UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Madianita Bogo

INTERFACE DA REDE DE CONTROLE DO CLUSTER CLUX

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação.

Prof. Thadeu Botteri Corso

Orientador

Florianópolis, fevereiro de 2002.

INTERFACE DA REDE DE CONTROLE DO CLUSTER CLUX

Madianita Bogo

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Fernando Álvaro Ostuni Gauthier

Banca Examinadora

Prof. Thadeu Botteri Corso, Dr. (Presidente)

Prof. José Mazzuco Junior, Dr.

Prof. Luis Fernando Friedrich, Dr.

Prof. Rômulo Silva de Oliveira , Dr.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACK	Reconhecimento
ARCNET	*Attached Resource Computer Network*
ATM	*Asynchronous Transfer Mode*
CORBA	*Common Object Request Broker Arquitecture*
FC	*Fibre Channel*
FTP	*File Transfer Protocol*
GAMMA	*Genoa Active Message Machine*
GNC	Gerente do Nó de Controle
GNTR	Gerente do Nó de Trabalho Real
HiPPI	*High Performance Parallel Interface*
HTTP	*Hyper Text Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IGMP	*Internet Group Management Protocol*
IEEE	*Institute Electrical and Eletronics Engineers*
IPC	*Interprocess Comunication*
LAM	*Local Área Multicomputer*
LAN	*Local Área Network*
LPI	Lista de pedidos Indefinidos
LPL	Lista de Pedidos Locais
LPR	Lista de Pedidos Remotos
MAN	*Metropolitan Area Networks*
MIMD	*Multiple Instruction Multiple Data*
MISD	*Multiple Instruction Single Data*
MPI	*Message Passing Interface*
MTI	*Massachussets Institute of Technology*
NC	Nó de Controle
NTR	Nó de Trabalho Real
NTV	Nó de Trabalho Virtual
PVM	*Parallel Virtual Machine*
RTT	*Round Trip Time*

SAN	*System Area Network*
SIMD	*Single Instruction Multiple Data*
SISD	*Single Instruction Single Data*
SMTP	*Simple Mail Transfer Protocol*
SO	Sistema Operacional
SYN	Sincronismo
TCGNC	Tabela de Conexões Remotas
TCGNTR	Tabela de Conexões Locais
TCP/IP	*Transmission Control Protocol / Internet Protocol*
UDP	*User Datagram Protocol*
WAN	*Wide Área Network*

RESUMO

Este trabalho faz parte do projeto de construção do Clux, um cluster para a exploração do processamento paralelo em um sistema constituído por um conjunto de estações de trabalho independentes, conectadas por uma rede de interconexão dinâmica. O objetivo deste trabalho é a implementação da interface da rede de controle do cluster Clux, responsável pelo transporte de mensagens de controle utilizadas para a gerência da configuração da rede de interconexão dinâmica.

Palavras-chave: Cluster de Computadores, Mecanismo de Comunicação, Sistemas Operacionais.

ABSTRACT

This work is part of the Clux construction project, a cluster for the parallel processing exploration in a system consisting of a set of independent workstations, connected by a dynamic interconnection network. The objective of this work is the implementation of the cluster Clux's control network interface, responsible for the control messages passing for the management of the configuration of the dynamic interconnection network.

1 Introdução

Neste capítulo, são apresentadas as motivações e os objetivos que levaram a execução deste trabalho. Também é descrita a organização do restante do texto.

1.1 Motivações e objetivos

A crescente procura por tecnologias cada vez mais eficientes e a necessidade de máquinas com alto poder computacional em diversas áreas, tornou necessário o aparecimento de novos ambientes confiáveis capazes de reduzir custos e otimizar a obtenção de resultados. Uma alternativa com boa relação custo/desempenho é a utilização de *clusters,* que são máquinas compostas por microcomputadores comuns (nós) interligados por uma rede de interconexão, trabalhando como uma máquina virtual capaz de executar aplicações paralelas e distribuídas.

A realização de projetos baseados em *clusters* de computadores tem crescido com grande intensidade. Mesmo assim, existe uma crescente necessidade de projetos que explorem as vantagens do processamento paralelo.

No Laboratório de Computação Paralela e Distribuída (LaCPaD) do departamento do Curso de Pós–Graduação em Ciência da Computação (CPGCC) da Universidade Federal de Santa Catarina (UFSC), está sendo desenvolvido o projeto de um *cluster* para a execução paralela de programas, denominado Clux.

A arquitetura do Clux foi baseada no multicomputador Crux [COR, 1999], que é um ambiente destinado a executar uma rede de processos comunicantes, visando o paralelismo real.

A idéia é a criação de um *cluster* diferente dos existentes atualmente, trazendo algumas inovações vantajosas para o processamento paralelo, como:

- Criação dinâmica de canais físicos diretos, que são conexões entre dois nós, de acordo com a demanda das tarefas que estão sendo executadas.
- Comunicação através de troca de mensagens realizadas por operadores próprios, adaptados do multicomputador Crux.

- Utilização dos canais físicos por apenas uma conexão em um determinado momento.
- Administração das conexões por um nó de controle, através de mensagens de controle trocadas através de uma rede de controle, evitando o *overhead* na comunicação entre os nós, pois só depois que as conexões estiverem estabelecidas é que as mensagens trafegão por outra rede, denominada rede de trabalho.
- Cada nó pode estar conectado com mais de um nó simultaneamente, ampliando a variedade de aplicações que podem ser executadas no *cluster*.

A implementação do mecanismo de comunicação do Clux foi dividida em implementação da interface da rede de controle e a implementação da interface da rede de trabalho.

O objetivo principal deste trabalho é a implementação da interface da rede de controle, que é responsável pela troca de mensagens de controle para a configuração da rede de trabalho. Paralelamente a este trabalho, está sendo desenvolvida a interface da rede de trabalho, que é responsável pela troca de mensagens entre os processos executando no Clux [REC, 2002].

Este texto é constituído de sete capítulos, os quais têm seu conteúdo básico listados a seguir:

- O capítulo 2 descreve os sistemas operacionais UNIX e LINUX, apresentando um pequeno histórico, principais características, estrutura, sistema de arquivos e funcionamento dos processos.
- O capítulo 3 introduz alguns aspectos de rede de computadores, descreve o protocolo de comunicação TCP/IP e a comunicação através de *sockets*.
- O capítulo 4 trata das máquinas paralelas, apresentando uma descrição básica e a classificação das mesmas, a descrição dos multicomputadores e uma visão geral do multicomputador Crux.
- O capítulo 5 descreve alguns aspectos dos *clusters* de computadores, algumas arquiteturas de rede e apresenta projetos de *clusters* existentes.

- O capítulo 6 apresenta o *cluster* Clux, descrevendo sua arquitetura, seu funcionamento básico, sua interface de comunicação, enfocando a implementação da interface da rede de controle que é o objetivo deste trabalho.

2 Sistema Operacional UNIX

Neste capítulo, são apresentados os sistemas operacionais UNIX e LINUX, sendo dividido em 6 subcapítulos. O subcapítulo 2.1 apresenta um breve histórico do UNIX. O subcapítulo 2.2 descreve as principais características do UNIX. O subcapítulo 2.3 apresenta a estrutura do UNIX e explica os seus componentes básicos. O subcapítulo 2.4 apresenta os processos do UNIX, citando os estados possíveis, estrutura e mecanismos de comunicação utilizados. O subcapítulo 2.5 apresenta uma visão geral do sistema de arquivos do UNIX. O subcapítulo 2.6 apresenta o sistema operacional LINUX, descrevendo um breve histórico, sua relação com sistema operacional UNIX e a diferença básica em relação a este.

2.1 Histórico

Pesquisadores do M.T.I (*Massachussets Institute of Technology*), da General Eletrics e dos Laboratórios Bell (uma subsidiária da AT&T) se uniram para trabalhar no projeto do MULTICS (*Multiplex Information and Computing Service*), um sistema operacional de grande porte que visava a utilização da mesma máquina por centenas de usuários. O projeto foi não foi concluído, mas Ken Thompson, pesquisador dos Laboratórios Bell, resolveu rescrever uma versão menos ambiciosa do MULTICS. No início o novo sistema foi chamado de UNICS (*UNIplexed Information and Computing Service*), numa paródia ao nome MULTICS, e posteriormente passou a se chamar UNIX.

Diversas versões do UNIX foram lançadas, sendo que, entre as atuais, as principais são a versão *System V Release 4,* desenvolvida pela AT&T, e a 4.4BSD, desenvolvida pela Universidade da Califórnia. Várias tentativas vêm sendo feitas visando padronizar essas versões do UNIX, além das inúmeras outras oferecidas pelo mercado. A primeira tentativa séria nesse sentido foi dada pelo IEEE (*Institute Electrical and Electronics Engineers)*, através do seu comitê POSIX (*Portable Operating System Interface*) que originou o padrão IEEE 1003, o qual define um conjunto de rotinas de biblioteca que todo sistema UNIX deve fornecer.

2.2 Características

O UNIX é um sistema operacional multiusuário e multitarefa muito poderoso. Várias características explicam o sucesso alcançado por ele:

- foi escrito em uma linguagem de alto nível, tornando-o fácil de compreender, alterar e portar para outras plataformas;
- desenho modular do sistema operacional, que permite aos usuários acrescentar ou remover partes (módulos) para adaptá-lo às suas necessidades específicas;
- sistema de arquivos implementa uma estrutura hierárquica de fácil manutenção e implementação;
- sistema de entrada e saída simplificado;
- oferece uma interface consistente para dispositivos periféricos;
- suporte à rede TCP/IP;
- interface gráfica amigável;
- possui muitos recursos de segurança para impedir que as informações sejam acidental ou propositadamente acessadas por usuários não autorizados.

2.3 Estrutura

A estrutura de um sistema operacional é definida pelo relacionamento existente entre seus componentes e pela forma como seu código é organizado.

A estrutura básica do UNIX (Figura 2.1) é a seguinte:

- ***Hardware*** – é a parte física do sistema, que é composta pelo processador, memória e os dispositivos de entrada/saída;
- ***Núcleo*** – é responsável por controlar o hardware permitindo que os programas de usuários tenham acesso aos serviços do sistema como criação

e gerência de processos, gerência de memória, sistema de arquivos e gerência de entrada/saída. O *shell* e outros programas utilizam esses recursos através de chamadas de sistema.

A estrutura do UNIX possibilita que o núcleo manipule o hardware diretamente. Normalmente para desenvolver um software aplicativo para UNIX, ou até mesmo um novo comando, não é necessário conhecer as características do hardware no qual o sistema funciona, mas apenas saber utilizar as chamadas do sistema operacional.

- ***Shell*** – é um programa que age como *interface* de comunicação entre o usuário e o sistema operacional, facilitando o acesso aos recursos e aos comandos disponíveis. É através do *shell* que o usuário faz a entrada dos comandos que vão ser executados pelo *UNIX*.

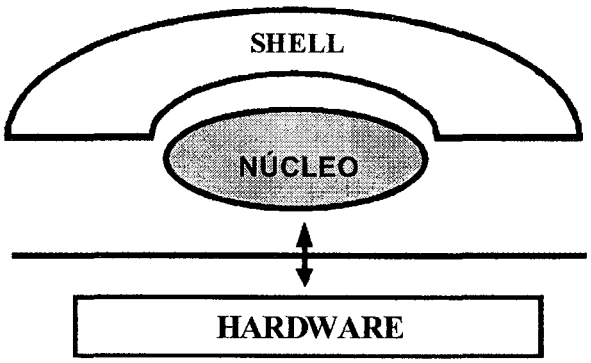

Figura 2.1 –Estrutura do *UNIX*

2.4 Processos

Um processo, também chamado de tarefa, é um programa em execução, incluindo os valores correntes de todos os registradores do hardware, e das variáveis manipuladas por ele no curso de sua execução [TAN, 1995]. Como o UNIX é um sistema multiprogramável vários processos podem estar sendo executados concorrentemente, através de um paralelismo aparente. Em sistemas grandes podem

existir até milhares de processos ativos e, dependendo da situação, os processos mudam de estado durante a sua execução. Os estados possíveis são:

- Executando: quando o processo está sendo executado pelo processador;
- Pronto: quando o processo está preparado e esperando que o processador seja alocado para executá-lo;
- Esperando ou bloqueado: quando o processo está aguardando um evento externo ou um determinado recurso para prosseguir o seu processamento;
- Zumbi: quando o processo encerrou sua execução, mas ainda possui entrada na tabela de processos.

Para o núcleo do UNIX um processo é uma entrada na tabela de processos. Cada processo possui o identificador do processo (PID), que é um número inteiro atribuído a ele pelo sistema operacional, e um identificador de usuário (UID), que é um número inteiro atribuído a cada usuário pelo administrador do sistema e que o processo adquire automaticamente no momento de sua criação.

Em determinados momentos os processos precisam se comunicar com o núcleo e também com outros processos para executar as suas atividades, para isso eles utilizam os mecanismos de comunicação que são conhecidos como IPC (*Inter Process Communication*). Esses mecanismos podem ser:

- Típicos do UNIX: sinais e pipes;
- Mecanismos do System V: memória compartilhada, semáforos e fila de mensagens;
- Mecanismos introduzido pelo BSD 4.2: sockets.

2.5 Sistema de Arquivos

O sistema de arquivos contém os diretórios, arquivos de sistema e arquivo de usuários. A sua estrutura hierárquica é de fácil implementação e manutenção. Fazendo

uma analogia, pode–se afirmar que essa estrutura é organizada como uma árvore (Figura 2.2).

No diretório raiz (/), que é o diretório pai (principal) do sistema de arquivos, podem existir arquivos propriamente ditos e outros diretórios, neste caso chamados de subdiretórios. Não existe um limite máximo de níveis, exceto pelo fato de que pode haver alguma restrição imposta pelo hardware ou que um número excessivo de níveis pode comprometer o desempenho do sistema.

No UNIX, existem 3 tipos de arquivos:

- Arquivo comum
- Arquivo tipo diretório
- Arquivos especiais vinculados aos periféricos

Um diretório nada mais é que um arquivo contendo uma lista de subditretórios e arquivos que podem ser alcançados através dele, ou seja, ele apenas guarda informações sobre outros arquivos e diretórios hierarquicamente subordinados a ele.

A segurança de acesso a arquivos e diretórios é provida através de um esquema de posse de arquivos e de permissões de acesso. Os arquivos do *UNIX* são sujeitos as permissões para leitura, escrita e execução.

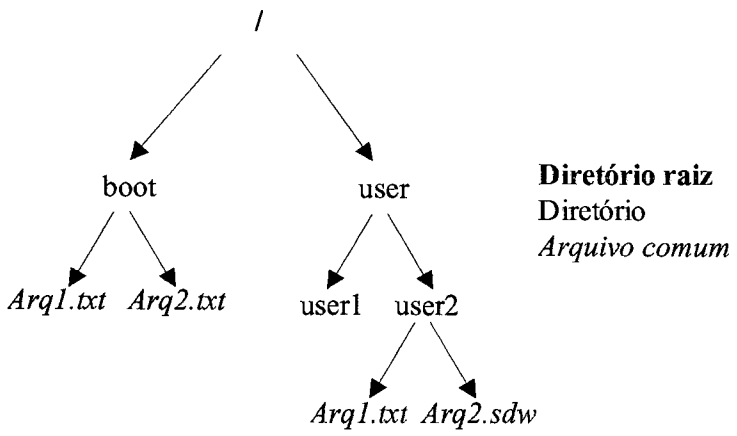

Figura 2.2 –Árvore de arquivos e diretórios do UNIX

2.6 LINUX

O LINUX é um sistema operacional, de código aberto que foi originalmente escrito por Linus Torvalds como um projeto pessoal, inspirado no MINIX, uma pequena versão do UNIX criada por Andrew Tannembaum . Depois de algum tempo trabalhando sozinho Torvalds solicitou ajuda a outros programadores que trabalharam através da Internet como voluntários. Atualmente o LINUX é desenvolvido por milhares de pessoas espalhadas pelo mundo.

Mesmo sendo considerado um clone do UNIX, pois possui as suas principais características e segue o padrão POSIX, o LINUX não tem origem no mesmo código fonte, tendo sido rescrito totalmente do nada. Ele possui todas as características do UNIX que foram apresentadas anteriormente nesse trabalho, por isso elas não serão listadas novamente.

Ao contrário do UNIX, o LINUX é um sistema aberto, permitindo que o usuário tenha acesso ao seu código fonte, podendo fazer alterações no núcleo (*kernel*), sendo fácil substituir módulos particulares do código por outros mais interessantes para o sistema ou para as aplicações. Assim, qualquer pessoa pode ver como o sistema funciona, corrigir seus problemas e fornecer alguma melhoria. Esses são alguns motivos de seu rápido crescimento, do aumento da compatibilidade de periféricos e de sua estabilidade.

Devido a isso, surgiram várias distribuições do LINUX, já que vários grupos de desenvolvedores passaram a organizar os arquivos e programas da maneira que eles achavam mais conveniente e que resolviam os seus problemas. Cada distribuição pode fazer qualquer alteração que achar importante, desde que respeite as condições impostas pela GPL *(General Public License)*.

