FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Clusters Beowulf

Por

Hugo Emanuel Gonçalves Teixeira

Maria João Almeida de Sá Barros

Paulo Jorge Marques Coelho

Sistemas Distribuídos

Mestrado em Eng. Electrotécnica e de Computadores
1º Semestre

Professor Dr. José Magalhães Cruz

PORTO
2000

SUMÁRIO

SIGLAS E ABREVIATURAS lv
INTRODUÇÃO 1

CAPÍTULO I
SISTEMAS COMPUTACIONAIS DE ELEVADO DESEMPENHO 2

1.1. CLUSTERS – CONTEXTO HISTÓRICO E MOTIVAÇÕES 6
1.2. CLUSTERS – ARQUITECTURA GENÉRICA 6
1.3. CLASSIFICAÇÃO DE CLUSTERS 8

CAPÍTULO II
CLUSTERS BEOWULF 9

2.1. ARQUITECTURA DE UM CLUSTER BEOWULF 9
2.2. APLICAÇÕES 12
2.3. VANTAGENS E DESVANTAGENS DOS CLUSTERS BEOWULF 12

CAPÍTULO III
HARDWARE 14

3.1. ACONDICIONAMENTO DO MATERIAL 14
3.2. REFRIGERAÇÃO DO SISTEMA 15
3.3. TOPOLOGIAS E TECNOLOGIAS DE REDE 15
3.4. NOVAS TECNOLOGIAS DE REDE ALTERNATIVAS PARA INTERLIGAR O CLUSTER 18
- ETHERNET 20
- FAST ETHERNET 20
- GIGABIT ETHERNET 20
- ATM 21
- SCI 22
- MYRINET 22
- HIPPI 24
- FDDI 25

CAPÍTULO IV
SOFTWARE 27

4.1. SISTEMA OPERATIVO 27
4.2. GESTÃO DE RECURSOS E ESCALONAMENTO 28
4.3. ESPAÇO DE PROCESSOS DISTRIBUIDO 29
4.4. PARALLEL VIRTUAL FILE SISTEM 32
4.5. AMBIENTES DE PROGRAMAÇÃO 37
DIPC 37
PVM 38
LAM 39
4.6. COMPILAÇÃO E DEBUG 40
4.7. APLICAÇÕES DE GESTÃO DO CLUSTER 41

CAPÍTULO V
IMPLEMENTAÇÕES 44

5.1. MAGI 44
5.2. HIVE 46
5.3. VITARA 49

CONCLUSÃO 52
BIBLIOGRAFIA 53
ANEXO – COMO IMPLEMENTAR UM BEOWULF 57

SIGLAS E ABREVIATURAS

ATM – Asynchronous Transfer Mode
BPROC – Beowulf Distributed Process Space
CC-NUMA – Cache-Coherent Nonuniform Memory Access
CESDIS – Center of Excellence in Space Data and Information Sciences
COW – Cluster of Workstations
CPU – Central Processing Unit
DIPC – Distributed IPC
DSM – Distributed Shared Memory
ESS – Earth and Space Sciences project
FDDI – Fiber Distributed Data Interface
GNU – GNU is Not Unix
GPID – Global Process Identifier
GPL – GNU Public License
GRE – Gestão de Recursos e Escalonamento
GSFC – Goddard Space Flight Center
HIPPI – High Performance Parallel Interface
HPC – High Performance Computing
IP – Internet Protocol
IPC – Interprocess Communication
LAM – Local Area Multicomputer
LAN – Local Area Network
MIMD – Multiple Instruction Multiple Data
MPI – Message Passing Interface
MPP – Massively Parallel Processors
NASA – National Aeronautics and Space Administration
NFS – Network File System
NOW – Network of Worstations
PC – Personal Computer
PID – Process Identifier
PoPC – Pile of PCs
PVFS – Parallel Virtual File System
PVM – Parallel Virtual Machine
SCI – Scalable Coherent Interface
SMP – Symmetric Multiprocessors
SO – Sistema Operativo
SSI – Single System Image

STP – Shielded Twisted Pair
TCP – Transmission Control Protocol
UDP – User Datagram Protocol
UTP – Unshielded Twisted Pair
WAN – Wide Area Network

