/inetd.conf`, as linhas referentes ao rshd e logind, veja abaixo as linhas:

```
shell stream tcp  nowait root  /usr/libexec/rshd    rshd
login stream tcp  nowait root  /usr/libexec/rlogind rlogind
```

Figura 6.20 – Arquivo `/cluster/conf/default/etc/inetd.conf`

Agora é preciso liberar o acesso sem senha a *shell* remota de todos os nós do *cluster*, isto é feito adicionando os endereços de ip, um por linha, de todos os nós do *cluster*, ao arquivo, `/cluster/conf/default/etc/hosts.equiv`, como no exemplo abaixo:

```
192.168.0.1
192.168.0.2
192.168.0.3
192.168.0.4
192.168.0.5
192.168.0.6
192.168.0.7
192.168.0.8
```

Figura 6.21 – Exemplo do arquivo `/cluster/conf/default/etc/hosts.equiv`

Arquivos de tradução de nomes em IP

A MPICH comunica-se através de nomes de hosts não por endereços de ip, como se trata de um sistema pequeno, será mostrado o arquivo `/etc/hosts`. A sintaxe do arquivo é: endereçoDeIP NomeDoHost, como no exemplo abaixo:

192.168.0.1	mestre.cluster.fema.net mestre
192.168.0.2	s2.cluster.fema.net s2
192.168.0.3	s3.cluster.fema.net s3
192.168.0.4	s4.cluster.fema.net s4
192.168.0.5	s5.cluster.fema.net s5
192.168.0.6	s6.cluster.fema.net s6
192.168.0.7	s7.cluster.fema.net s7
192.168.0.8	s8.cluster.fema.net s8

Figura 6.22 – Exemplo do arquivo /etc/hosts

Agora os programas que utilizam a MPICH já podem ser executados em no sistema.

PVM - *Parallel Virtual Machine* fornece um conjunto integrado de bibliotecas e de ferramentas de *software*, que emula um ambiente de programação paralela que permite agregar, de maneira transparente, arquiteturas homogêneas ou heterogêneas.

No *FreeBSD* a PVM está disponível para instalação via ports, cuja localização é `/usr/ports/net/pvm`. Para instalá-la, é necessário executar somente um comando, *make install*. Após a instalação, é preciso declarar a variável de ambiente PVM_ROOT com o diretório `/usr/local/lib/pvm`. Para isso, adiciona-se a seguinte linha de comando ao arquivo `/etc/csh.cshrc`:

```
setenv PVM_ROOT /usr/local/lib/pvm.
```

Figura 6.23 – Arquivo /etc/csh.cshrc

A configuração da PVM é realizada através de um *prompt*. O comando `pvm` inicia o *daemon* de `pvm3d`, que é o *daemon* de monitoramento do cluster, e após abre o *prompt*. Porém, antes de iniciar o *cluster*, é preciso configurar o RSH, como para a MPI e o arquivo de hosts.

A configuração necessita da mesma configuração do `/etc/inetd`, `/etc/hosts.equiv` e `/etc/hosts` que a MPICH precisa.

Para finalizar execute o comando `pvm` e depois adicionar os demais nós com o comando `add` do prompt do `pvm`.

```
$ pvm
pvm> add s2
pvm> add s3
pvm> add s4
pvm> add s5
pvm> add s6
pvm> add s7
pvm> add s8
pvm> quit
```

Figura 6.24 – Comandos de configuração da PVM

Agora o Cluster está pronto.

Neste capítulo foi apresentado uma dentre varias formas de se configurar um *cluster* utilizando o sistema operacional *FreeBSD* e as bibliotecas MPI e PVM. Foram apresentados exemplos dos arquivos de configuração e comandos de instalação e execução das ferramentas necessárias.

7. CONCLUSÃO

Com esse estudo foi possível conhecer novas tecnologias, conhecer melhor as estruturas e arquiteturas que existem, mas que também pode-se evoluir ainda mais. É uma área em que vem crescendo e se tornando imprescindível cada vez mais, porém existem alguns casos em que não é recomendado se utilizar este tipo de serviço.

Antes de começar a construir um *cluster* deve-se fazer um estudo sobre as necessidades, para poder tomar as decisões corretas sobre a arquitetura, características que o *cluster* vai ter, ou seja, qual é a melhor solução.

Com um custo reduzido, pode-se fazer a construção do *cluster* até mesmo com computadores mais antigos, é possível se construir um supercomputador com vários processadores, um alto nível de processamento.

As bibliotecas de paralelismo vem evoluindo garantindo sistemas mais estáveis com melhor performance, com algoritmos e protocolos mais otimizados. Os sistemas operacionais também tem grande influencia para o crescimento dessa tecnologia, pois com sistemas gratuitos foi possível obter melhores resultados, conseguindo gerenciar melhor e mais rápido o fluxo dos dados.

Para trabalhos futuros pode ser feito testes com diferentes tipos de rede, diferentes estruturas, pesquisa sobre programação paralela e sistemas distribuídos.

REFERÊNCIAS

PITANGA, Marcos. **Construindo Supercomputadores com Linux**. 3. ed. Rio de Janeiro: Brasport, 2008. 374 p.

PINA, António M. **Máquina Paralela Virtual(PVM)**. Departamento de informática – Universidade do Minho – Campos de Gualtar – Braga – Portugal. Está disponível em: <http://www3.di.uminho.pt/~amp/textos/pvm/pvm.html>. Acesso em: 18 Jun. 2013

PESSOA, Márcio **Um cluster HPC com PVM**. Está disponível em: <http://pessoa.eti.br/main/2003/05/14/um-cluster-hpc-com-pvm/>. Acesso em: 18 Jun. 2013

FERREIRA, Virgílio J. F. MPI: **UMA FERRAMENTA PARA IMPLEMENTAÇÃO PARALELA**. - Universidade Federal do Rio de Janeiro - Departamento de Engenharia Industrial - Rio de Janeiro. Está disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382002000100007. Acesso em: 10 Jun. 2013

KUNIGAMI, Guilherme. **Biblioteca MPI**. Está disponível em: <http://kuniga.wordpress.com/2010/09/17/biblioteca-mpi/>. Acesso em: 10 Jun. 2013

POHLMANN, Frank **Por que FreeBSD**. Está disponível em: <http://www.ibm.com/developerworks/br/library/os-freebsd/>. Acesso em: 07 Jun. 2013

LEHEY, Greg **Explicando BSD**. Está disponível em: http://www.freebsd.org/doc/pt_BR.ISO8859-1/articles/explaining-bsd/index.html. Acesso em: 07 Jun. 2013

HUBBARD, Jordan **FreeBSD Handbook**. Está disponível em: <http://www.openit.com.br/freebsd-hb/history.html>. Acesso em: 07 Jun. 2013

FINS, Bruno **O que é cluster e como funciona?** Está disponível em: <http://faqinformatica.com/o-que-e-cluster-e-como-funciona/>. Acesso em: 22 Fev. 2013

MORIMOTO, Carlos E. **Brincando de cluster**. Está disponível em: <http://www.hardware.com.br/artigos/cluster/>. Acesso em: 22 Fev. 2013