Algumas distribuições que utilizam o sistema operacional LINUX são: *Slackware, Debian, Red Hat, Conectiva e Suse.* Elas estão disponíveis gratuitamente na Internet e são uma alternativa de diminuição de custos em relação aos sistemas UNIX existentes.

3 Redes de Computadores

Neste capítulo, são apresentados alguns aspectos de redes de computadores e o protocolo de comunicação TCP/IP, sendo dividido em 3 subcapítulos. O subcapítulo 3.1 descreve a classificação das redes de computadores quanto a sua disposição geográfica. O subcapítulo 3.2 apresenta a topologia das redes de computadores, descrevendo as organizações topológicas mais comuns e mostrando uma tabela com um paralelo entre as mesmas. O subcapítulo 3.3 descreve o conjunto de protocolos TCP/IP, explicando seus principais protocolos e a comunicação através de *sockets*.

3.1 Classificação das Redes de Computadores

O termo redes de computadores se aplica a um grupo de computadores autônomos interconectados, com o objetivo de compartilhar recursos e trocar informações entre si.

Uma forma comum de classificar as redes de computadores é a baseada em sua distribuição geográfica, sendo divididas nas seguintes categorias:

- **Redes Locais:** Também chamadas de LANs (*Local Area Network*), são redes com abrangência geográfica limitada a uma organização, de forma a respeitar o limite de alguns quilômetros [CHI, 1999]. Qualquer rede que exista dentro de um único prédio, ou mesmo em um grupo de prédios adjacentes, é considerada uma rede local. As LANs têm um tamanho restrito, o que significa que o pior tempo de transmissão é limitado e conhecido com a devida antecedência [TAN, 1997].
- **Redes Metropolitanas:** Também chamadas de MANs (*Metropolitan Area Networks)*, são redes de alcance médio, por exemplo, com computadores interligados espalhados por uma cidade de médio porte [CHI, 1999]. Esta classificação é pouco utilizada.
- **Redes de Longo Alcance**: Também chamadas de WANS *(Wide Area Network)*, são redes que não possuem limitação de distância, consistem

geralmente na conexão de duas ou mais redes locais distribuídas em uma grande área geográfica, tais como uma cidade, um país ou um continente. Em geral são conectadas por meio de ligações oferecidas por empresas provedoras de serviços de telecomunicação [CHI, 1999].

3.2 Topologia das Redes de Computadores

A topologia descreve o modo pelo qual os componentes de uma rede de computadores estão conectados fisicamente entre si [CHI, 1999]. As organizações topológicas mais comuns são barramento, topologia em estrela e anel [GOI, 2002].

3.2.1 Topologia em barramento

A topologia em barramento se caracteriza pela ligação dos computadores a um mesmo meio de transmissão (figura 3.1). Como a comunicação de todos os componentes é realizada por um meio físico de comunicação único, é necessária uma identificação de endereço único para cada dispositivo da rede.

As informações circulam na rede em forma de pacotes que são recebidos por todos os componentes do barramento, que o descartam caso este não seja destinado ao seu endereço.

A dependência de um único meio de transmissão, estabelece o risco de uma falha interromper toda a comunicação no barramento [GUI, 2002]. Além disso, todo o tráfego em um mesmo canal de comunicação pode sofrer colisões que precisam ser tratadas.

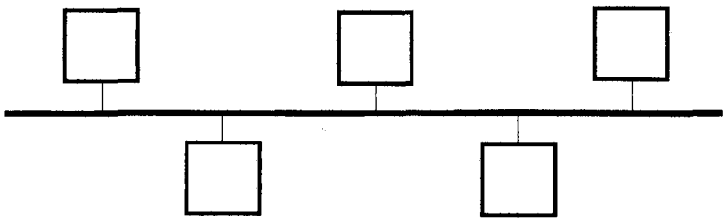

Figura 3.1 –Topologia em barramento.

3.2.2 Topologia em Anel

A topologia em anel conecta os componentes da rede em uma cadeia circular, onde cada um deles é conectado ao seguinte até o último componente ser conectado ao primeiro, fechando assim o anel (figura 3.2). Cada dispositivo da rede é identificado por um endereço único, sendo que cada um examina os dados que estão trafegando pelo anel. Caso esses dados não sejam endereçados ao componente que os está examinando são endereçados ao seguinte até que alcencem o seu destino. O fluxo de informações em redes com topologia em anel pode ser bidirecional ou unidirecional.

As redes com topologia em anel podem atingir grandes distâncias, no entanto existe uma limitação prática do número de estações em um anel. Este limite é devido aos problemas de manutenção e confiabilidade que são algumas das desvantagens dessa topologia.

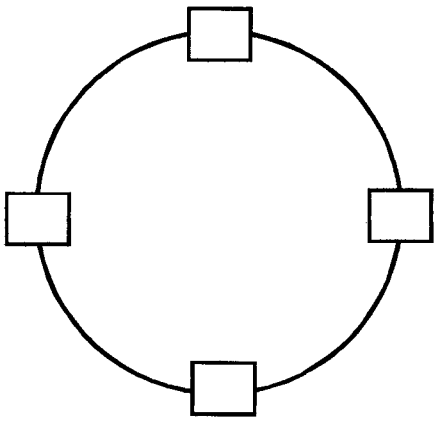

Figura 3.2 –Topologia em anel.

3.2.3 Topologia em estrela

Na topologia em estrela cada componente da rede é interligado a um dispositivo central, através do qual todas as mensagens trafegam (figura 3.3). Podem existir ligações lógicas ponto a ponto ou multiponto, dependendo do periférico concentrador utilizado, por exemplo, *hub* ou *switch*. A topologia estrela multiponto é

um pouco mais complexa que a topologia em estrela ponto a ponto, pois neste caso existem múltiplos dispositivos de ligação centrais [GOI, 2002].

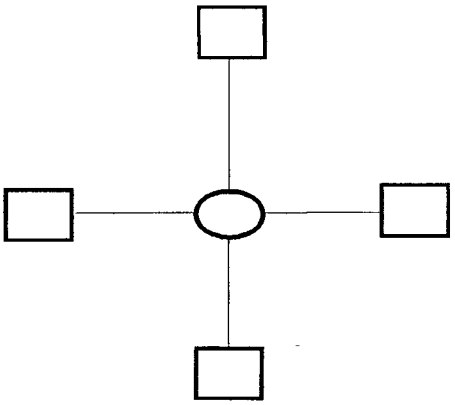

Figura 3.3 –Topologia em estrela.

3.2.4 Tabela Comparativa das Topologias de Rede

A seguir é apresentada uma tabela comparativa que faz um paralelo entre as três topologias de organização de redes apresentadas (tabela 3.1).

Tabela 3.1 –Comparação entre as topologias de rede.

Topologia	***Vantagens***	***Desvantagens***
Barramento	Fácil instalação Baixo custo Fácil acrescentar novos nós	Difícil isolar os problemas
Anel	Baixo custo Desempenho uniforme	Difícil isolar os problemas Difícil acrescentar novos nós
Estrela	Facilidade para detectar falhas Monitoramento centralizado	Alto custo

3.3 O Conjunto de protocolos TCP/IP

Protocolo é um conjunto de regras que estabelece como os dados serão transmitidos na rede [ERI, 2002]. Padrões de protocolos são aceitos por equipamentos de comunicações de diferentes fabricantes [GUI, 2002]. Existem vários protocolos, mas no texto será abordado apenas o conjunto de protocolos TCP/IP por ser relevante na realização deste trabalho.

O modelo TCP/IP abrange um conjunto de protocolos, no qual os principais são o TCP e o IP, por isso o nome. Ele tem por objetivo estabelecer a transferência de dados entre dois ou mais computadores interligados.

Os protocolos que fazem parte do TCP/IP atuam em camada (Figura 3.4):

- **Camada de Aplicação:** contém os protocolos de alto nível como, o HTTP, FTP, TELNET, SMTP, entre outros.
- **Camada de Transporte:** contém os protocolos de transmissão, como o UDP e o TCP.
- **Camada de Rede:** contém protocolos de conexão, como: IP, ICMP, ARP e o RARP.
- **Camada de Sub-Rede:** contém os *gateways* e roteadores. É responsável pelo enlace de dados entre as diversas redes conectadas.

Figura 3.4 – Camadas do TCP/IP.

As camadas do TCP/IP trocam informações verticalmente de forma hierárquica, onde cada camada geralmente interage com os protocolos das camadas imediatamente superior e inferior, ou seja, o TCP/IP funciona como uma pilha de protocolos.

3.3.1 Protocolo IP

O IP (*Internet Protocol*) define um mecanismo de entrega de pacotes não confiável e sem conexão [COM, 1995]. Ele provê três ítens importantes:

- A unidade básica de dados a ser transferida na rede;
- O software de IP executa a função de roteamento, escolhendo um caminho sobre o qual os dados serão enviados;
- Um conjunto de regras que envolvem a idéia da expedição de pacotes não confiáveis. Estas regras indicam como as máquinas poderiam processar os pacotes, como e quando as mensagens de erros poderiam ser geradas, e as condições em que os pacotes podem ser descartados.

Na transferência de dados, o pacote recebido da camada superior é dividido em pacotes chamados datagramas, que são enviados para a camada de interface com a rede para transmissão. O datagrama IP foi definido para a transmissão de dados em uma rede de comutação de pacotes, que é uma técnica utilizada para melhorar o desempenho da rede.

3.3.2 Protocolo ICMP

O ICMP (*Internet Control Message Protocol*) é utilizado para transmissão de mensagens de controle ou de erro entre os programas IP das máquinas envolvidas na troca de dados. Geralmente o ICMP é considerado integrante da camada IP, mas ele apenas utiliza os serviços dessa camada.

Na maioria das vezes as mensagens ICMP são geradas pelos roteadores, mas podem ser geradas pela máquina destino. Quando ocorre um problema com a transmissão de dados, o ICMP apenas *Pentium* uma mensagem de erro para o IP da máquina de origem, que deve informar o erro para um programa de aplicação individual ou tomar alguma providência para corrigir o problema [COM, 1995].

3.3.3 Protocolo UDP

O UDP (*User Datagram Protocol)* provê um serviço de entrega de datagramas não confiável e sem conexão. Isso significa que o serviço oferecido pelo UDP:

- não garante que o datagrama seja entregue ao destino;
- não guarda um registro dos datagramas enviados;
- não evita a duplicação de mensagens;
- não faz controle de fluxo;
- não garante a ordenação dos dados;
- não controla o congestionamento da rede.

O UDP utiliza o IP para transportar as mensagens, mas adiciona a capacidade de distinção entre múltiplos destinos dentro de um dado computador [COM, 1995], fornecendo a cada mensagem um número de porta origem e um número de porta destino que possibilitam a multiplexagem e demultiplexagem dos dados. Isto é, o UDP aceita datagramas de várias aplicações diferentes e os *Pentium* à camada IP para transmissão. Por outro lado, aceita vários datagramas UDP que chegam da camada IP e os encaminha para o programa de aplicação apropriado. Além disso, o UDP possibilita a verificação de que os dados chegaram intactos ao destino, através do cálculo do *ch*ecksum.

Muitos programas de aplicações que utilizam o UDP trabalham bem em um ambiente local, já que a sua simplicidade diminui o *overhead* na rede tornando o

transporte de mensagem mais fácil e rápido, mas falham dramaticamente quando utilizados em grandes redes TCP/IP [COM, 1995]. Assim, quando necessário, esses programas de aplicação é que devem se preocupar com a confiabilidade, tratando de problemas como perda e duplicação de mensagem, atraso, ordenamento de bytes e perda da conectividade.

Datagramas UDP

Cada datagrama UDP é composto de um cabeçalho e de uma área de dados (Figura 3.5)

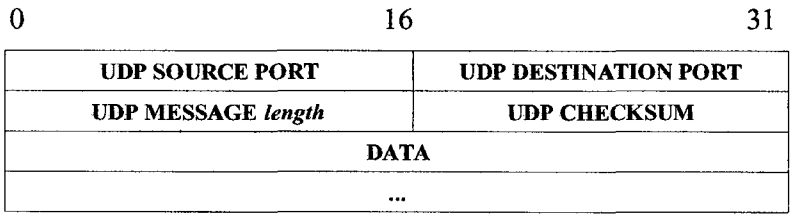

Figura 3.5 –Datagrama UDP.

Como se encontram numa camada superior à do protocolo IP, os datagramas UDP são encapsulados num datagrama IP para serem enviados pela rede. Por sua vez, os datagramas IP são encapsulados numa estrutura dependente do tipo de rede à qual a máquina destino está ligada.

Cada programa de aplicação deve negociar com o Sistema Operacional um número de porta antes de enviar o datagrama. Uma vez que a porta tenha sido atribuída, qualquer datagrama que o programa de aplicação envie terá seu número especificado no campo Porta Origem (*UDP SOURCE PORT*).

As mensagens que chegam e possuem um número de porta destino que está correntemente em uso são armazenadas na fila correspondente a esta porta. Caso contrário, o UDP envia uma mensagem de erro ICMP do tipo porta inalcansável e descarta o pacote. Se a fila estiver cheia, mesmo que o número da porta seja válido, ocorre um erro e o datagrama é descartado.

3.3.4 Protocolo TCP

O TCP (*Transmission Control Protocol*) foi projetado para oferecer fluxo de bytes ponto a ponto confiável [TAN, 1997], que se adapta dinamicamente às propriedades da rede. Como a camada IP não fornece nenhuma garantia na entrega dos datagramas, cabe ao TCP prevenir ou solucionar as falhas que possam ocorrer. O serviço oferecido pelo TCP para uma aplicação pode ser caracterizado pelas seguintes propriedades:

- O TCP fornece multiplexação e demultiplexação de mensagens, obtida através do mecanismo de portas, assim como no UDP;
- O TCP fornece conexões entre as aplicações envolvidas na transferência de dados. Conceitualmente, uma aplicação faz uma chamada que deve ser aceita por outra aplicação [COM, 1995] e tanto a enviante quanto a recebedora devem afirmar que desejam trocar informações. Quando as duas aplicações estão prontas para se comunicar são estabelecidos os pontos finais da comunicação criando um *socket* TCP, que é um canal lógico que possibilita a transmissão de dados entre os dois pontos.
- O TCP fornece confiabilidade. Quando o TCP envia dados a outro ponto espera um reconhecimento em retorno. Caso não receba uma mensagem de reconhecimento o TCP automaticamente retransmite os dados e espera por um maior tempo [STE, 1998]. Depois de um determinado tempo, geralmente 4 a 10 minutos dependendo da implementação, o TCP abandona as tentativas de retransmissão. Para cada conexão o TCP mantém uma variável RTT (*round–trip time*), que é a melhor estimativa no momento para o cálculo de ida e volta em relação ao destino em questão [TAN, 1997], de forma que se saiba o quanto esperar por um reconhecimento.
- O TCP fornece ordenamento dos dados associando um número de seqüência a todos os *bytes* que são enviados. Portanto, se o segmento chegar fora de ordem poderá ser reordenado. Se os dados forem duplicados, o TCP poderá detectar esse problema e descartar a duplicata [STE, 1998].

- O TCP fornece controle de fluxo, também chamado de janela anunciada. Cada reconhecimento contém a informação com a quantidade de dados que o ponto está apto a receber [COM, 1995], ou seja, informa o espaço livre no *buffer* corrente do recebedor. O tamanho da janela é alterado dinamicamente durante a execução e quando o *buffer* estiver cheio a aplicação deve ler os dados armazenados antes que possa receber mais dados.
- O TCP fornece conexão *full–duplex*. O TCP mantém informações, como número de seqüência e tamanho da janela, para cada direção do fluxo de dados possibilitando que as aplicações enviem e recebam dados ao mesmo tempo em uma conexão, ou seja, fornece funcionalidade para fluxos de dados concorrentes.

Segmento TCP

A unidade de transferência utilizada pelo TCP é chamada de segmento. Segmentos são trocados para estabelecer conexões, transferir dados, enviar reconhecimentos, indicar tamanho de janela e fechar conexões [COM, 1995]. Cada segmento é formado pelo cabeçalho TCP seguido pela área de dados (figura 3.6).

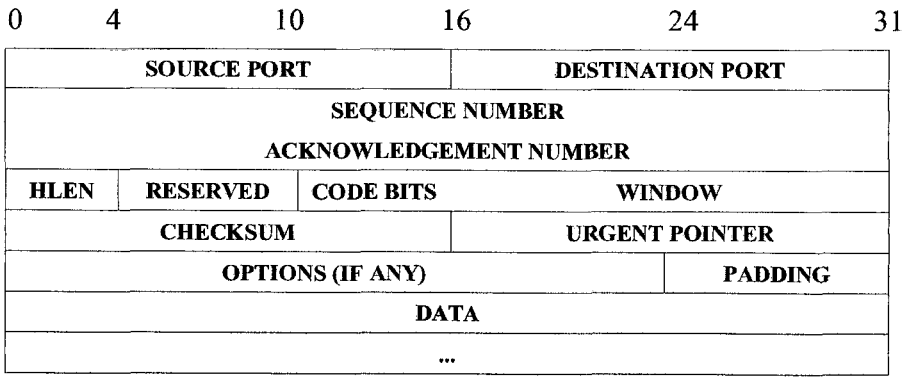

Figura 3.6 –Segmento TCP.