Cap. – capítulo
Fig. – figura

INTRODUÇÃO

São cada vez mais frequentes as aplicações que necessitam de enorme capacidade computacional, que não pode ser obtida em simples máquinas sequenciais com uma única unidade de processamento central – CPU. Apesar de ainda ser possível melhorar o desempenho dos processadores, a actual evolução acabará por se deparar com restrições inalienáveis como por exemplo a dimensão física dos componentes, a velocidade da luz, as leis da Termodinâmica, e o custo, que se poderá revelar excessivo com a aproximação a esses limites. Por conseguinte, a única solução económica e tecnologicamente viável consiste na interligação de múltiplos processadores de forma a permitir a execução cooperativa de tarefas com um objectivo comum em paralelo. A origem do processamento paralelo como substituto do processamento sequencial é, portanto, bipolar:

- O processamento paralelo revelou-se a alternativa viável na resolução do *ponto de estrangulamento* que constitui um único processador num sistema;
- O custo de um sistema paralelo é consideravelmente inferior ao de um sistema sequencial com desempenho análogo, uma vez que o preço dos componentes é uma função da sua capacidade com crescimento superior a linear (aproxima-damente exponencial).

O presente trabalho pretende constituir uma introdução aos sistemas de processamento paralelo designados Clusters Beowulf, cuja proliferação se tem vindo a verificar nos últimos anos. Estes clusters têm-se revelado cada vez mais a alternativa economicamente viável aos sistemas paralelos de elevado desempenho tradicionais.

A organização do trabalho é a seguinte: No primeiro capítulo aborda-se as diversas arquitecturas de sistemas computacionais de elevado desempenho – não se trata de uma descrição exaustiva pois o seu objectivo consiste unicamente em enquadrar a arquitectura Beowulf. Esse capítulo termina com uma análise genérica dos clusters. No segundo capítulo apresenta-se a arquitectura Beowulf propriamente dita. No terceiro capítulo descreve-se essencialmente as tecnologias de rede disponíveis e no quarto apresenta-se algum do *software* que pode ser utilizado. O quinto capítulo descreve algumas implementações concretas de clusters Beowulf. Em anexo inclui-se um pequeno tutorial acerca da implementação de um cluster deste género.

CAPÍTULO I

SISTEMAS COMPUTACIONAIS DE ELEVADO DESEMPENHO

Durante as últimas décadas verificou-se o aparecimento de um conjunto de sistemas computacionais de elevado desempenho (HPC – *High Performance Computing*) baseados em diversas arquitecturas, das quais se refere as mais comuns:

- Massively Parallel Processors (MPP);
- Symmetric Multiprocessors (SMP);
- Cache-Coherent Nonuniform Memory Access(CC-NUMA);
- Clusters;
- Sistemas Distribuidos.

Estas diferentes arquitecturas, representadas na figura 1.a., distinguem-se essencialmente pela forma de interligação de unidades de processamento e de memória.

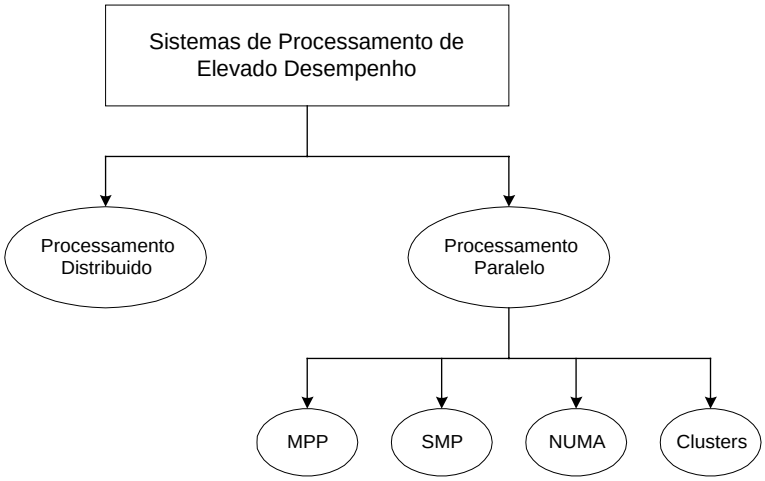


Figura 1.a. – Arquitecturas de HPC

Sistemas Distribuidos

Os sistemas distribuidos são basicamente as redes convencionais de computadores independentes interligados por alguma tecnologia de rede, podendo ou não existir uma camada de software que tente tanto quanto possível oferecer uma imagem unificada do sistema ao utlizador (Transparência). São tipicamente sistemas de grande dimensão e heterogéneos em