Estabelecendo uma conexão

Para estabelecer uma conexão, ou criar o *socket* entre os pontos, o TCP utiliza o *three-way handshake* (Figura 3.7). A conexão é iniciada quando um dos pontos, que será chamado de cliente, solicita a conexão realizando o *active open*. O outro ponto, que será chamado de servidor, deve estar preparado e esperando uma requisição de conexão, *passive open*. Os passos para o estabelecimento de uma conexão são os seguintes:

1. Inicialmente o cliente solicita a conexão e envia ao servidor um segmento de sincronização (SYN) com o número de seqüência inicial (j) para os dados que enviará na conexão.
2. O servidor responde enviando um segmento de reconhecimento (ACK) do SYN do cliente (ACK j+1) e o seu próprio segmento SYN com o número de seqüência inicial (k) para os dados que enviará na conexão.
3. O cliente envia ao servidor o reconhecimento do SYN recebido (ACK k+1). Neste momento a conexão está pronta e a troca de dados pode ser iniciada.

Normalmente não são enviados dados com o segmento SYN que contém apenas um cabeçalho IP, um cabeçalho TCP e possíveis opções TCP [COM, 1995].

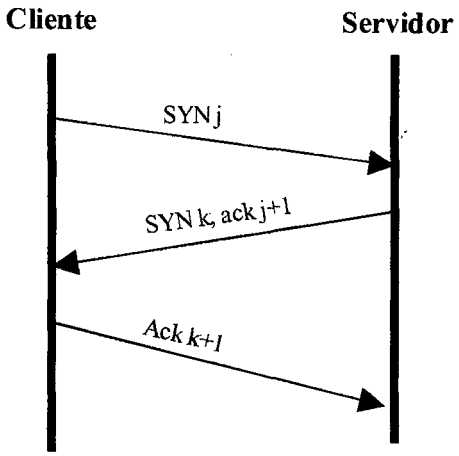

Figura 3.7 –Estabelecendo Conexão TCP.

É possível que dois pontos solicitem conexão simultaneamente entre o mesmo par de *sockets*. Neste caso será estabelecida apenas uma conexão, pois as conexões são identificadas por suas extremidades [TAN, 1997].

Encerrando uma conexão

Enquanto são necessários três passos para estabelecer a conexão, para finalizá-la são necessários quatro (Figura 3.8):

1. Um dos lados executa o *active close* iniciando a seqüência de término da conexão, possivelmente com a chamada de sistema close(). Ele envia um segmento avisando que finalizou o envia de dados na conexão.
2. O lado que recebe o segmento FIN executa o *passive close*. Ele reconhece o recebimento enviando um ACK. O recibo do FIN é passado para a aplicação como um fim-de-arquivo, já que não será aceito mais enviar dado na conexão além dos que já estejam enfileirados.
3. Quando a aplicação que recebeu o fim-de-arquivo executa um *close()*, é enviado um FIN para o lado que iniciou o processo de término da conexão.
4. Quando o FIN é recebido um ACK é enviado como reconhecimento e a conexão é finalizada.

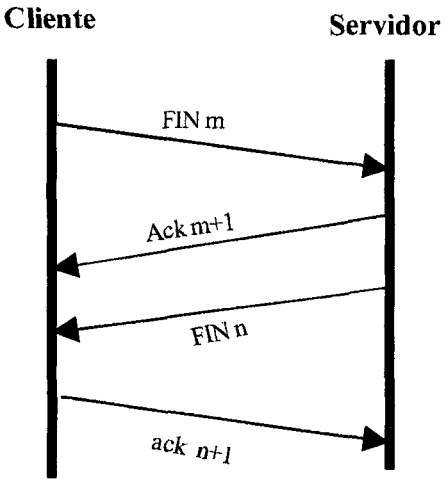

Figura 3.8 –Encerrando a conexão TCP.

Uma vez que o ACK final foi enviado, a conexão não poderá ser fechada e reutilizada por um período de 2 vezes o tempo máximo de vida do segmento. Esta restrição é imposta para o caso do ACK ser perdido. Se o ACK não for recebido nesse período de tempo o segmento FIN será enviado novamente.

3.3.5 Sockets

Um *socket* é uma extremidade de um canal de comunicação entre processos, permitindo a transmissão bidirecional dos dados. Para criar esse canal, cada processo deve criar o seu próprio *socket,* especificando o endereço do domínio e o tipo de *socket,* pois a comunicação só é possível entre *sockets* do mesmo tipo e no mesmo domínio. Existem dois endereços de domínio amplamente usados, o *UNIX domain*, para comunicação de processos que compartilham um sistema de arquivos comum, e o *internet domain*, no qual dois processos se comunicam através de uma rede.

Cada um dos domínios têm seu próprio formato de endereçamento. O endereço de um *UNIX domain socket* é uma *string*, que basicamente é uma entrada no sistema de arquivos. Um endereço de um *internet domain socket* é composto por um par formado pelo número IP e o número da porta de serviço do protocolo utilizado para a transmissão dos dados, sendo um par para o servidor e outro para o cliente [DUM, 1995].

Os *UNIX domain sockets* são uma alternativa aos mecanismos IPC descritos no capítulo 2, sendo uma boa solução para comunicação entre processos executando em uma mesma máquina. Geralmente esses *sockets* são até duas vezes mais rápidos que os *sockets TCP* [STE, 1998].

Os tipos de sockets do *internet domain* são os *Socket* UDP, *Socket* TCP e *Raw Socket.*

Sockets TCP

Os *sockets* TCP são mais conhecidos como *sockets stream, já que* os dados são transferidos como uma corrente de caracteres. Este *socket* é mais confiável, pois exige

que uma conexão seja estabelecida, através do *three–way handshake* (figura 3.7), para que transferência de dados possa ser iniciada. Esse canal fica aberto até que seja realizado um procedimento de desconexão (figura 3.8), que pode ser iniciado pelo cliente ou pelo servidor.

Eles são muito utilizados para aplicações que necessitam de confirmação de entrega dos dados, apesar do prejuízo na velocidade.

Sockets UDP

Nos sockets UDP os dados são transferidos em pacotes com tamanhos pré–definidos, por isso são mais conhecidos como *sockets datagram*. Eles são mais rápidos e menos seguros, pois não necessitam que uma conexão seja estabelecida antes que dois processos possam enviar e receber dados entre si. Já que não existe uma conexão, quando os dados são enviados pelo canal o processo origem não tem garantias de que os dados alcançarão o seu destino corretamente. As aplicações que utilizam os *sockets UDP*, devem conter alguma rotina que verifique e tome alguma providência caso ocorra problemas na transferência dos dados.

É amplamente usado em aplicações onde a rapidez na entrega é mais importante que a exatidão da mesma.

4 Máquinas Paralelas

Neste capítulo, são apresentados alguns aspectos das máquinas paralelas, sendo dividido em 3 subcapítulos. O subcapítulo 4.1 trata das máquinas paralelas, apresentando uma descrição básica das mesmas, uma possível classificação para a arquitetura de computadores e a classificação das máquinas paralelas segundo Flynn. O subcapítulo 4.2 faz uma breve descrição dos multicomputadores. O subcapítulo 4.3 apresenta uma visão geral do multicomputador Crux, descrevendo sua arquitetura e o seu funcionamento básico.

4.1 Classificação

As máquinas paralelas são formadas por nós processadores interligados fisicamente através de uma rede de interconexão. Elas são utilizadas com o objetivo de proporcionar grande poder computacional através da cooperação de processadores na execução de eventos concorrentes, oferecendo confiabilidade e uma boa relação custo/desempenho, suprindo a necessidade de um processamento intensivo.

Embora exista um princípio básico para a construção de máquinas paralelas, o projeto pode diferir em diversos pontos devido a diversidade de modelos propostos e dos critérios adotados. Essas possibilidades de construção, como mecanismo de acesso a memória, forma de controle das unidades de processamento e maneiras de conectar os componentes, dificultam uma classificação geral para esse tipo de máquina. Entretanto, na figura 4.1 é apresentada uma possível classificação para a arquitetura dos computadores [COR, 1999].

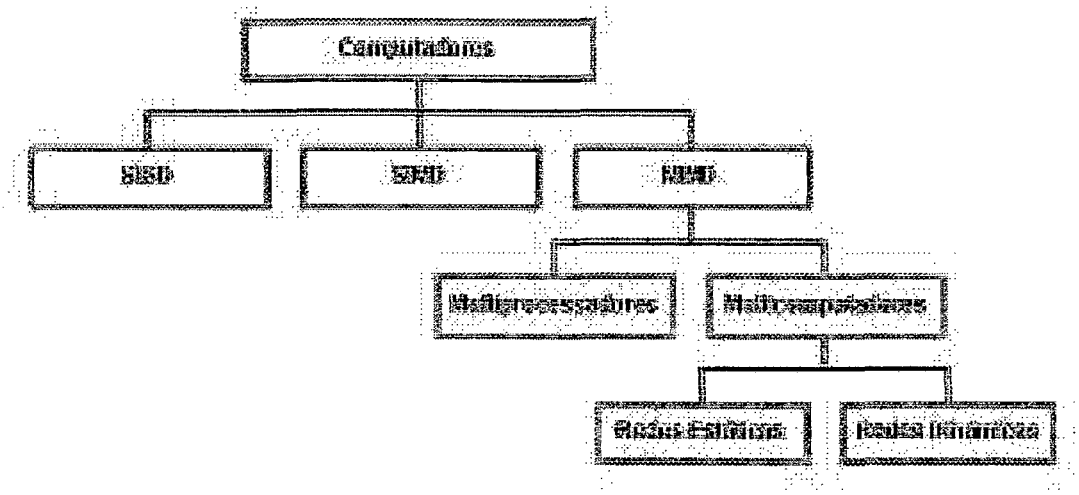

Figura 4.1 –Classificação geral para as arquiteturas de computadores.

A classificação de Flynn, que é a mais aceita e utilizada, se apoia no número de fluxos de instruções e no número de fluxos de dados para classificar um computador em quatro categorias distintas:

- **SISD (*Single Instruction, Single Data*)** – Nessa categoria as máquinas possuem uma unidade de processamento e uma unidade de controle. Um único programa é executado sobre seu dados. É o modelo mais utilizado em computadores comerciais.

- **SIMD (*Single Instruction, Multiple Data)*** – Essas máquinas possuem uma unidade de controle e múltiplas unidades de processamento. Um único programa opera simultaneamente sobre dados diferentes. É um modelo adequado para processamentos que repetem o mesmo cálculo para diferentes conjuntos de dados, como os cálculos matriciais e vetoriais.

- **MISD *(Multiple Instruction, Single Data)*** – Essas máquinas possuem uma unidade de processamento e múltiplas unidades de controle. Várias instruções operam simultaneamente sobre os mesmos dados. É um modelo de difícil aplicação prática.

- **MIMD** ***(Multiple Instruction, Multiple Data)*** – Essas máquinas possuem múltiplas unidades de controle e múltiplas unidades de processamento. Várias instruções independentes operam sobre seu próprio conjunto de dados. Esse é o modelo mais adequado para as aplicações de uso geral.

As máquinas paralelas de uso geral se enquadram na arquitetura MIMD. Nessa categoria se encaixam os multiprocessadores , que possuem um conjunto de memória compartilhado por todos os computadores, e os multicomputadores, que são compostos por nós ou elementos processadores com memória privativa.

Os multicomputadores constituem os ambientes físicos alvo desse trabalho.

4.2 Multicomputadores

Os multicomputadores são compostos de nós autônomos, cada um dos quais dispondo basicamente de um processador, de memória privativa e de suporte físico para a comunicação com os outros nós (figura 4.2) [COR, 1999].

Algumas características dos multicomputadores servem para diferenciá–los de outras máquinas paralelas e tornam a sua utilização bastante vantajosa:

- Grande número de nós homogêneos;
- Proximidade física dos nós;
- Comunicação através de troca de mensagens;
- Alto grau de paralelismo real;

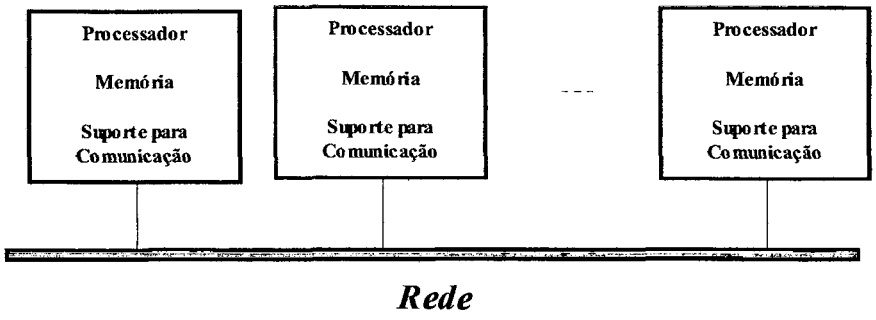

Figura 4.2 –Multicomputador.

A interação entre os nós de um multicomputador é alcançada por meio do envio de mensagens que trafegam por redes de interconexão de alta velocidade. As características das redes de interconexão, que representam o elemento determinante do desempenho do multicomputador, podem ser distinguidas de acordo com a natureza dessas redes:

- **Redes estáticas** – Esse tipo de rede (figura 4.3a) apresenta uma topologia fixa [CAM, 1995], onde as ligações entre os nós são estabelecidas na construção do sistema e não podem ser reconfiguradas. Os canais são permanentes.
- **Redes dinâmicas** – Essas redes (figura 4.3b) podem ser reconfiguradas através de circuitos de chaveamento eletrônico cuja manipulação permite a atribuição de canais físicos temporários que podem ser criados de acordo com as características de comunicação de um determinado problema.

As redes de interconexão dinâmicas importantes para este trabalho são o barramento e o *crossbar*.

Barramento

É um modelo muito flexível, já que permite adicionar nós sem efetuar grandes alterações no sistema. Esse tipo de rede não funciona muito bem com grande número de

nós devido a sua baixa capacidade de transferência. O barramento pode atingir uma eficiência aceitável se as comunicações através dele não forem freqüentes.

Crossbar

Um *crossbar* N x M possui N entrada dispostas verticalmente e N saídas horizontalmente, formando uma rede do tipo grelha, na qual a intersecção de duas linhas representa um comutador, que controla a conexão entre a entrada e a saída correspondentes. Um *crossbar* com N entradas e N saídas permite a conexão de até N pares de entrada/saída, onde cada entrada/saída pode participar de apenas uma conexão por vez. É uma rede muito poderosa, porém muito cara quando aumenta muito de tamanho.

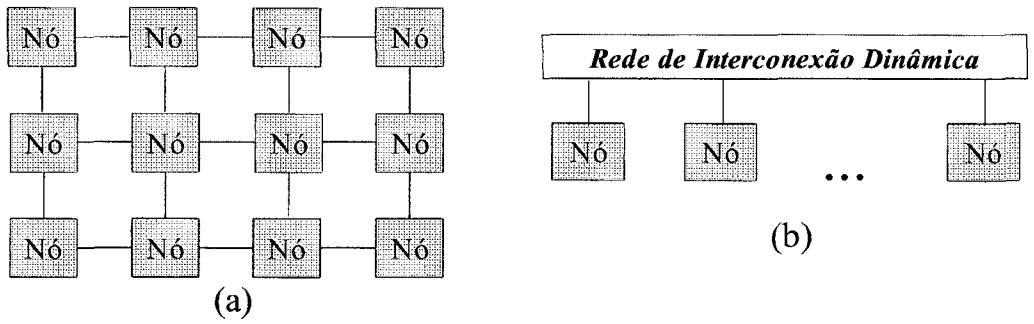

Figura 4.3 –Redes estáticas e redes dinâmicas.

4.3 Multicomputador Crux

O Multicomputador Crux (figura 4.4) é um mecanismo de conexão dinâmica de canais físicos diretos por demanda [COR, 1999], que possui quatro componentes em sua arquitetura:

- **Nós de Trabalho:** computadores que possuem um processador, memória privativa e um *hardware* de suporte à comunicação, que executam os processos de programas paralelos;

- **Rede de Trabalho:** um *crossbar* utilizado para transportar mensagens de tamanho arbitrário através de canais físicos diretos, que são criados pela rede de controle, entre pares de nós de trabalho;
- **Nó de Controle:** utilizado para a configuração da rede de trabalho, é responsável pela conexão e desconexão dos canais físicos diretos entre os nós de trabalho, em resposta a solicitação dos nós de trabalho;
- **Rede de Controle:** um barramento utilizado para transportar pequenas mensagens de controle entre os nós de trabalho e o nó de controle, para a gerência das conexões dos canais diretos.

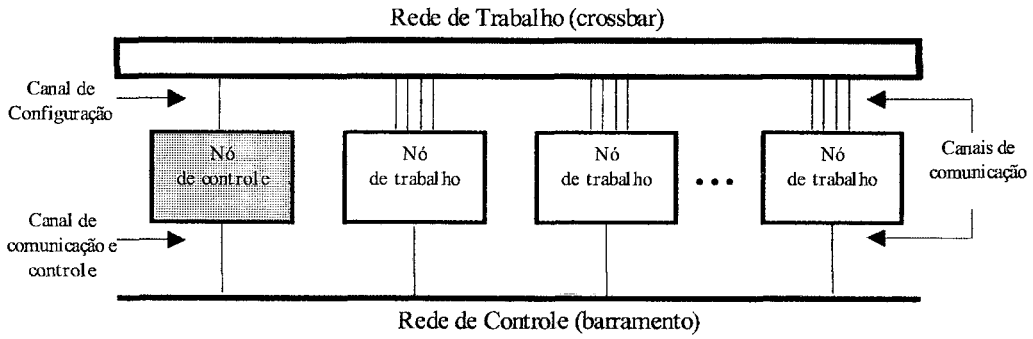

Figura 4.4 –Arquitetura do multicomputador Crux.

O Crux possui dois níveis básicos de comunicação: rede de trabalho (*crossbar*) e rede de controle (barramento). A arquitetura apresentada (figura 4.4) pode possuir um número expressivo de nós, cada um ligado à rede de trabalho por 4 canais de comunicação, podendo existir quatro canais físicos diretos conectados e trocando mensagens ao mesmo tempo. Além disso, possui um canal de comunicação com a rede de controle.

A rede de controle é utilizada para transporte de mensagens de controle e para resolver o problema de conexão de canais diretos. A rede de trabalho é configurada pelo nó de controle e é responsável pelo de transporte de mensagens entre os processos de programas paralelos.

As conexões dos canais físicos só ocorre se os dois nós envolvidos desejarem a conexão, ou seja, os dois lados devem estar dispostos a trocar mensagens. Os nós de trabalho se comunicam através da rede principal pelos canais físicos diretos. Antes do início da troca de mensagens é verificada a existência de um canal físico direto entre os nós trabalhadores. Caso exista um canal, a comunicação pode ser efetivada imediatamente; caso contrário, um pedido de conexão é enviado ao nó de controle, que cria os canais diretos que serão utilizados pelos nós de trabalho, enviando as mensagens de controle pela rede auxiliar. O nó de controle também recebe dos nós trabalhadores os pedidos de desconexão.

No Crux, cada nó de trabalho possui informações sobre as suas conexões, e o nó de controle, que deve estar permanentemente acessível aos demais nós, possui informação sobre a situação de todos os nós do multicomputador.

5 *Clusters* de Computadores

Neste capítulo, são apresentados alguns aspectos dos *clusters* de computadores, sendo dividido em 7 subcapítulos. O subcapítulo 5.1 apresenta uma visão geral dos *clusters* de computadores, descrevendo seus princípios básicos, sua arquitetura e forma de utilização. O subcapítulo 5.2 descreve as principais motivações para a utilização dos *clusters*, listando seus principais benefícios. O subcapítulo 5.3 lista os principais cuidados que devem ser levados em consideração ao planejar um *cluster*. O subcapítulo 5.4 descreve os sistemas operacionais, enfocando as características importantes para o funcionamento dos *clusters*. O subcapítulo 5.5 descreve algumas arquiteturas de rede de computadores, mais detalhademente a Myrinet, que é uma arquitetura de rede desenvolvida para a utilização em projetos de *clusters*. O subcapítulo 5.6 descreve como pode ser realizada a comunicação nos *clusters*. O subcapítulo 5.7 apresenta alguns projetos de *clusters* existentes e uma tabela comparativa entre esses.

5.1 Visão Geral dos C*lusters* de Computadores

Aplicações que requerem cada vez mais confiabilidade e desempenho estão exigindo estudos e investimentos em plataformas tolerantes a falhas e com grande poder computacional. Existem diversas arquiteturas que possibilitam a realização de processamento de alto desempenho [ROS, 2001]. Uma tendência de baixo custo é a utilização de *clusters* de computadores interconectados por uma rede local comum ou de alto desempenho [FER, 2001].

Os computadores que fazem parte do *cluster* geralmente são chamados de nós e trabalham forma integrada. Na maioria das vezes, o sistema possui um nó mestre, responsável pelo controle e pelo balanceamento da carga de trabalho entre os demais nós (Figura 5.1), assegurando a otimização da execução das tarefas.

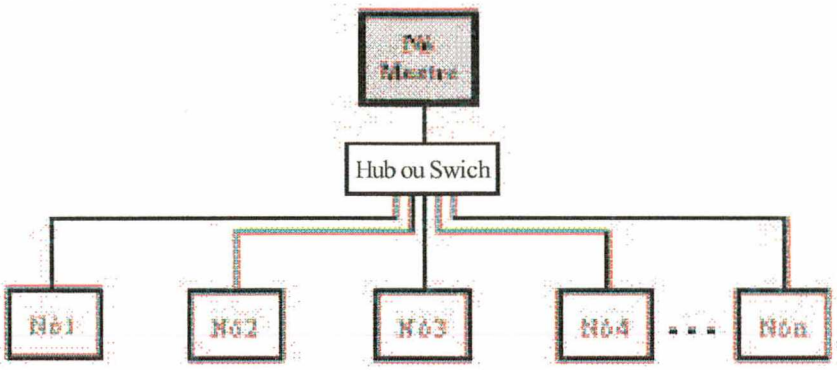

Figura 5.1 –Um modelo comum de *Cluster*.

Existem dois métodos de utilizaçao de *clusters*: alta disponibilidade e alto desempenho [CPW, 2001]. Os *clusters* de alta disponibilidade tem a finalidade de manter a continuidade do funcionamento, mesmo quando um determinado nó se torna inoperante. *clusters* de alto desempenho reúnem os nós para que trabalhem juntos na resolução de um mesmo problema, através da distribuição da carga de trabalho.

A utilização de *clusters* é uma boa alternativa para a execução de aplicações paralelas e distribuídas [BAR, 1999]. Ela é indicada para programas científicos, de engenharia, grandes sistemas com banco de dados robustos, sistemas que exigem alto grau de confiabilidade, servidores de rede com grande carga de trabalho, portais de grande acesso, *sites* que enviam milhões de *e–mails* diariamente e como um ambiente para pesquisas dentro da área de computação paralela [ZOM, 1996].

5.2 Motivações para Utilizar *Clusters*

Os *clusters* oferecem muitos benefícios sobre os outros tipos de ambientes computacionais de alto desempenho. Algumas das motivações para a utilização de *clusters* são:

- Os computadores que compõem os *clusters* podem ser heterogêneos [MEN, 2000], garantindo independência em relação a fabricantes.

- Nós podem ser acrescentados progressivamente de acordo com a necessidade, tornado os sistemas em *cluster* altamente escaláveis.
- A substituição de componentes é relativamente simples, sendo que o novo nó pode ser um computador diferente do que está sendo substituído.
- A utilização de máquinas comuns, que são produzidas em grande escala, resulta em um custo total bastante atrativo se comparado aos supercomputadores tradicionais.
- Máquinas antes ociosas podem ser ligadas ao *cluster* obtendo maior poder computacional com custos marginais [ZOM, 1996].
- Cada nó pode ser um sistema completo, permitindo seu reaproveitamento como uma máquina individual.

5.3 Cuidados ao Projetar um *Cluster*

Apesar das várias vantagens apresentadas anteriormente, fatores como o tipo de aplicação alvo e as tecnologias existentes devem ser levados em consideração na hora de planejar um *cluster*. Alguns aspectos que podem dificultar o trabalho e devem ser avaliados cuidadosamente são:

- Os equipamentos de rede não foram criados pensando em processamento paralelo.
- Existe pouco *software* que trata o *cluster* como uma máquina única.
- Algumas aplicações não podem ser paralelizadas, exigindo que as tarefas sejam executadas em série e de forma independente.

5.4 Sistemas Operacionais e *Clusters*

Os sistemas operacionais para *clusters* possuem a finalidade de escalonar a execução dos processos entre os componentes do *cluster*, prover abstrações para o software de alto nível, realizar sincronização e comunicação de processos, controlar o acesso de usuários e gerenciar os dispositivos da máquina.

Existem diversas características desejáveis em um sistema operacional para *clusters*, mas nota–se que nem sempre se consegue atingir todas elas, pois existem casos onde se consegue a melhoria em um certo aspecto em detrimento de outro [FER, 2001]. As principais características [CHA, 2000] são:

- O gerenciamento deve ser realizado de forma intuitiva, criando a visão de uma única máquina, ao invés de diversas máquinas interconectadas.
- A estabilidade deve ser garantida com tratamento contra falhas na execução de processos, recuperação após erro e execução segura de tarefas.
- A extensabilidade deve ser garantida por mecanismos de integração com novos dispositivos, através de *drivers* ou módulos que possam ser incorporados ao sistema operacional.
- A escalabilidade deve ser garantida com a capacidade de acréscimo de novos nós sem que o funcionamento do *cluster* seja afetado e sem que sejam necessárias grandes alterações de configuração.

A solução mais utilizada para obter as características acima é a utilização de um sistema operacional existente acrescido de uma camada intermediária, também denominada camada de controle.

O LINUX é o sistema operacional mais utilizado em *clusters* no momento [LIN, 2001] pois atende satisfatoriamente aos requisitos apresentados anteriormente, possuindo as principais características necessárias.

5.5 Arquiteturas de Rede para *Clusters*

Existem várias tecnologias de rede que podem ser adotadas na construção de um *cluster*. Abaixo são descritas algumas das muitas arquiteturas de rede existentes:

- ***Ethernet*** – É uma rede de transmissão de barramento, que permite controle descentralizado a velocidades de 10 ou 100 Mbits/s [TAN, 1997]. Durante anos, a *Ethernet* de 10 Mbits/s foi a tecnologia de rede dominante [PPG, 2001].

- ***Fast Ethernet*** – Várias tecnologias diferentes recebem o nome de *"Fast Ethernet"*, no entanto geralmente este termo se refere a tecnologia *Ethernet* com a velocidade de transmissão de 100 Mbits/s [PPG, 2001].

- ***ATM (Asyncronous Transfer Mode)*** – Foi projetada para prover uma interface de *software* com baixo *overhead*, para maior eficiência no gerenciamento de pequenos pacotes e em aplicações de tempo real [DIE, 1997]. Além disso, é uma rede com uma alta largura de banda que é suportada pelo LINUX.

- ***Gigabit Ethernet*** – É uma proposta recente na forma da norma IEEE 802.3z, que surgiu como uma possível alternativa ao emprego da ATM em redes que demandam grande largura de banda para a transmissão de dados. Esta tecnologia provê capacidade de 1 Gbps, tanto em transmissão *duplex* quanto semi–*duplex*, mediante o emprego do protocolo CSMA/CD [USP, 2001].

- **ARCNET (*Attached Resource Computer Network*)** – É uma rede local organizada fisicamente como barramento, que conecta até 255 nós [PPG, 2001]. Utiliza um protocolo baseado em *token* que estrutura a cadeia lógica como um anel. Possui desempenho mais consistente que a *Ethernet*, porém é mais cara.

- **HiPPI (*High Performance Parallel Interface*)** – Foi criada para prover uma alta largura de banda para transferência de grandes quantidades de dados [DIE, 1997]. É um padrão de rede ANSI que está sendo utilizado

com mais freqüência no mercado cliente–servidor de alta velocidade, utilizando comunicação paralela de 32 bits que pode transferir dados a uma taxa de até 1,6 Gbps, podendo percorrer dezenas de quilômetros. É suportada pela maioria dos supercomputadores e estações de trabalho de todos os barramentos padrão de mercado [STU, 1996].

- **FC (*Fibre Channel*)** – Sucessor do HiPPI, possui como estrutura básica um *crossbar* [TAN, 1997], formando canais de dados com conexões que podem ser estabelecidas para um único pacote ou permanecer por um longo período.
- **Myinet** – É uma rede local (LAN) para a construção de programas paralelos [SCH, 1999], desenvolvida pela empresa Myricon, visando formar uma tecnologia de interconexão baseada em chaveamento e comunicação por pacotes [OVE, 2001].

Das redes citadas acima a Myrinet foi desenvolvida para utilizaçao em projetos de *clusters* de computadores. Levando este fato em consideração, essa arquitetura é descrita com mais detalhe a seguir.

Myrinet

A myrinet foi desenvolvida com o objetivo de se tornar uma tecnologia de interconexão baseada em chaveamento e comunicação através de roteamento de pacotes [OVE, 2001]. Essa tecnologia é voltada principalmente para interconexão de *clusters* em uma rede local (LAN), com baixo custo e alto desempenho. As características que distinguem a Myrinet das demais redes são [OVE, 2001]:

- Portas e interfaces *full–duplex.*
- Controle de fluxo, de erro e monitoramento contínuo dos *links.*
- *switches* para aplicações de alta disponibilidade.
- Suporte a qualquer topologia.

- Estações possuem programa de controle para interagir diretamente com os processos.

Um *link* myrinet (figura 5.2) é formado por um par de canais *full−duplex* que são responsáveis pela transmissão de pacotes de tamanho variável e pelo controle do fluxo de informações [MYR, 2001]. Assim como em redes *ethernet*, podem ser transportados concorrentemente pacotes de vários tipos de protocolos, suportando diferentes interfaces de software, como o TCP/IP por exemplo. O roteamento dos pacotes ao longo da rede é feito através do seu cabeçalho [MYR, 2001].

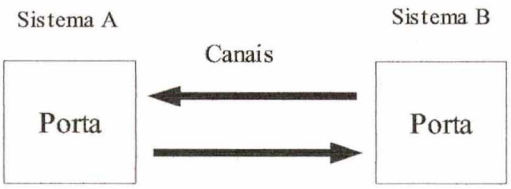

Figura 5.2: Representação de um *Link* Myrinet.

Os ítens necessários para a construção de uma rede myrinet são as interfaces dos equipamentos de *hardware*, *switches* para redes de maior porte, cabos, adaptadores e um *software* de suporte da empresa Myricon ou algum outro *software* que suporte redes Myrinet [MYR, 2001]. O software desenvolvido pela Myricon é baseado na troca de mensagens e está disponível em várias plataformas, como por exemplo em várias distribuições do LINUX.

A myrinet é muito útil para redes de grande escala viabilizando a construção de praticamente todos os tipos de topologia de rede. Mesmo assim podem ser configuradas redes pequenas, sendo que a menor topologia possível é composta por apenas duas estações conectadas diretamente.

O desempenho das soluções baseadas em Myrinet depende muito do *software* que é utilizado sobre a camada de rede [MYR, 2001]. Podem ser utilizados quaisquer tipos de *software* de interface, mas o ideal é a utilização de um *software* de interface

otimizado e projetado especialmente para Myrinet, o que permite alcançar resultados bastante satisfatórios.

5.6 Comunicação em *Clusters*

A comunicação de baixo–nível pode ser feita através de *sockets, device drivers* ou bibliotecas a nível de usuário [DIE, 1997], sendo que os *sockets* UDP e TCP são os mais utilizados. A interface de *software* padrão do UNIX para transporte de dados na rede é uma parte do núcleo denominada *device driver*.

Dois métodos comumente utilizados para troca de informações entre os nós são troca de mensagens e memória compartilhada distribuída. Quando o objetivo é a execução de aplicações que utilizam processamento paralelo é necessária a utilização de bibliotecas específicas para essa finalidade. Existem três padrões que estão bem consolidados nessa área [MEN, 2000]:

- **PVM (*Parallel Virtual Machine*)** – Consiste em um pacote de *software* que possibilita que *clusters* heterogêneos se comportem como se fosse uma única máquina virtual paralela, utilizando os recursos de maneira simultânea. É uma biblioteca padronizada que permite a utilização de máquinas de diversos modelos, desde pequenos computadores pessoais até supercomputadores. É utilizada para programação paralela através da troca de mensagens [SUN, 2001].

- **MPI (*Message Passing Interface*)** – Foi criado baseado em modelos de diversas empresas, como IBM e Intel, visando ser um padrão flexível, portável e prático para implementação de programação paralela por troca de mensagens. Duas implementações especiais do MPI são LAM, mantido na Universidade de Notre Dame, e MPICH, desenvolvido no Argonne National Laboratory e pela Universidade do Estado do Mississippi.

- **CORBA (*Common Object request Broker Architecture*)** – É um padrão utilizado para troca de mensagens entre objetos na rede [FAG, 2001]. Foi proposto pela OMG (*Object Management Group*) para permitir que

aplicações se comuniquem entre si independentemente da plataforma utilizada. Os serviços do CORBA são oferecidos a nível de sistema para comunicação entre aplicações orientadas a objetos, sendo que os objetos envolvidos na comunicação podem ser escritos em diferentes linguagens.

5.7 Projetos de *Clusters*

Atualmente existem vários projetos envolvendo *clusters*, criados para diversas finalidades, como, por exemplo, serem recurso de pesquisa em universidades e ambientes para execução de programas paralelos. Nesta seção serão apresentados alguns deles, enfocando a estrutura básica e o objetivo de cada um.

5.7.1 *Cluster* Beowulf

O modelo de arquitetura baseado em *clusters* é representado principalmente pelo *cluster Beowulf*, que foi desenvolvido inicialmente pela NASA, no Centro Espacial Goddard em Greendbelt, Maryland, no ano de 1994 [KLA, 2001]. Ele foi construído para manipular as informações recebidas pelos satélites, visando baixo custo e alto desempenho através da utilização de recursos disponíveis no mercado. Atualmente existem vários *clusters* do tipo *beowulf*, com aplicações e configurações variadas dependendo do tamanho e finalidade, mas seguindo o mesmo princípio de funcionamento.