termos de *hardware* e de *software*. Podem incluir qualquer dos sistemas paralelos referidos como seu subconjunto. Ao contrário dos sistemas paralelos o principal objectivo não é a rapidez de processamento mas sim questões organizacionais e económicas, acesso distribuido aos dados, fiabilidade e escalabilidade. Em geral os recursos destes sistemas não são exclusivamente dedicados ao processamento paralelo mas a actividades de gestão de informação da organização a que pertencem.

Sistemas Paralelos

Os sistemas paralelos são exclusivamente dedicados ao processamento de aplicações paralelas (excepto alguns tipos de clusters que serão referidos oportunamente).

A arquitectura MPP consiste usualmente num conjunto de algumas centenas de nós interligados através de uma tecnologia de rede de alta velocidade, ou seja, baixa latência e elevada largura de banda. Cada nó é constituído por:

- Um ou vários processadores;
- Uma unidade de memória principal;
- Eventualmente, unidades de armazenamento de dados.

Cada nó executa uma cópia distinta do sistema operativo. Alguns dos nós podem encontrar-se ligados a dispositivos periféricos. Tipicamente existe apenas um nó por processador – arquitectura *Shared-Nothing* uma vez que não há memória nem dispositivos de armazenamento partilhados pelos nós. Trata-se de uma arquitectura semelhante aos clusters mas que apresenta usualmente maior número de nós e latência mais reduzida.

A arquitectura SMP, ilustrada na figura 1.b.,é do tipo *Shared-Everything*, ou seja, todos os processadores partilham a memória, o barramento de comunicação e o sistema de input/output. Desta forma é necessária apenas uma cópia do sistema operativo.
O problema da contenção no acesso aos recursos partilhados limita o número de processadores – usualmente 64 no máximo. Este problema limita consideravelmente a escalabilidade destes sistemas. A comunicação entre processadores é efectuada através da memória partilhada.
Para reduzir a contenção no acesso à memória estes sistemas utilizam memória *cache* de alta velocidade junto de cada processador, que armazena a informação da memória principal que foi acedida mais recentemente. Todos os pedidos de acesso à memória passam pela memória *cache* e apenas se extendem à memória principal quando aquela não é capaz de responder ao pedido efectuado. Este mecanismo reduz a comunicação no barramento e permite aumentar o

número de processadores no sistema. Quanto maior a memória *cache* individual maior é a probabilidade de conter a informação que o processador necessita, ou seja, maior é a taxa de acerto (*hit rate*). No entanto, a introdução de memória *cache* origina a possibilidade de incoerência dos dados. Para resolver este problema empregam-se usualmente dois mecanismos:

- *Write-through-cache*: sempre que se escreve na memória *cache* escreve-se também na memória principal;
- *Snoopy-cache*: todas as memórias *cache* monitorizam o barramento. Sempre que se observe uma escrita numa posição de memória principal que esteja replicada nessa *cache* esta ou actualiza o respectivo valor ou o elimina.

As memórias *cache* utilizadas com estes dois mecanismos permitem a redução da ocupação média do barramento de comunicação – as leituras que sejam supridas pela memória local não causam ocupação do meio partilhado.
Todos os sistemas operativos actuais dedicados a servidores suportam esta arquitectura.

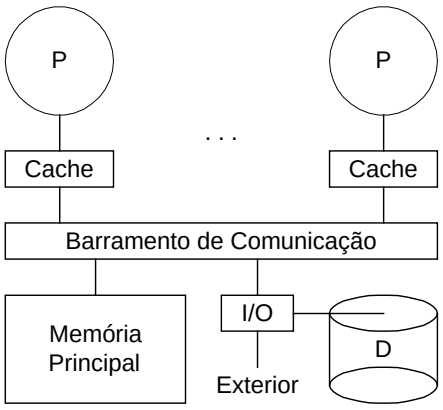


Figura 1.b. – Arquitectura SMP

Os sistemas CC-NUMA são multiprocessador e com uma arquitectura de acesso não uniforme à memória. Tal como em SMP todos os processadores têm uma visão global da memória, no entanto, os tempos de acesso dependem da proximidade da memória a que se acede, ao contrário dos sistemas de acesso uniforme à memória. A figura 1.c. ilustra esta arquitectura.