Hardware

O *cluster Beowulf* é normalmente formado por um ou mais nós principais e por nós secundários conectados por uma rede local de alta velocidade, como por exemplo *FastEthernet* ou Myrinet. Cada nó secundário é uma unidade independente que pode ser substituída, ou retirada do *cluster*, sem afetar a disponibilidade do *cluster* como um todo.

Em muitos casos, os nós secundários não precisam ser computadores completos, dispensando disco, monitor, teclado, *mouse* e outros periféricos. Assim sendo, eles são dependentes dos nós principais, que os gerenciam e controlam remotamente, agindo como servidores de arquivos e como portas de acesso para os usuários [PIT, 2001]. O grau de dependência dos nós secundários em relação aos nós principais contribui para o caráter paralelo ou distribuído do *cluster*. Em aplicações paralelas, todos os nós secundários podem ser configurados pelo nó principal, trabalhando como uma uma máquina única. Em aplicações distribuídas, cada tarefa pode ser executada por apenas um nó secundário e o nó principal guarda a informação final, fazendo com que o *cluster* execute como um sistema de diferentes máquinas e não como uma única máquina virtual [KLA, 2001].

A configuração dos computadores que compõem o *cluster* variam em relação ao tipo e à quantidade de processadores utilizados, ao tamanho e velocidade da memória e da rede de interconexão. Essa configuração deve ser definida de acordo com o desempenho desejado para as aplicações alvo [PIT, 2001].

Software

O sistema operacional utilizado por um *cluster* do tipo *Beowulf* geralmente é o LINUX. Outros sistemas operacionais como o *Windows 2000* e o *Windows NT* tiraram vantagens das suas características para a construção de *clusters Beowulf* do tipo *Windows NT* [PIT, 2001].

Os nós são interligados de maneira a simular o comportamento de um supercomputador paralelo. O princípio do funcionamento consiste na divisão das tarefas em partes independentes, para que as informações sejam distribuídas entre os vários nós. A coordenação das tarefas é feita por transmissão de mensagens de um nó para outro utilizando bibliotecas específicas, como por exemplo PVM e MPI.

Uma configuração de duas ou mais máquinas rodando LINUX em rede, compartilhando pelo menos uma estrutura de diretórios via NFS, podendo executar um *shell* remoto (rsh) e possuindo bibliotecas de PVM ou MPI, já é considerada um *cluster Beowulf* [KLA, 2001].

As áreas de aplicação de um *cluster Beowulf* são muito diversificadas, sendo que é uma arquitetura que pode ser utilizada por grupos de pesquisa, instituições de ensino ou por empresas comerciais. É uma arquitetura interessante para qualquer aplicação que exija alto desempenho, sem custos excessivos ou para qual seja vantajosa a utilização de computação paralela, como, por exemplo, a realização de cálculos muito complexos.

5.7.2 PARNASS 2

O Parnass 2 foi desenvolvido para ser utilizado em pesquisas no departamento de matemática aplicada da Universidade de Bonn, Alemanha. O projeto consiste de um *cluster* dedicado de computadores pessoais, agindo como um supercomputador para realizar tarefas computacionais que são muito pesadas para estações de trabalho e servidores convencionais. O *cluster* utiliza componentes comerciais, como processadores *Pentium* II, placas de rede *Fast Ethernet* e o sistema operacional LINUX, o que o torna uma solução com uma boa relação custo/desempenho, permitindo que ele concorra de maneira surpreendente com sistemas 10 a 100 vezes mais caro [SCH, 1999].

Hardware

O Parnass2 passou por algumas alterações na sua estrutura desde que foi iniciado o projeto [PAR, 2000]. A composição física corrente é formada por 72 nós com dois processadores *Pentium* II e com 256 Mbytes de memória RAM. Cada nó possui adaptadores de rede *Fast Ethernet* ou Myrinet.

A rede de interconexão principal é a Myrinet, configurada como uma rede local com velocidade de 1,28 Gbps. Adicionalmente, os nós são conectados por rede *Fast Ethernet*. Os componentes físicos principais da rede são: 3 *switches Fast Ethernet switches* e 11 *switches Myrinet.* Os adaptadores de rede estão ligados por uma topologia de rede em três níveis, com largura de banda de 82 Gbps (figura 5.3).

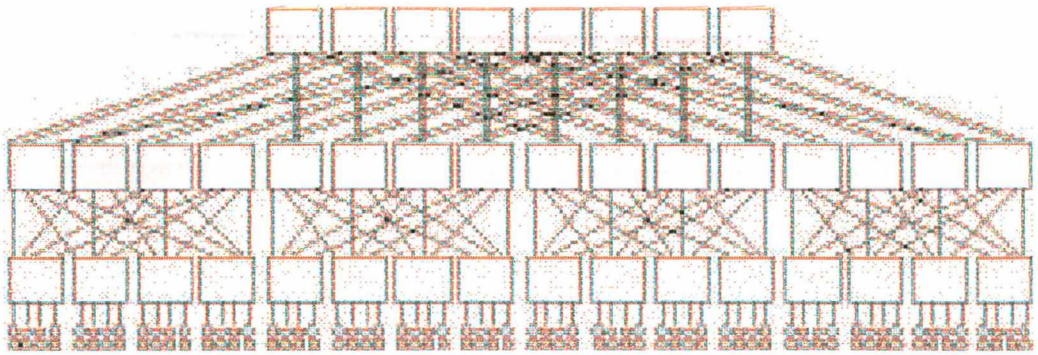

Figura 5.3: Topologia do cluster Parnass2 com rede em três níveis.

O Parnass2 possui um desempenho de 51,2 Gflops, um total de 27 Gbytes de memória principal e por volta de 560 Gbytes de disco rígido [PAR, 2001]. O objetivo é alcançar 32 Gbytes de memória principal [SCH, 1999].

Software

O Parnass2 utiliza o sistema operacional LINUX. O principal ambiente para troca de mensagens é o Score 2.4/MPICH−PM, que atualmente é o sistema de troca de mensagens mais rápido para Myrinet, explorando a comunicação rápida entre processos dentro de um nó multiprocessado.

O principal objetivo do Parnass2 é ser uma plataforma para execução de cálculos científicos que exigem alto desempenho. Muitas aplicações para solução de equações vem sendo executadas atualmente e existem várias em desenvolvimento [SCH, 1999].

5.7.3 TORNADO

O Tornado é um *cluster* de estações de trabalho da Universidade do Texas, que utiliza computadores pessoais e estações de trabalho comuns. É uma ferramenta educacional e de pesquisa utilizada por estudantes, como suporte ao projeto e à implementação de processos paralelos [TOR, 2001].

Hardware

O Tornado é um *cluster* heterogêneo, pois seus componentes não possuem uma configuração padrão e independem de fornecedor. Atualmente ele possui 8 nós com processador *Pentium*, 4 nós com processador 486 e 8 estações de trabalho *Sun Sparc*.

A rede é *Ethernet* 10 baseT e os nós são interconectados com um *switch* 3Com *Superstack* II.

Software

Os computadores pessoais possuem o sistema operacional LINUX e as estações SUN possuem o sistema operacional UNIX Solaris. A troca de mensagens entre os nós é realizada utilizando PVM e LAM/MPI.

As aplicações executadas nesse *cluster* são programas paralelos que exigem grande poder computacional, desenvolvidos em pesquisas dos estudantes da Universidade do Texas.

5.7.4 GALAXY

O galaxy é um dos principais recursos para projetos de pesquisa da Universidade Estadual de Nova York, Stoony Brook. Seu projeto foi iniciado em 1997 para ser uma solução eficaz de baixo do custo, com flexibilidade de configuração para aplicações particulares e para servir como ferramenta de aprendizado de montagem de computadores. É uma arquitetura escalável, construída com componentes facilmente substituíveis.

Hardware

Os principais componentes do Galaxy são um nó principal, nós de serviço, nós de cálculo, *switches* e sub–sistemas de entrada/saída paralela (Figura 5.4). Na configuração corrente os equipamentos são: 1 nó principal com 4 processadores

Pentium Pro e 512 Mbytes de memória principal, 2 nós de serviço com 2 processadores *Pentium* II e uma grande quantidade de nós de cálculo com processadores *Pentium* II/III, totalizando 128 processadores.

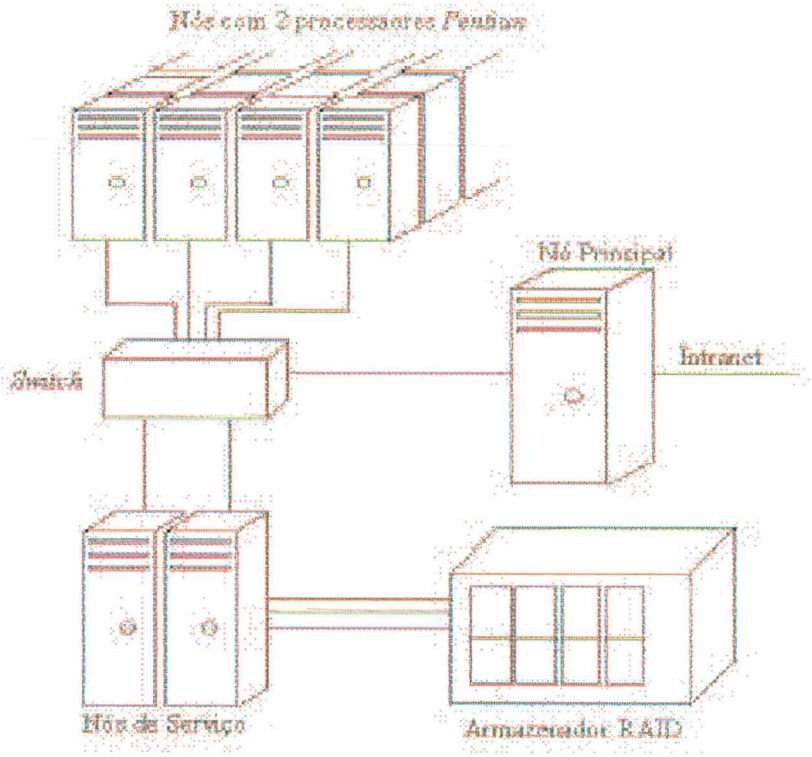

Figura 5.4: Visão esquemática do computador Galaxy

O nó principal é responsável pela interface do usuário com o computador, pois provê os serviços de acesso ao sistema, compilação e execução de programas e monitoramento dos trabalhos. Os nós de serviço abrigam o sistema de arquivos e os programas de autenticação da rede. Os nós de cálculo seguem instruções dos nós de serviço para executar o máximo de tarefas possíveis.

A rede tem uma topologia multiestrela com todos os nós e servidores conectados como um *cluster* em uma rede mista *FastEthernet* e *Gigabit Ethernet*. Um *switch* não bloqueante, BigIron 8000 com 128 portas *FastEthernet* e 8 portas *Gigabit Ethernet*, interliga todos os componentes, tornando o Galaxy um sistema

verdadeiramente paralelo. Os nós de cálculo utilizam placas de rede Intel EtherExpress Pro 100 NICs.

Cada nó possui um disco local com capacidade de armazenamento pequena que é utilizado principalmente para armazenamento temporário. Para evitar a redundância no armazenamento das bibliotecas e ferramentas de desenvolvimento, que são utilizados por todos os nós, é utilizado um *hardware* RAID, que provê um gerenciamento assíncrono em tempo real das transferências de dados.

Software

Todos os nós utilizam o sistema operacional LINUX. Novos nós são configurados através de um disco de carga inicial que possui a configuração própria para os nós e as demais alterações, como por exemplo a substituição da versão do núcleo do sistema operacional, são realizadas remotamente. A troca de mensagens é realizada utilizando uma das duas implementações do MPI: LAM ou MPICH. A maioria dos usuários do Galaxy utilizam o MPICH.

As aplicações mais usadas no Galaxy são para a realização de cálculos muito complexos.

5.7.5 GAMMA

O GAMMA *(Genoa Active Message MAchine)* é um sistema rápido de troca de mensagens para comunicação entre processos, que executa em *clusters* de computadores conectados por redes *FastEthernet* ou *Gigabit Ethernet.* Ele pode ser utilizado em estações de trabalho autônomas, em computadores paralelos de SPMD *(Single Program Multiple Data)* ou em aplicações MIMD.

Hardware

O GAMMA executa em um conjunto de computadores com processador *Intel Pentium,* AMD K6 ou modelos superiores [DIS, 2001].

Os *clusters* que executam o GAMMA são conectados por um *hub* ou *switch* em uma rede 100baseT, sendo *FastEthernet* ou *Gigabit Ethernet*. Cada nó deve possuir no mínimo um adaptador de rede NIC (*Network Interface Card*) [THE, 2001]. Os nós podem possuir um adaptador de rede adicional para serem interligados a outra rede local [DIS, 2001].

Software

Os *clusters* que utilizam o GAMMA possuem sistema operacional LINUX acrescido de algumas chamadas de sistemas e de um *driver* NIC adaptado. Todas as características multiusuário e multitarefa do LINUX foram preservadas e utilizadas também para aplicações paralelas [DIS, 2001].

A comunicação do GAMMA é baseada em *Active Ports*, um mecanismo derivado de *Active Messages*. A adoção desse mecanismo permite um protocolo sem cópias intermediárias das mensagens trocadas entre emissor e receptor, denominado protocolo de zero cópias [DIS, 2001].

Para alcançar melhor desempenho o GAMMA provê os seguintes mecanismos básicos para troca de mensagens: grupo de processos estáticos para programas SMPD, comunicação ponto a ponto, comunicação *broadcast* entre processos e barreira de sincronização, que explora um algoritmo desenvolvido especialmente para reduzir colisões em redes *Ethernet* compartilhadas [DIS, 2001].

Os mecanismos básicos de comunicação do GAMMA estão embutidos no núcleo do LINUX, implementados como chamadas de sistema. Outras partes, como barreira de sincronização, rotinas de inicialização, algumas rotinas coletivas e *stubs* para serviços básicos, são armazenadas em uma biblioteca de programação a nível de usuário [THE, 2001].

O sistema GAMMA está apto a gerenciar simultaneamente seu mecanismo próprio e comunicação IP padrão em *Ethernet* 100base–T. Para isso é necessário preservar no *cluster* serviços de rede usuais (telnet, rsshel, nfs), assim como executar programas paralelos GAMMA/PVM/MPI utilizando implementações baseadas PVM e MPI [THE, 2001].

5.7.6 Tabela Comparativa

A seguir é apresentada uma tabela comparativa entre os projetos de *cluster* descritos anteriormente.

Tabela 5.1 –Tabela comparativa entre os projetos de *clusters*.

Projeto	***Desenvolvedor***	***Sistema Operacional***	***Biblioteca para Troca de Mensagens***	***Tipo de Rede***
Beowulf	NASA	Linux Windows	PVM MPI	FastEthernet Myrinet
PARNASS	Universidade de Bonn, Alemanha	LINUX	Score 2.4 /MPICH	FastEthernet Myrinet
Tornado	Universidade do Texas	LINUX UNIX Solaris	PVM LAM/MPI	Ethernet
Galaxy	Universidade Estadual de NY	LINUX	LAM/MPI	FastEthernet GigabitEthernet
GAMMA	UNIGE – Itália	LINUX	Active Ports	FastEthernet GigabitEthernet

6 Clux

Neste capítulo, é apresentado o projeto do *cluster* Clux e a implementação do mecanismo de comunicação do mesmo, sendo dividido em 6 subcapítulos. O subcapítulo 6.1 descreve a arquitetura final idealizada e a arquitetura atual do Clux e os componentes básicos do mesmo. O subcapítulo 6.2 apresenta uma visão geral do Clux descrevendo aspectos relevantes em relação aos canais físicos e à forma de inicialização do Clux. O subcapítulo 6.3 descreve a comunicação no Clux e os formatos das mensagens que trafegam pelas redes. O subcapítulo 6.4 apresenta a rede de trabalho, seus componentes e os operadores da sua interface. O subcapítulo 6.5 apresenta detalhadamente a rede de controle, seus componentes, os operadores da sua interface e o processo gerente que controla as conexões/desconexões remotas no Clux. No subcapítulo 6.6, são descritos passo a passo dois exemplos típicos da atuação do gerente de controle na conexão e desconexão dos canais físicos diretos.

6.1 Arquitetura

A arquitetura do *cluster* Clux é similar à do multicomputador Crux visto na seção 4.3. O Clux idealizado é composto por 32 microcomputadores completos, 4 *switches* e 1 *hub*. Desses microcomputadores, 31 são configurados como nós de trabalho e 1 como nó de controle (figura 6.1). Os nós de trabalho possuem 5 placas *ethernet*, 1 conectada ao *hub* e 4 conectadas aos *switches,* e o nó de controle possui uma placa *ethernet* ligada ao *hub*. A ligação com o *hub* constitui a rede de controle e as ligações com os *switches* constituem rede de trabalho.

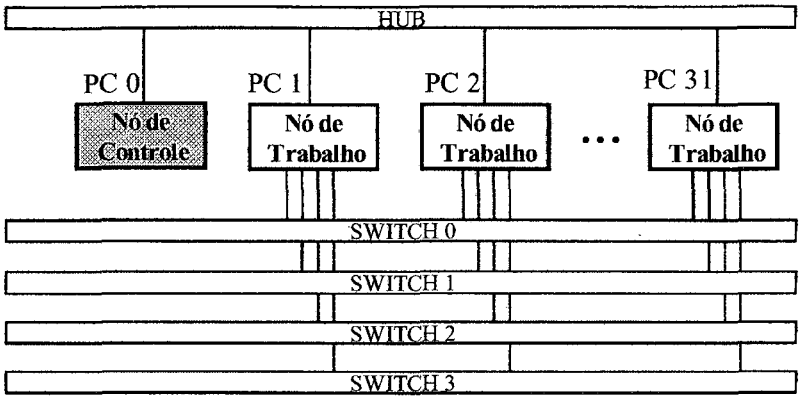

Figura 6.1 –Arquitetura do Clux.

O ambiente montado para o desenvolvimento do software do Clux é composto por 4 microcomputadores completos, 1 *hub* e 2 *switches*. Desses microcomputadores, 3 são configurados como nós de trabalho e 1 como nó de controle (figura 6.2). Os nós de trabalho possuem 3 placas *ethernet*, 1 conectada ao *hub* e 2 conectadas aos *switches*, e o nó de controle possui uma placa *ethernet* ligada ao *hub*. A ligação com o *hub* constitui a rede de controle e as ligações com os *switches* constituem rede de trabalho. Esse ambiente tem proporções menores que o idealizado, mas opera com o mesmo princípio básico de funcionamento.

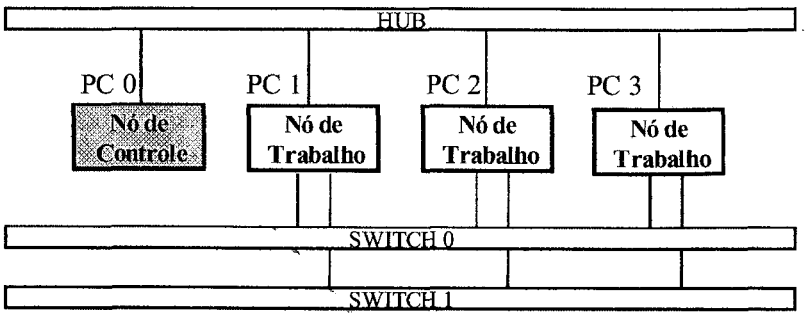

Figura 6.2 –Arquitetura atual do Clux.

Os componentes básicos do *cluster* Clux são (figura 6.3):

- **Nó de Trabalho Virtual (NTV):** Processos executados dentro dos nós de trabalho reais, responsáveis pela execução dos programas usuários e servidores de sistema.
- **Nó de Trabalho Real (NTR):** Executa um determinado número de NTVs e um processo controlador, denominado gerente do nó de trabalho real (GNTR), responsável pela configuração das comunicações dos NTVs que estão executando dentro do próprio NTR e pelas comunicações com nó de controle quando um NTV local desejar se conectar com ou desconectar de um NTV remoto.
- **Nó de controle (NC):** Possui um processo controlador denominado gerente do nó de controle (GNC), responsável pela configuração da rede de trabalho, realizando a conexão/desconexão de canais físicos diretos entre NTVs executando em NTRs distintos.
- **Rede de Trabalho:** É utilizada para o transporte de mensagens de tamanho arbitrário entre dois NTVs situados em NTRs distintos.
- **Rede de Controle:** Representada pelas ligações com o *hub*. É utilizada para o transporte das mensagens de controle entre o NC e os NTRs, visando o gerenciamento da configuração da rede de trabalho.

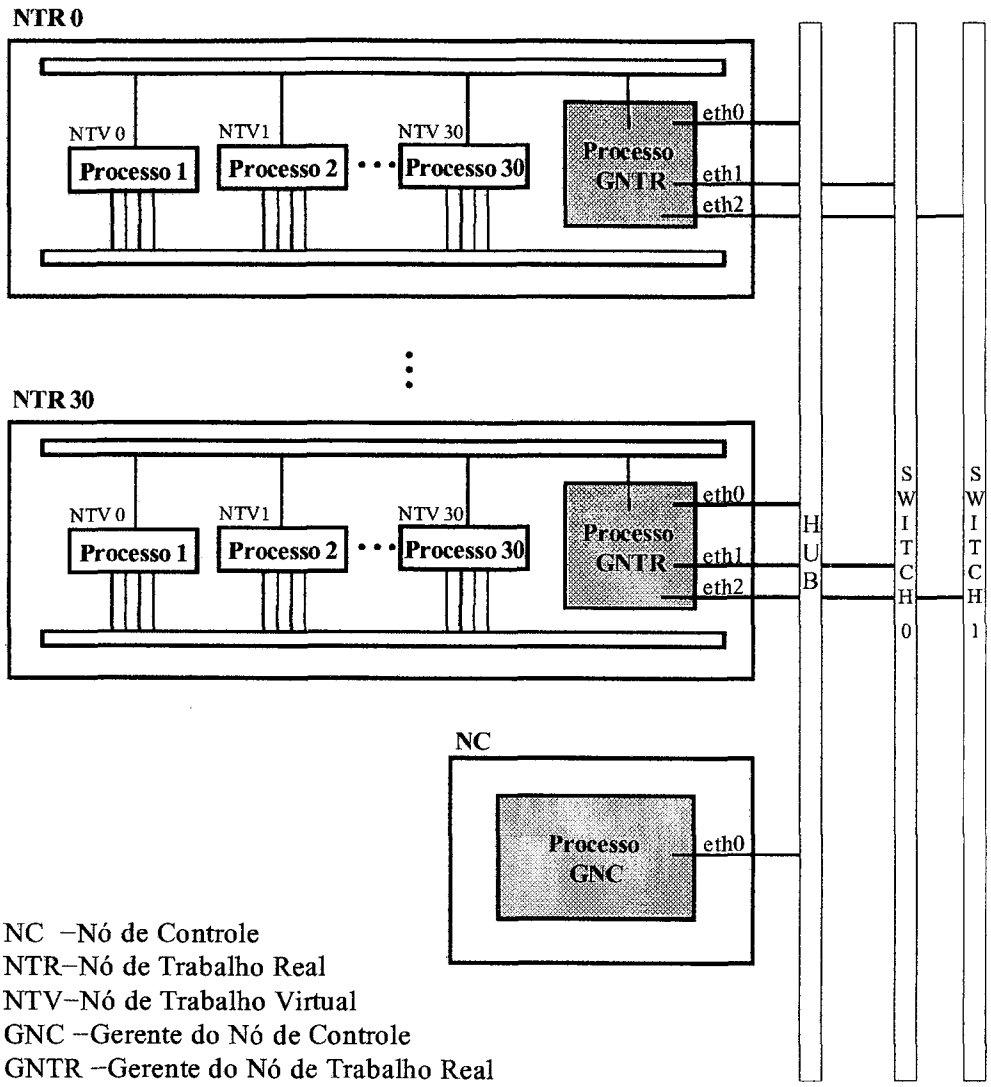

Figura 6.3 –Visão geral do cluster Clux

Os NTVs, o GNTR e os canais locais que os conectam procuram reproduzir a estrutura do Clux no interior de cada NTR.

6.2 Visão Geral do *cluster* Clux

O funcionamento básico do Clux consiste na conexão/desconexão dinâmica de canais diretos entre dois NTVs, de acordo com a demanda dos programas executando no *cluster*. Cada NTV pode estar conectado com até 4 outros NTVs em determinado momento, sendo que estas conexões podem ser locais ou remotas. Em um NTR, podem existir no máximo 2 conexões remotas ao mesmo tempo, considerando a existência de 2 *switches* no ambiente de desenvolvimento, que são os canais físicos. Os canais são denominados canais físicos quando ligam dois NTVs executando em NTRs distintos, sendo representados pelas ligações com os *switches*. Cada um desses canais é utilizado por apenas uma conexão em um determinado momento.

A inicialização do *cluster* acontece individualmente em cada um dos nós. Cada NTR possui um *script* com os procedimentos para a identificação do nó e para a ativação do GNTR. O NC possui um script com os procedimentos para a identificação do nó e para a ativação do GNC. Assim que um nó é ligado, esse *script* é executado nos procedimentos de carga inicial, inicializando o funcionamento do Clux, que é gerenciado pelos GNTRs e pelo GNC.

6.3 Comunicação

A comunicação entre processos é síncrona e acontece por de troca de mensagens através de operadores específicos adequados à arquitetura do Clux, que estão definidos na biblioteca do Clux (libclux.a). Existem operadores específicos para a rede de trabalho e para rede de controle, que são descritos nas seções 6.5.3 e 6.6.2. Esses operadores foram adaptados do Sistema Operacional ACrux [COR, 1999].

Tanto a troca de mensagens de controle quanto a troca de mensagens entre processos usuários são feitas através de *sockets*. As comunicações locais utilizam *sockets* AF_UNIX e as comunicações remotas utilizam *sockets* AF_INET, mais precisamente *sockets* UDP.

Pela rede de controle do Clux trafegam mensagens de controle *request* e *reply*. Pela rede de trabalho trafegam as mensagens msg, que são trocadas entre os NTVs através dos canais conectados. A seguir essas mensagens são descritas detalhadamente.

Mensagem request

São as mensagens utilizadas pelos operadores de comunicação para repassar as requisições dos NTVs ao GNTR e, eventualmente, deste ao GNC. A figura 6.4 apresenta o formato da mensagem *request*. Ela é formada por três campos, sendo *idRequest* um caractere que identifica o tipo de requisição (tabela 6.1), *idNTVo* a identificação do NTV origem da requisição e o *idNTVd* a identificação do NTV destino da requisição. A forma de identificação dos NTVs está descrita na seção 6.5.2.

Figura 6.4 – Formato da mensagem *request*.

A tabela 6.1 lista e descreve o significado dos tipos de mensagens *request* que são trocadas no Clux.

Tabela 6.1 –Tipos de mensagens *request*.

Tipo (idRequest)	***Descriçao***
C	Pedido de conexão para um NTV específico
A	Pedido de conexão para um NTV qualquer
D	Pedido de desconexão entre dois NTVs
E	Pedido de exclusão de uma requisição de conexão

Mensagem reply

São as mensagens utilizadas para responder às requisições enviadas pelos NTVs aos GNTRs, e as enviadas por estes ao GNC. A figura 6.5 apresenta a estrutura e o formato da mensagem *reply*. Ela é formada por quatro campos, sendo *idReply* um caractere que identifica o significado da resposta, *idNTV1* a identificação do NTV a que se destina a resposta, *idNTV2* a identificação do outro NTV envolvido na comunicação, e *txtReply* um *string* utilizado para enviar informações adicionais relevantes.

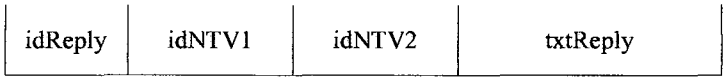

Figura 6.5 –Formato das mensagens *reply*.

A tabela 6.2 lista e descreve o significado dos tipos de mensagens *reply* que são trocadas no Clux.

Tabela 6.2 –Tipos de mensagens *reply*.

Tipo (idReply)	***Descrição***	***Observação***
–1	Aviso de erro	
1	Pedido atendido com sucesso	
2	Esperar outra reposta *reply* do GNTR	
3	Solicitar uma desconexão	O campo idNTV2 recebe o NTV a solicitar desconexão
4	Esperar outra reposta *reply* do GNC	
5	Enviar dados pelo endereço retornado	O campo txtReply o endereço IP para envio de dados
6	Conexão remota efetuada	O campo txtReply recebe o número do canal físico

Mensagem msg

São as mensagens, propriamente ditas, trocadas entre os NTVs pelos canais conectados através da rede de trabalho. A figura 6.6 apresenta o formato da mensagem msg. Ela é formada por 4 campos, sendo que *idNTVo* identifica o NTV que enviou a mensagem, *idNTVd* identifica o NTV que deve receber a mensagem, *length* identifica o tamanho da mensagem e *txtMsg* possui os dados da mensagem.

idNTVo	idNTVd	*length*	txtMsg

Figura 6.6 –Formato das mensagens msg

6.4 Rede de Trabalho

A rede de trabalho é formada pelos canais físicos diretos e é utilizada para a troca de mensagens entre os NTVs executando em NTRs distintos. Depois que as conexões são realizadas, o canal físico é utilizado por apenas uma dupla de NTVs para a troca de mensagens em um determinado momento.

6.4.1 Nó de Trabalho Real

Um NTR é um microcomputador completo que pode executar até 31 processos ao mesmo tempo (figura 6.7), onde cada processo é um NTV. Além desses processos, o NTR executa um processo controlador GNTR, responsável pela criação dos NTVs, distribuição dos programas para execução, configuração das comunicações locais e pelas solicitações de conexões/desconexões remotas ao GNC.

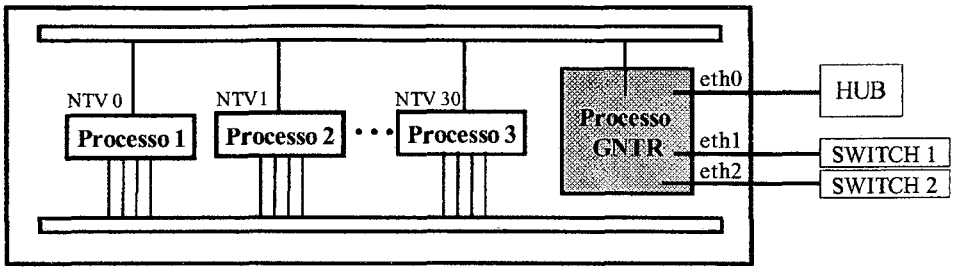

Figura 6.7 –Nó de trabalho Real.

6.4.2 Nó de Trabalho Virtual

Os NTVs são processos que executam dentro dos NTRs. São criados pelo GNTR e ficam esperando para executar qualquer programa do Clux, como programas usuários ou servidores de sistema.

Os NTVs se comunicam com o GNTR para solicitar conexão ou desconexão de canais, e uma vez conectados, trocam mensagens entre si. São identificados pela concatenação de 4 dígitos decimais, onde os dois primeiros representam o número do NTV e os dois últimos representam o número do NTR onde o NTV está executando. Por exemplo, 1201 identifica o NTV 12 que executando do NTR 01. As identificações iniciadas por 01 até 05 são reservadas para os servidores de sistema, por exemplo, um NT V identificado por 0201 executa apenas servidores de sistema.

6.4.3 Interface da Rede de Trabalho

A interface da rede de trabalho visa a transferência de mensagens entre os processos que estão executando nos NTVs. Para isso utilizam os operadores que são explicados nessa seção.

Os prefixos dos nomes dos operadores significam:

- **s_** : Disponível tanto para processos usuários quanto para processos servidores de sistema.
- **ss_** : Disponível apenas para processos servidores de sistema.

- **kcv_** : Utilizado pelos NTVs para sua comunicação com o GNTR.
- **ksv_** : Utilizados pelo GNTR para sua comunicação com os NTVs.
- **l_** : Utilizado pelos NTVs para solicitarem conexões e desconexões ao GNTR.

Os operadores utilizados para a troca de mensagens entre os NTVs são:

- **s_Send(idNTV, msg, *length)*:** Utilizado para enviar ao processo que está executando no NTV idNTV uma a mensagem msg de tamanho *length*.
- **s_Receive(idNTV, msg, *length*):** Utilizado para recepção de uma mensagem msg de tamanho *length* de um processo que está executando no NTV idNTV.
- **ss_ReceiveAny(idNTV, msg, *length*)**: Utilizado para recepção de uma mensagem msg de tamanho *length* de um processo que está executando em outro NTV qualquer. Nesse operador, o idNTV é um parâmetro de retorno.

Os operadores **s_Send**, **s_Receive** e **ss_ReceiveAny** utilizam outros operadores para a solicitação de conexão/desconexão dos canais ao GNTR. Esses operadores são:

- **l_Connect(idNTVo, idNTVd, *channel*)**: Utilizado pelo NTV idNTVo para solicitar ao GNTR uma conexão com o NTV idNTVd. O parâmetro de retorno *channel* recebe a identificação do canal por onde a mensagem será enviada.
- **l_ConnectAny(idNTVo, idNTVd, *channel*)**: Utilizado pelo NTV idNTVo para solicitar ao GNTR uma conexão com um NTV qualquer. O parâmetro de retorno idNTVd recebe a identificação do NTV destino e o parâmetro de retorno *channel* recebe a identificação do canal por onde a mensagem será enviada.
- **l_Disconnect(idNTVo, idNTVd):** Utilizado pelo NTV idNTVo para solicitar ao GNTR a desconexão com o NTV idNTVd.

Os operadores **l_Connect**, **l_ConnectAny** e **l_Disconnect** utilizam outros operadores para a transferência de mensagens de controle entre os NTVs e os GNTRs são:

- **kcv_SendReceive(*request,reply*)**: Utilizado pelo NTV para enviar a requisição *request* ao GNTR e receber a resposta *reply* do GNTR.
- **ksv_ReceiveAny(*request*)**: Utilizado pelo GNTR para receber uma requisição *request* enviada por um NTV idNTV qualquer.
- **ksv_Send(idNTV, *reply*)**: Utilizado pelo GNTR para enviar a resposta *reply* ao NTV idNTV.