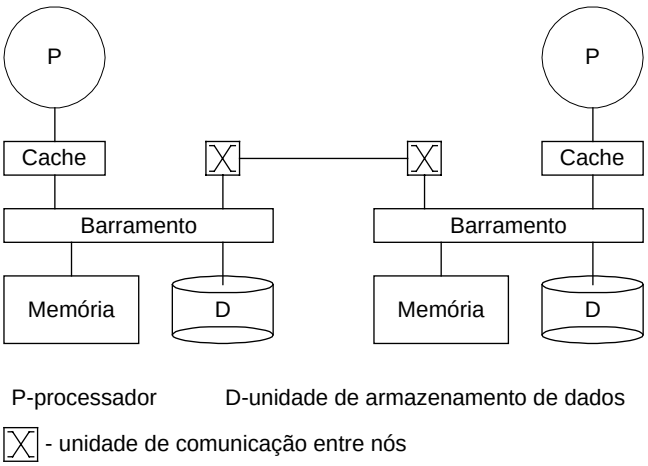


Figura 1.c. – Arquitectura NUMA

Uma vez que a memória se encontra fisicamente distribuída é necessário recorrer a mecanismos que garantam a sua consistência.

Os Clusters, também usualmente designados *Cluster of Workstation* (COW) ou *Network of Workstation* (NOW), consistem basicamente num conjunto de *Workstations* ou Computadores Pessoais (PCs), interligados por uma tecnologia de rede. Trata-se portanto de uma arquitectura multicomputador ou *loosely-coupled*. A arquitectura Beowulf encontra-se incluída neste sub-tipo de sistemas paralelos.

A tabela seguinte sumariza as principais características dos sistemas referidos:

Característica	MPP	SMP, CC-NUMA	Clusters	Sist Distribuído
Nº de Nós	O(100)-O(1000)	O(10)-O(100)	O(100)-O(1000)	O(10)-O(1000)
Comunicação	Mensagens e memória partilhada distribuida	Memória partilhada e memória partilhada distribuida(NUMA)	Mensagens e Distributed IPC	Ficheiros partilhados, RPCs, mensagens, Distributed IPC
Imagem de máquina virtual	Parcial	Sempre em SMP, Parcial emNUMA	Desejada mas não necessária	Parcial ou não existente
Nº de cópias e tipo do SO	N micro-Kernels	Um Kernel monolítico em SMP e vários em NUMA	N iguais em todas as plataformas, monolíticos ou não	N monolíticos iguais ou não
Espaços de endereços	Múltiplos ou único com memória partilhada distribuida	Único	Múltiplos ou único	Múltiplos
Segurança entre nós	Desnecessária	Desnecessária	Eventualmente	Necessária
Propriedade	Uma organização	Uma organização	Uma organização	Uma ou mais org.s

1.1. CLUSTERS – CONTEXTO HISTÓRICO E MOTIVAÇÕES

Desde o início da década de 90 tem-se vindo a verificar uma tendência gradual de migração de soluções dispendiosas, especializadas e com arquitecturas proprietárias, como é o caso dos tradicionais supercomputadores, para soluções de menor custo e mais flexíveis como os clusters. As principais razões que motivaram a utilização dos clusters são essencialmente as seguintes:

- O desempenho das *Workstations* e PCs tem evoluído de forma muito acentuada com a duplicação da capacidade dos processadores a cada 18 meses (Lei de Moore);
- As novas tecnologias das redes locais permitem elevada largura de banda e baixa latência;
- Os clusters são mais fáceis de integrar em redes já existentes;
- As ferramentas de desenvolvimento para esta arquitectura têm vindo a ser objecto de normalização e rápida evolução;
- Os clusters são uma alternativa de menor custo que os restantes sistemas paralelos e apresentam um elevado potencial de Escalabilidade.

A limitação mais relevante continua a ser a rede de comunicações, em termos de latência e largura de banda, pelo que esta arquitectura se torna viável apenas no caso de aplicações que não utilizem a rede de uma forma intensiva.