6.4.4 Gerente do Nó de Trabalho Real

O GNTR é um processo controlador executado em cada um dos NTRs. Assim que é iniciado, o GNTR cria os 31 processos NTVs. A partir de então ele passa a atender solicitações para execução de programas, designando NTVs para executá-los, e requisições de conexão/desconexão vindas dos NTVs.

Quando um NTV deseja trocar mensagens com outro NTV, envia uma solicitação de conexão para o GNTR, que verifica se o NTV destino está executando localmente. Em caso positivo, o próprio GNTR gerencia a comunicação entre os NTVs. Em caso contrário, envia uma solicitação de conexão ao GNC através da rede de controle, o qual configura a rede de trabalho conectando o canal físico direto para a comunicação.

O GNTR administra uma tabela de conexões dos NTVs denominada TCGNTR, contendo a situação de todos os NTVs que estão executando localmente, uma lista de pedidos de conexões locais denominada LPL, contendo as solicitações de conexões locais ainda não atendidas, e uma lista de pedidos indefinidos denominada LPI, contendo as solicitações de conexão para um NTV qualquer ainda não atendidas. O GNTR mantém o controle da situação atual de todos os NTVs do NTR. A estrutura dessas listas e outros detalhes mais aprofundados de seu funcionamento podem ser encontrados em [REC, 2002].

O objetivo desta dissertação é a implementação da interface da rede de controle e, por isso, ela é apresentada com detalhes na próxima seção. A implementação da interface da rede de trabalho é descrita em outra dissertação [REC, 2002].

6.5 Interface Rede de Controle

A rede de controle é utilizada exclusivamente para troca de mensagens de controle entre o GNC e os GNTRs, como solicitações de conexão/desconexão, confirmações de conexão/desconexão e mensagens de erro. Os componentes básicos dessa rede são os operadores da rede de controle e o GNC.

6.5.1 Operadores da Rede de Controle

A interface da rede de controle é utilizada para a transferência de mensagens de controle entre o GNC e os GNTRs, visando a configuração das conexões e desconexões dos canais físicos diretos entre NTVs executando em NTRs distintos. Os operadores utilizados pelo GNTR são **r_Connect**, **r_ConnectAny** e o **r_Disconnect**. Eles utilizam os operadores **kc_SndRcv**, para o envio de requisições e recebimento das repostas. O GNC utiliza os operadores **ks_ReceiveAny** e **ks_Send** para receber as requisições e enviar as repostas, respectivamente.

Os prefixos dos nomes dos operadores significam:

- **kc_** : Utilizado pelos GNTRs para sua comunicação com o GNC.
- **ks_** : Utilizados pelo GNC para sua comunicação com os GNTRs.
- **r_** : Utilizado pelos GNTRs para solicitarem conexões e desconexões remotas ao GNC.

ks_ReceiveAny(request)

Esse operador é utilizado exclusivamente pelo GNC para ficar aguardando uma requisição *request* de um GNTR qualquer. Ele cria *socket* e é executada a função

rcvfrom do protocolo UDP para aguardar uma mensagem, que será armazenada em seu parâmetro de retorno *request*.

ks_Send(idNTV, reply)

Esse operador é utilizado exclusivamente pelo GNC para enviar a resposta *reply* de uma requisição *request* ao GNTR origem. Assim que o GNC recebe uma requisição *request* através do **ks_ReceiveAny**, atende a solicitação recebida e envia a resposta *reply* através do *socket* com a função ***sendto*** do protocolo UDP para o GNTR que enviou a requisição.

kc_SndRcv(request, reply)

Esse operador é utilizado exclusivamente pelos GNTRs através dos operadores **r_Connect**, **r_ConnectAny** e **r_Disconnect** para transmitir uma requisição *request* ao GNC e esperar uma resposta *reply* do GNC. É criado um *socket* UDP com o endereço IP do NC e com a porta do GNC. Em seguida, a requisição *request* é enviada com a função ***sendto*** do protocolo UDP através desse *socket*. Depois disso fica esperando a mensagem de resposta *reply* com a função ***rcvfrom*** do protocolo UDP. Se o campo idReply da mensagem *reply* for igual a 4 (ver tabela 6.2), é aguardada uma nova mensagem do GNC que conterá o *reply* a ser retornado à função chamadora. Caso o campo idReply da mensagem *reply* seja diferente de 4, é retornada imediatamente à função chamadora.

r_Connect(idNTVo, idNTVd, channel)

Esse operador é utilizado pelos GNTRs para requisitar ao GNC uma conexão de um NTV local com um NTV específico, que está executando num NTR distinto. Primeiramente é montada a requisição *request* com o campo idRequest igual a C (ver tabela 6.1), o campo idNTVo recebe a identificação do NTV que requisitou a conexão e o campo idNTVd recebe a identificação do NTV destino. Em seguida, é realizada uma chamada ao operador **kc_SndRcv** para enviar ao GNC a requisição *request* e receber a

resposta *reply*. Se o campo idReply de *reply* for igual a −1 (ver tabela 6.2), houve erro na conexão e o operador **r_Connect** retorna −1 ao GNTR. Se o campo idReply de *reply* for igual a 6 (ver tabela 6.2), a conexão foi realizada com o canal identificado no campo txtReply de *reply*, assim **r_Connect** armazenará no seu parâmetro de retorno *channel* o endereço IP para a criação do *socket* UDP para a troca de mensagens. Se o campo idReply de *reply* for igual a 3 (ver tabela 6.2), o GNC envia um *reply* com os campos idNTV1 e idNTV2 contendo os NTVs para solicitar desconexão, assim é realizada uma chamada a **r_Disconnect** e, logo após, inicia o procedimento de solicitação de conexão. Os endereços IP de todos os canais dos NTRs estão guardados num arquivo de cabeçalho denominado cabrc.h. Sempre que o campo idReply de *reply* for diferente de −1 o **r_Connect** retorna 1 para indicar que a conexão foi realizada com sucesso.

r_ConnectAny(idNTVo, idNTVd, channel)

Esse operador é utilizado pelo GNTR para que o NTV local idNTVo solicite ao GNTR uma conexão com um NTV *idNTVd* qualquer. Primeiramente é montada a requisição *request* com o campo idRequest igual a A (ver tabela 6.1), o campo idNTVo recebe a identificação do NTV que requisitou a conexão. Assim como no **r_Connect**, é utilizado o operador **kc_SndRcv** para enviar ao GNC a requisição *request* e receber a resposta *reply*. Se o campo idReply de *reply* for igual a −1 (ver tabela 6.2), houve erro na conexão e o operador **r_ConnectAny** retorna −1 ao GNTR. Se o campo idReply de *reply* for igual a 6 (ver tabela 6.2), a conexão foi realizada com o canal identificado no campo txtReply de *reply* e com o NTV identificado no campo idNTV2 de *reply,* assim **r_Connect** armazenará no seu parâmetro de retorno *channel* o endereço IP para a criação do *socket* UDP e no seu parâmetro de retorno idNTVd o NTV destino da conexão. Sempre que o campo idReply de *reply* for diferente de −1 o **r_ConnectAny** retorna 1 para indicar que a conexão foi realizada com sucesso.

r_Disconnect(idNTVo, idNTVd)

Esse operador é utilizado pelo GNTR para requisitar a desconexão de um NTV local idNTVo com um NTV executando em um NTR distinto idNTVd. Primeiramente é montada a requisição *request* com o campo idRequest igual a D (ver tabela 6.1), o campo idNTVo recebe a identificação do NTV que requisitou a desconexão e o campo idNTVd recebe o outro NTV da conexão. Assim como em **r_Connect** é realizada uma chamada ao operador **kc_SndRcv** para enviar ao GNC a requisição *request* e receber a resposta *reply*. Se o campo idReply de *reply* for igual a 1 (ver tabela 6.2) indica que a desconexão foi realizada e o **r_Disconnect** retorna 1.

6.5.2 Gerente do Nó de Controle

O GNC é o processo responsável pela configuração da rede de trabalho de acordo com a demanda dos NTVs, controlando as conexões e desconexões dos canais físicos diretos entre dois NTVs executando em NTRs distintos. Assim que o Clux inicia o seu funcionamento, o GNC começa a executar no NC e fica aguardando requisições vindas de GNTRs quaisquer.

O GNC administra uma tabela de controle das conexões (TCGNC), uma lista com os pedidos de conexões remotas para NTVs específicos (LPR) e uma lista de pedidos de conexão para NTVs quaisquer (LPI).

Tabela de conexões do GNC

A TCGNC é uma tabela que possui uma entrada para cada canal físico do Clux (figura 7.1), utilizada para controlar as suas conexões. As informações existentes em cada entrada são a identificação do NTR, o número do canal e as identificações do NTV origem e destino da conexão para este canal.

Como na arquitetura atual do Clux cada NTR possui 2 canais físicos, na TCGNC existem duas entradas para cada NTR, sendo uma entrada para cada canal.

Quando as identificações dos NTVs não estão preenchidas, o canal está livre para futuras conexões.

A figura 6.8, mostra, como exemplo, a TCGNC parcialmente preenchida em uma situação hipotética. O preenchimento das entradas 1 e 5 indica que o NTV 0101 está conectado ao NTV 0503 pelo canal físico 1. O preenchimento das entradas 6 e 8 indica que canal físico 2 dos NTRs 03 e 04 estão livres.

	NTR	*Canal*	*NTV 1*	*NTV 2*
1	01	1	0101	0503
2	01	2	0101	0702
3	02	1	0102	0604
4	02	2	0702	0101
5	03	1	0503	0101
6	03	2		
7	04	1	0102	0604
8	04	2		
9	...	...	...	...

Figura 6.8 : Exemplo da tabela de conexões TCGNC.

Lista de Pedidos Remotos

A LPR é uma lista utilizada para guardar os pedidos de conexão pendentes para um NTV específico, requisitados através do operador **r_Connect** (figura 7.2). As conexões só podem ser efetivadas quando existir algum canal livre e os dois NTVs estiverem interessados em se conectar. Assim sendo, quando chega uma requisição de conexão para um NTV específico e não existe canal físico livre ou não existe uma solicitação de conexão do NTV destino para o NTV que está solicitando a conexão, o pedido é guardado na LPR até que a conexão possa ser efetivada. Os pedidos são guardados em ordem de chegada, e retirados quando os NTVs estão aptos a se conectar.

A figura 6.9, mostra, como exemplo, a LPR parcialmente preenchida em uma situação hipotética. O preenchimento da entrada 1 indica que o NTV 0101 está aguardando um pedido de conexão do NTV 0903.

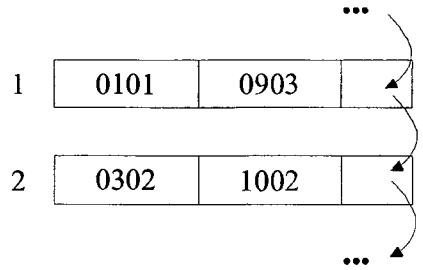

Figura 6.9: Exemplo da lista LPR.

Lista de Pedidos Indefinidos

A LPI é uma lista utilizada para guardar os pedidos pendentes de conexão para um NTV qualquer, requisitados através do operador **r_ConnectAny** (figura 7.3). Sempre que chega uma requisição de conexão para um NTV qualquer, o pedido é guardado na LPI até que chegue um pedido de conexão para o NTV origem e a conexão possa ser efetivada. O GNC possui os pedidos de todos os GNTRs e esses possuem uma LPI com os pedidos vindos dos seus NTVs locais.

Sempre que um GNTR recebe um pedido de l_ConnectAny, inclui o pedido na LPI local e avisa ao GNC, através de **r_ConnectAny**, que o NTV quer se conectar com qualquer outro e este inclui o mesmo pedido na LPI do NC. Sempre que o GNTR realiza uma conexão local, retira o pedido da LPI local, e avisa ao GNC, que também retira o pedido da LPI do NC. Sempre que o GNC realiza uma conexão remota, retira o pedido da sua LPI, e avisa ao GNTR, que também retira o pedido da LPI do NTR. Assim sendo, cada GNTR possui uma LPI com os pedidos dos NTVs locais e o GNC possui uma LPI com os pedidos de todos os NTRs em determinado momento.

A figura 6.10, mostra, como exemplo, a LPI parcialmente preenchida em uma situação hipotética. O preenchimento da entrada 2 indica que o NTV 0302 está aguardando um pedido de conexão de um NTV qualquer.

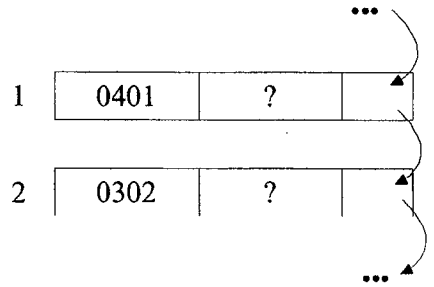

Figura 6.10: Exemplo da lista LPI.

Algoritmo Genérico do GNC

A figura 6.11 apresenta o algoritmo genérico do GNC.

```
1. Cria LPR
2. Cria &LPI
3. Cria TCGNC

4. Laço infinito
5.      espera requisição com ks_ReceiveAny(&request)
6.      se campo idRequest for igual a C (ver tabela 6.1)
7.              se não existe canal livre
8.                      insere pedido em LPR
9.                      Campo idReply = 4 (tabela 6.2)
10.                     responde ao GNTR origem ks_Send(idNTVo, reply);
11.                     Campos idReply=3 e idNTV2=idNTV 1° conectado (tabela 6.2)
12.                     responde GNTRs com NTVs a desconectar ks_Send(idNTV, reply);
13.             senão #Existe canal livre#
14.                     se existe pedido de destino para origem em LPR
15.                             atualiza TCGNC
16.                             exclui pedido de destino para origem de LPR
17.                             Campos idReply = 6 e txtReply = canal (tabela 6.2)
18.                             responde ao destino ks_Send(idNTVd, reply)
19.                             responde ao origem ks_Send(idNTVo, reply)
20.                     senão se existe pedido de destino para " " em LPI
21.                             atualiza TCGNC
```

Continua ...

```
... Continuação
22.                         exclui pedido de LPI
23.                         Campos idReply = 6 e txtReply = canal (tabela 6.2)
24.                         responde  ao destino ks_Send(idNTVd, reply)
25.                         responde  ao origem ks_Send(idNTVo, reply)
26.                 senão #Não existe pedido#
27.                         insere pedido em LPR
28.                         Campo idReply = 4 (tabela 6.2)
29.                         responde  ao origem ks_Send(idNTVo, reply)
30.                 fim se
31.         fim se
32.     senão se  campo idRequest  = D (tabela 6.1)
33.         atualiza TCGNC
34.         Campo idReply = 1 (tabela 6.2)
35.         responde ao origem ks_Send(idNTVo, reply)
36.         se existe pedidos de conexões pendentes em LPI ou LPR
37.                 exclui pedidos das listas
38.                 atualiza TCGNC
39.                 Campos idReply = 6 e txtReply = canal (tabela 6.2)
40.                 responde aos dois GNTRs ks_Send(idNTV, reply)
41.         fim se
42.     senão se  campo idRequest = A (tabela 6.1)
43.         insere pedido em LPI
44.         Campo idReply = 1
45.         responde ao origem ks_Send(idNTVo, reply)
46.     senão se  campo idRequest = E (tabela 6.1)
47.         exclui de LPI pedido de NTV origem para qualquer NTV
48.     fim se
49.fim do laço
```

Figura 6.11: Algoritmo genérico do GNC.

O algoritmo genérico do GNC é comentado a seguir, utilizando a numeração das linhas como referência:

linhas 1–3 Cria as estruturas manipuladas pelo GNC.

linha 4 Inicia um laço infinito para que o GNC fique executando ininterruptamente, enquanto o *cluster* estiver em funcionamento.

linha 5 Fica aguardando uma requisição com o operador **ks_ReceiveAny**, vinda de algum GNTR.

linhas 7–31 Se o pedido veio de **r_Connect**, é verificado se existe canal físico livre para os dois NTRs. Se não existe canal livre o pedido é inserido em LPR e uma resposta *reply* é enviada ao GNTR origem indicando que o idNTVo terá que aguardar outra mensagem do GNC quando for possível atender a requisição, após isso são verificados os dois NTVs envolvidos na conexão mais antiga com o canal desejado e um *reply* é enviado aos GNTRs indicando que uma desconexão, entre esses dois NTVs, deve ser solicitada. Se existe canal livre, é verificado se existe algum pedido do NTV destino para o NTV origem em LPR ou se existe algum pedido do NTV destino para qualquer NTV em LPI. Se existe pedido em alguma das listas, a TCGNC e a lista correspondente (LPI ou LPR) são atualizadas e o GNC envia a reposta aos GNTRs que solicitaram as conexões, sendo enviada primeiro ao que entrou antes na lista. Se não existir pedido do destino em nenhuma das duas listas, o GNC insere o pedido em LPR e envia ao GNTR origem uma resposta *reply* indicando que o GNTR terá que aguardar outra mensagem do GNC, assim que for possível atender a requisição.

linhas 32–41 Se o pedido veio de **r_Disconnect** a TCGNC é atualizada e o GNC envia uma reposta *reply* ao origem indicando que o pedido de desconexão foi recebido e executado. Depois é verificado se existem pedidos de conexões pendentes, que já podem ser efetuadas, em LPR e LPI. Se existe pedido pendente as listas e a TCGNC são atualizadas e o GNC envia aviso aos GNTRs que solicitaram as conexões, sendo que a resposta é enviada em ordem de chegada na lista.

linhas 42–45 Se o pedido veio de **r_ConnectAny**, é incluído em LPI e o GNC envia uma resposta ao origem, informando que um pedido de conexão deve ser aguardado.

linhas 46–49 Se o GNTR enviar um pedido igual a E (ver tabela 6.1), indica que este realizou uma conexão local e excluiu pedido de sua LPI. Logo, o GNC terá que retirar da sua LPI o pedido do NTV indicado no campo idNTVo da mensagem *request* para um NTV indeterminado.