Mais recentemente a distinção entre *Workstations* – plataformas tipicamente Unix dedicadas a cálculos complexos e exaustivos – e PCs – mais orientados para funções administrativas e processamento de texto, tem vindo a desaparecer. Nos últimos anos verificou-se uma convergência entre ambos tanto ao nível do desempenho dos processadores como ao nível das funcionalidades dos sistemas operativos. Por esta razão e devido ao seu custo mais reduzido, os PCs têm vindo a substituir as *Workstations*.

1.2. CLUSTERS – ARQUITECTURA GENÉRICA

Conforme referido anteriormente, um cluster consiste num conjunto de máquinas – nós do sistema – constituídas por processador, memória e eventualmente unidades de armazenamento de dados e periféricos, interligadas de forma a executar tarefas de uma forma cooperativa. Genericamente, cada nó pode ser uma máquina uniprocessador ou uma máquina

SMP por exemplo. No entanto, usualmente os nós são PCs conforme já se referiu. A figura 1.2.a. representa a arquitectura de um cluster.

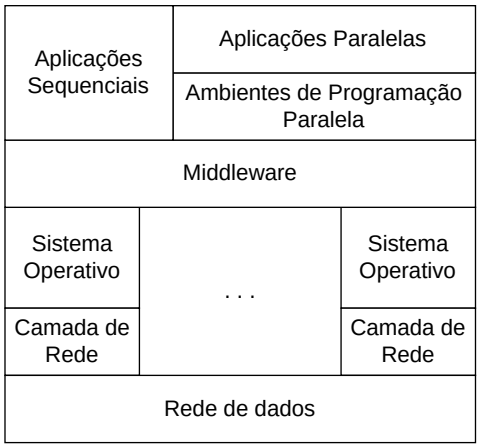


Figura 1.2.a. – Arquitectura de um Cluster

A camada designada de Middleware, situada entre o sistema operativo e as aplicações, é responsável pela criação do conceito de Transparência do sistema, também designado *Single System Image* (SSI), controlo de transacções e recuperação de falhas. Idealmente, o Middleware de fornecer os seguintes serviços:

- Ponto de Acesso único: o acesso deve ser efectuado como se de uma máquina se tratasse. Por exemplo deve ser possível efectuar *telnet beowulf.fe.up.pt* mas não *telnet nó1.beowulf.fe.up.pt*;
- Sistema de Ficheiros único: o utilizador deve ver o sistema de ficheiros das diversas máquinas como uma única hierarquia sob a mesma directoria raíz;
- Espaço de Memória único: tanto quanto possível a camada de Middleware deve oferecer um serviço de memória partilhada distribuída de forma transparente para os utilizadores e aplicações;
- Espaço de Input/Output único: qualquer nó deve conseguir efectuar operações de input/output em dispositivos locais ou remotos utilizando um espaço de endereços único;
- Espaço de Processos único: cada processo deve ter um identificador único no cluster e deve poder criar e comunicar com processos no próprio nó ou num outro qualquer nó. O conjunto de processos deve ser gerido de uma forma global;

- Migração de Processos: para equilibrar a carga dos nós deve ser possível efectuar dinamicamente a migração de processos entre os diversos nós.

1.3. CLASSIFICAÇÃO DOS CLUSTERS

Os clusters são usualmente classificados de acordo com um conjunto de atributos, nomeadamente:

a) Aplicação:
- Clusters de Elevado Desempenho – utilizados em cálculo científico;
- Clusters de Elevada Disponibilidade – utilizados em tarefas críticas;

b) Utilização dos nós:
- Clusters Dedicados – os recursos do cluster são utilizados exclusivamente para processamento paralelo;
- Clusters Não Dedicados – cada nó é dedicado a um utilizador e as aplicações paralelas correm nos ciclos de processor não utilizados pelas tarefas submetidas pelos utilizadores;

c) Hardware:
- Clusters de PCs (CoPCs) ou Pilha de PCs (PoPC) – os nós são PCs;
- Clusters de *Workstations* (COW) – os nós são *Workstations*;
- Clusters de SMPs (CLUMPs) – os nós são máquinas SMP;

d) Sistema Operativo:
- Clusters Linux;
- Clusters Solaris;
- Clusters NT;
- Clusters AIX;

e) Configuração dos nós:
- Clusters Homogéneos – todos os nós com a mesma arquitectura e sistema operativo;
- Clusters Heterogéneos – nós com arquitecturas e sistemas operativos distintos;

f) Número de nós:
- Clusters de Grupo – de 2 a 100 nós;
- Clusters Departamentais – de dezenas a centenas de nós;
- Clusters Organizacionais – com várias centenas de nós.

CAPÍTULO II

CLUSTERS BEOWULF

Os clusters Beowulf resultaram de um projecto iniciado pela NASA no Verão de 1994 no centro de pesquisas CESDIS, localizado no *Goddard Space Flight Center*, no âmbito do projecto *Earth and Space Sciences*, cujo objectivo primário consistia em determinar a aplicabilidade de arquitecturas de processamento paralelo a problemas do domínio das ciências espaciais, nomeadamente para o tratamento de dados recolhidos por satélite, a preços acessíveis. O primeiro destes clusters foi construído por Thomas Sterling e Don Becker e consistia em 16 nós com processadores Intel 486 Dx4 a 100Mhz interligados por uma tecnologia de rede Ethernet a 10 $Mbits^{-1}$ do tipo *channel bonded* com dois barramentos. Desde então diversas instituições têm construído os seus próprios clusters Beowulf. Actualmente existem clusters deste tipo com mais de um milhar de nós.
A designação Beowulf não se trata de um acrónimo com significado técnico. Beowulf é o nome de um herói lendário da literatura épica medieval inglesa, que foi escolhido pelos criadores desta arquitectura.

Os clusters Beowulf surgem num contexto de necessidade crescente de elevada capacidade de processamento em diversas áreas científicas com o objectivo de constituírem sistemas computacionais poderosos e economicamente viáveis. Para isto contribuiu a evolução do desempenho dos processadores, a aproximação entre PCs e *Workstations*, a diminuição do custo das tecnologias de rede e dos próprios processadores e o software gratuito e *open source* , como o Linux por exemplo, ao abrigo da GNU Public License (GPL).

2.1. ARQUITECTURA DE UM CLUSTER BEOWULF

Como qualquer cluster, enquadra-se na classe de sistemas computacionais de elevado desempenho, HPC, de processamento paralelo. Como sub-tipo de cluster trata-se de uma pilha de PCs (PoPC), interligados por tecnologias de rede comuns. Portanto, trata-se de uma topologia *Multiple Instruction Multiple Data* (MIMD), do tipo multicomputador ou *loosely coupled*, baseada no kernel monolítico do Linux, à qual é intrínseca a comunicação através de mensagens (os mecanismos de memória partilhada, se existentes, são suportados por serviços de mensagens subjacentes).

A base física destes sistemas são *commodity hardware components*, ou seja, componentes largamente comercializados (*massificados*). O software utilizado é de elevado desempenho e

fiabilidade e é gratuito como por exemplo os sistemas operativos Linux e FreeBSD sobre os quais são instaladas ferramentas de processamento paralelo também gratuitas e amplamente utilizadas como é o caso do PVM e MPI. São sistemas dedicados ao processamento paralelo, com uma rede de comunicações privada, de uso exclusivo do cluster. Todos os nós têm a mesma arquitectura e correm o mesmo sistema operativo. O seu desempenho é da ordem do GigaFlop ou dezenas de GigaFlop e encontra-se em contínuo crescimento devido à evolução dos processadores e ao aparecimento de clusters cada vez maiores.

O custo é um factor essencial, sendo um dos objectivos desta arquitectura conseguir uma razão capacidade de processamento/preço superior a qualquer outro tipo de sistema de processamento paralelo. Este objectivo tornou-se central na aceitação destes clusters por parte do mercado. Inicialmente considerava-se até que para poder ser designado Beowulf o preço total do cluster não deveria ultrapassar os 50.000 dólares. Com o aumento gradual da dimensão destes sistemas este limite deixou de fazer sentido.

De acordo com a classificação apresentada no capítulo 1.3. os clusters Beowulf são:

- De elevado desempenho;
- Dedicados;
- Pilhas de PCs;
- Homogéneos.