6.6 Exemplos

Para que fique mais simples compreender a atuação do GNC no atendimento das requisições, alguns exemplos com situações possíveis são descritos nessa seção. Esses exemplos são mostrados na seguinte seqüência: situação em que o *cluster* se encontra antes das requisições, os passos seguidos pelo GNC para atender e responder às requisições e a situação final do *cluster* depois que as requisições foram atendidas.

A numeração das setas nas figuras 7.6 e 7.9 correspondem a ordem temporal de execução dos passos para estabelecer as conexões e desconexões.

6.6.1 Conexão Entre Dois NTVs Remotos

Neste exemplo, são apresentadas as ações do GNC para atender requisições quando dois NTRs desejam realizar uma conexão e existe canal livre para conectá–los.

Situação Inicial

A situação do *cluster* no momento da requisição é que não existem pedidos nas listas LPI e LPR e os canais estão conectados como mostra a figura 6.12.

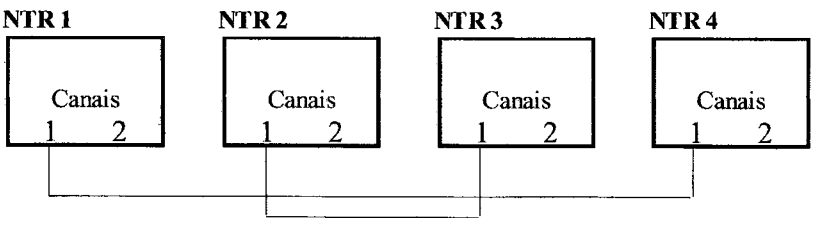

Figura 6.12 –Situação inicial dos canais conectados.

A atuação do GNC

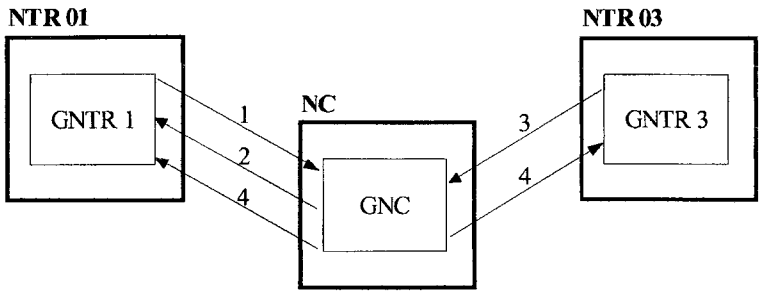

Figura 6.13 –Passos do GNC para atender as requisições.

1–O GNC recebe do GNTR 1 uma requisição, através do operador **ks_ReceiveAny**, de conexão do NTV 0101 com um NTV qualquer, que foi gerado pelo operador **r_ConnectAny** e enviado pelo kcv_SndRcv. O pedido é guardado na lista LPI.

2–O GNC retorna ao GNTR origem uma resposta, através do operador **ks_Send**, indicando que este espere uma solicitação de conexão de um NTV qualquer para o NTV 0101.

3–O GNC recebe do GNTR 3 uma requisição, através do operador **ks_ReceiveAny**, de conexão do NTV 0603 com o NTV 0101. O GNC verifica que os dois NTRs estão com o canal 2 livre e que existe pedido do NTV 0101 para um NTV qualquer em LPI, portanto a conexão pode ser efetuada. A tabela de conexões TCGNC e a lista LPI são atualizadas.

4–O GNC retorna ao GNTR 1 uma resposta informando que NTV 0101 foi conectado com o NTV 0603 pelo canal 2, através do operador **ks_Send**.

5–O GNC retorna ao GNTR 3 uma resposta informando que o NTV 0603 foi conectado com o NTV 0101 pelo canal 2, através do operador **ks_Send**.

6–Depois disso os NTVs 0101 e 0603 podem começar a troca de mensagens pelo canal conectado.

Situação Final

Depois que as requisições foram atendidas, a situação do *cluster* ficou como mostra a a figura 6.14.

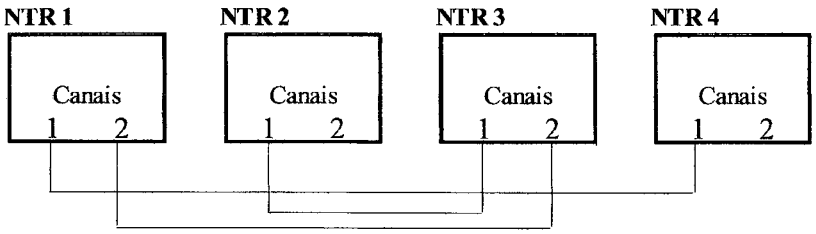

Figura 6.14 –Situação final dos canais conectados.

6.6.2 Desconexão Entre Dois NTVs Remotos e Conexão de Pedidos Pendentes

Neste exemplo são apresentadas as ações do GNC para atender uma requisição de desconexão de um GNTR qualquer. Com a liberação do canal, o GNC realiza uma conexão pendente e avisar aos NTRs envolvidos.

Situação Inicial

A situação do *cluster* no momento da requisição é que em LPR existe uma solicitação do NTR1 (do NTV 0401 para NTV 0702) e uma solicitação do NTR2 (do NTV 0702 para NTV 0401) e . Os canais estão conectados como mostra a figura 6.15.

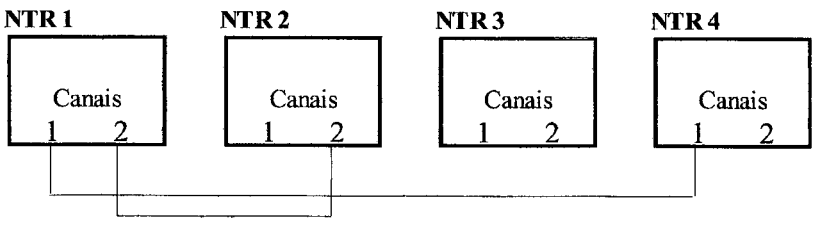

Figura 6.15 –Situação inicial dos canais conectados.

Atuação do GNC

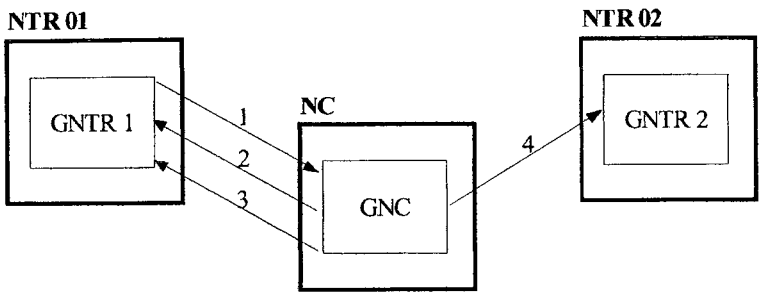

Figura 6.16 –Passos do GNC para atender as requisições.

1–O GNC recebe do GNTR 1 uma requisição, através do operador **ks_ReceiveAny**, de desconexão do NTV 0101 com o NTV 0902.

2–O GNC atualiza a TCGNC e retorna ao GNTR origem uma resposta indicando que a desconexão foi realizada. Depois disso, verifica que em LPR existem pedidos pendentes entre os NTR 1 e NTR 2, e que a conexão pode ser efetuada, pois o canal 2 foi liberado para os estes NTRs.

3–O GNC avisa ao GNTR 1 que o NTV 0401 foi conectado com o NTV 0702 pelo canal 2.

4–O GNC avisa ao GNTR 2 que o NTV 0702 foi conectado com o NTV 0401 pelo canal 2.

5–Depois disso, os NTVs 0401 e 0702 podem trocar mensagens pelo canal conectado.

Situação Final

Na situação final, os canais continuam conectados como apresentado na figura 6.15, da situação inicial. Isso acontece porque o canal foi desconectado, mas logo em seguida foi realizada uma nova conexão pelo mesmo canal.

7 Conclusões

Inúmeros projetos para a adaptação ou construção de ambientes paralelos utilizam a arquitetura baseada em *clusters* de computadores. Isso acontece devido a facilidade de adaptar o funcionamento dos *clusters* para trabalharem de forma semelhante às arquiteturas paralelas dedicadas e os custos reduzidos para a construção dos mesmos em relação a essas arquiteturas.

O cluster Clux foi projetado visando a exploração do processamento paralelo. Levando-se em consideração os projetos estudados no decorrer do trabalho, percebeu-se que Clux possui algumas vantagens inovadoras, sendo as principais:

- Os canais físicos são criados dinamicamente, de acordo com a demanda dos processos.
- Os canais físicos não são compartilhados, resultando numa maior velocidade na comunicação através da rede de trabalho.
- Os operadores para a troca de mensagens foram projetados para essa arquitetura, utilizando apenas os recursos necessários para o funcionamento do Clux, evitando *overhead* com testes e tráfego de mensagens desnecessárias.

O Clux compõe uma superestrutura que utiliza o sistema operacional UNIX e o protocolo de redes de computadores TCP/IP, utilizando *sockets* UDP na comunicação.

O mecanismo de comunicação do Clux é formado pela interface da rede de trabalho e pela interface da rede de controle. O objetivo deste trabalho foi a implementação da interface rede de controle do Clux, representada por um conjunto de operadores e por um processo gerente do nó de controle.

Estima-se que o projeto Clux, devido a sua amplitude, proporcione o surgimento de outros trabalhos que aproveitem o mecanismo de comunicação implementado. Dessa forma, linguagens como o Super Pascal e pacotes como o PVM e o MPI podem sofrer adaptações para tirarem proveito desse mecanismo.

A implementação das interfaces da rede de trabalho e de controle foi realizada utilizando recursos de protocolos convencionais de rede de computadores, como a comunicação através de *sockets*. A utilização de um protocolo específico desenvolvido especialmente para a rede de interconexão do cluster poderia melhorar consideravelmente o desempenho do ambiente. Levando–se em consideração o que foi dito, sugere–se que seja desenvolvido um projeto com essa finalidade.

8 Referências Bibliográficas

[BAR, 1999] BARRETO, Marcos E., NAVAUX, Philippe O.A. *Um Ambiente para Execução de Aplicações Paralelas em Agregados de Multiprocessadores.* Universidade Federal do Rio Grande do Sul. Porto Alegre. 1999.

[CAM, 1995] CAMPOS, Rodrigo A. Um Sistema Operacional Fundamentado no Modelo Cliente–Servidor e um Simulador Multiprogramado de Multicomputador. Dissertação de mestrado. CPGCC–UFSC. Florianópolis, 1995.

[CHA, 2000] CHAPIN, Steve; WORRINGEN, Joaquim. *Operating Systems. Cluster Computing White Paper.* Versão 2.0. Dezembro de 2000.

[CHI, 1999] CHIAZZOTTO, Mauro; SILVA, Luís Antonio. TCP/IP Tecnologia e Implementação. Editora Érica, São Paulo, 1999.

[COM, 1995] COMMER, Douglas E. *Internetworking with TCP/IP Principles, Protocols, and Architecture.* Vol.1. 3ª edição. Ed. Prentice Hall. New Jersey. 1995

[COR, 1999] CORSO, Thadeu B. Crux: Ambiente Multicomputador Configurado por Demanda. Tese de doutorado. CPGEE. UFSC. Florianópolis. 1999.

[CPW, 2001] Disponível por www em: www.computerword.iol.pt/cw_dictionary_01.asp?artigo=1741. Pesquisado em outubro de 2001.

[DIE, 1997] DIETZ, Hank. *LINUX Parallel Processing Using clusters.* Purdue University School of Eletrical and Computer Engineering. 1997.

[DIS, 2001] Disponível por www em: http://www.disi.unige.it/project/gamma. Pesquisado em novembro de 2001.

[DUM, 1995] DUMAS, Arthur. Programando WinSock. Ed. AXCEL Books. Rio de Janeiro. 1995.

[ERI, 2002] Disponível por www em: http://hammer.prohosting.com/~cribcrto/avcd2.html. Pesquisado em janeiro de 2002.

[FAG, 2001] Disponível por www em: http://www.efagundes.com/disc2/00000020.htm . Pesquisado em outubro de 2001.

[FER, 2001] FERRETO, Tiago Coelho; et al. GNU/Linux como Plataforma para Pesquisa em Alto Desempenho. PUCRS. CPAD. Centro de Pesquisa em Alto Desempenho. Porto Alegre. 2001.

[GOI, 2002] Disponível por www em: http://www.fe.up.pt/~goii2000/M3/redes1.htm . Pesquisado em janeiro de 2002.

[GUI, 2002] Disponível por www em: http://www.gta.ufrj.br/grad/guilherme/netware3.ht. Pesquisado em janeiro de 2002.

[KLA, 2001] Disponível por www em: http://www.inf.ufrgs.br/gpesquisa/propcar/disc/cmp134/trabs/T1/011/klat2. Pesquisado em outubro de 2001.

[LIN, 2001] Disponível por www em: http://www.dhbrown.com. Pesquisado em novembro de 2001.

[MEN, 2000] MENDES, A. L. de Andrade. Migração de Aplicações para *Clusters* de PCs. Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e Tecnológicas. Plano de Dissertação de Mestrado. Curitiba. Fevereiro de 2000.

[MYR, 2001] Disponível por www em: http://inf.ufrgs.br/gpesquisa/propcar/disc/cmp134/trabs/T1/001/myrinet. Pesquisado em novembro de 2001.

[OVE, 2001] Disponível por www em: http://www.myri.com/myrinet/overview/index.html. Pesquisado em novembro de 2001.

[PAR, 2001] Disponível por www em: http://www.parnass.net/. Pesquisado em outubro de 2001.

[PIT, 2001] Disponível por www em: http://www.tiraduvidas.hpg.ig.com.br/Dicas/030801a.htm. Pesquisado em novembro de 2001.

[PPG, 2001] Disponível por www em: http://www.ppgia.pucpr.br/~maziero/parapuc/rrass/esplogica.htm. Pesquisado em outubro de 2001.

[REC, 2002] RECH, Luciana. Interface da Rede de Trabalho do *Cluster* Clux. Dissertação de Mestrado. CPGCC. UFSC. Fevereiro de 2002.

[ROS, 1999] G. DE ROSE, C.. Arquiteturas paralelas versáteis e de baixo custo para a pesquisa e o Pesquisa na área de processamento paralelo e distribuído. Conferência Latino Americana de Informática. Assunção, Paraguai. 1999.

[SCH, 1999] SCHWEITZER, Marc Alexander.,ZUMBUSCH, Gerhard.,GRIEBEL, Michael. Parnass2: A *cluster* of Dual–Processor Pcs. Editor: W. Rehm, T. Ungerer, Chemnitzer Informatik Berichte. 1999.

[STE, 1998] STEVENS, Richard W. *Unix Networking Programming*. Vol.1. 2ª edição. Ed: Prentice–Hall. E.U.A. 1998.

[STU, 1996] STUMPF, Ricardo Dantas; NASSUR, Carlos Constantino Moreira; BON, Carlos Eduardo Bacellar. Redes de Fibra Ótica. Universidade de Brasília. 1996.

[SUN, 2001] Disponível por www em: http://www.cbpf.br/~sun/SUNHPC.html. Pesquisado em OUTUBRO de 2001.

[TAN, 1995] TANENBAUM, Andrew S. Sistemas Operacionais Modernos. Ed: Prentice Hall do Brasil Ltda. Rio de Janeiro. 1995

[TAN, 1997] TANENBAUM, Andrew S. Redes de Computadores. Ed: Campus. 3ª edição. Rio de Janeiro. 1997.

[THE, 2001] Disponível por www em: http://www.disi.unige.it/dottorato/THESES/1999-02-CiaccioG.ps.gz. Pesquisado em novembro de 2001.

[TOR, 2001] Disponível por www em: http://www.acm.org/crossroads/xrds6-1/tornado.html. Pesquisado em OUTUBROde 2001.

[USP, 2001] Disponível por www em: http://www.uspnet.usp.br/evolucao/node15.html. Pesquisado em outubro de 2001.

[ZOM, 1996] ZOMAIA, Albert Y. H. *Parallel & Distributed Computing HandBook.. Series on Computer Engineering.* Ed: McGraw-Hill. E.U.A. 1996., Albert Y. H.