A arquitectura dos clusters Beowulf encontra-se esquematizada na figura 2.1.a.. Desta fazem parte os seguintes componentes:

- Servidor: nó do sistema que controla o cluster, efectua monitorização e distribuição das tarefas, funciona como servidor de ficheiros e efectua a interface com o exterior tanto ao nível de interligação de redes como ao nível de utilizador. Portanto, é responsável pela segurança e pela imagem que o exterior tem do cluster. É também designado *front-end* do sistema. Normalmente existe apenas um servidor por cluster, no entanto, alguns clusters de elevada dimensão têm mais que uma máquina a funcionar como servidor. O servidor é também vulgarmente a única máquina do cluster com periféricos que permitem a interface com o utilizador;
- Clientes: os restantes nós do sistema são designados clientes. São configurados e controlados pelo servidor. Não podem ser acedidos pelo exterior do cluster. São usualmente máquinas sem teclados nem monitores e eventualmente até sem disco rígido acedidas por *login* remoto a partir do servidor, e que se encarregam do processamento de tarefas atribuidas pela máquina servidor. Constituem a capacidade de processamento e armazenamento de dados do cluster;

- Rede privada dedicada que interliga todos os nós do cluster usualmente de uma tecnologia standard como a Ethernet.

Apesar de existirem diversos módulos de *software* – como por exemplo, alterações específicas do kernel do Linux, bibliotecas de processamento paralelo e ferramentas de configuração – que tornam o cluster mais fácil de configurar e utilizar, é possível construir um sistema que se enquadra nesta designação apenas com uma distribuição do Linux sem necessidade de software adicional. Por exemplo, dois computadores pessoais ligados em rede com partilha de ficheiros por *Network File System* (NFS) e possibilidade de executar *remote shell* (rsh) podem ser considerados uma máquina virtual Beowulf muito simples.

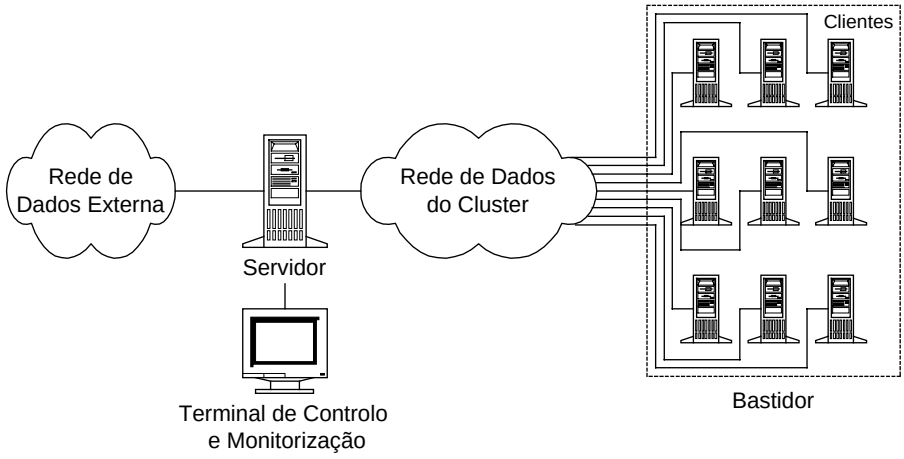


Figura 2.1.a. – Arquitectura de um cluster Beowulf

Estes clusters apresentam algumas diferenças em relação aos tradicionais *Clusters of Workstations* (COWs):

- Utilizam tipicamente componentes *massificados* pelo mercado e de custo reduzido;
- Os nós são dedicados ao processamento paralelo e não a utilizadores específicos;
- A rede de comunicações é privada e isolada do exterior;
- O software utilizado pretende criar a imagem de uma única máquina virtual;
- A interface com o utilizador efectua-se apenas ao nível do servidor.

Devido a todos estes factores, um cluster Beowulf assemelha-se mais que um COW a uma única máquina virtual com capacidade de processamento paralelo.
Estes clusters são ainda vulgarmente classificados de acordo com o hardware utilizado para diferentes objectivos de desempenho:

- Clusters Beowulf de Classe 1: integralmente constituídos por *hardware* vulgar para aplicações não tão exigentes em termos de desempenho como os da classe 2;
- Clusters Beowulf de Classe 2: utilizam *hardware* mais específico de forma a possibilitarem graus de desempenho mais elevados.

O maior número de implementações nesta área foi até agora levado a cabo por universidades e outras instituições de investigação com grande capacidade de cálculo. Entretanto começaram a aparecer empresas interessadas em comercializar este tipo de sistemas como por exemplo a COM2000 (www.com2000si.com). O cliente pode adquirir o cluster completamente configurado e pronto para exploração, necessitando apenas de efectuar a sua manutenção ou então, como é mais vulgar, pode adquirir *hardware* por nó ficando a seu cargo a montagem e configuração.

2.2. APLICAÇÕES

Conforme referido anteriormente, os cluster Beowulf aplicam-se em áreas com necessidade de processamento paralelo intensivo de grandes quantidades de dados. Para além da Computação Científica em geral destacam-se as seguintes aplicações:

- Servidores HTTP: a distribuição de recursos e carga pelos diversos nós aumenta a capacidade de resposta do servidor HTTP;
- Bases de Dados: os clusters Beowulf apresentam uma arquitectura completamente adequada às topologias de Sistemas de Bases de Dados Paralelos do tipo *Shared-Nothing*;
- Inteligência Artificial: área que utiliza algoritmos que ocupam os recursos do sistema de uma forma intensiva e exponencial em relação á dimensão do problema;
- Computação Gráfica: área na qual o tempo de processamento é crítico e tipicamente constitui uma limitação à evolução da qualidade de imagem dos objectos.

2.3. VANTAGENS E DESVANTAGENS DOS CLUSTERS BEOWULF

A utilização de clusters Beowulf apresenta um conjunto de vantagens e desvantagens em relação às restantes arquitecturas de HPC. As vantagens foram já brevemente referidas no âmbito das razões que motivaram o aparecimento dos clusters em geral (cap. 1.1.). Neste

capítulo refere-se novamente as principais vantagens desta arquitectura e também algumas desvantagens.

É possível enumerar as seguintes vantagens:

- Utilização de componentes de *hardware* vulgares e largamente disseminados e testados. O preço destes componentes é consideravelmente mais reduzido que o de componentes específicos utilizados nas restantes arquitecturas. Por outro lado, o risco da construção do cluster é minimizado pelo facto de todo o *hardware* ser reutilizável para outras aplicações como vulgares PCs;
- O *software* utilizado é gratuito e *open source*. Por conseguinte, é flexível e robusto uma vez que qualquer pessoa pode tentar efectuar *debug* e alterações ao código de base conforme necessário. As ferramentas de programação *standard* aumentam ainda a portabilidade das aplicações;
- São sistemas altamente escaláveis, ou seja, podem crescer à medida das necessidades, o que permite acompanhar a evolução tecnológica dos componentes. Possibilita ainda uma elevada capacidade de adaptação às crescentes necessidades de processamento. Portanto, é uma arquitectura flexível em termos de evolução;
- A gestão de falhas é simples, eficiente e consiste apenas na substituição dos nós em falha;
- O custo é reduzido, em comparação com as restantes arquitecturas, uma vez que utiliza componentes de *hardware* vulgares e *software* gratuíto. Por este motivo apresenta uma razão preço/desempenho mais reduzida que os restantes sistemas. A flexibilidade de evolução contribui também para o custo mais reduzido uma vez que não é necessário adquirir equipamento topo-de-gama que será mais barato no futuro e pode ser facilmente adicionado ao cluster. Estima-se que o desempenho destes sistemas seja no mínimo três vezes superior a equipamentos com as outras arquitecturas de HPC com o mesmo preço.

No entanto existem algumas desvantagens na utilização destes sistemas, nomeadamente:

- O *hardware* de rede tipicamente utilizado não foi criado especificamente para processamento paralelo, o que origina latência superior e largura de banda inferior a outras arquitecturas como por exemplo o SMP. Os algoritmos de programação utilizados deverão ser tolerantes a este problema. A utilização de hardware mais específico que minimize este inconveniente origina custos acrescidos;
- É necessária experiência na administração de redes de computadores para construir, configurar e gerir um cluster Beowulf.

CAPÍTULO III

HARDWARE

Antes de decidir encomendar qualquer tipo de hardware, é boa ideia considerar primeiro o desenho do sistema. Existem basicamente duas componentes de hardware envolvidas no projecto de um sistema Beowulf: o tipo de computadores ou nós que se vai usar, e a forma como se vão interligar esses nós. Embora o número de opções não seja muito grande, existem algumas decisões importantes que devem ser tomadas relativamente ao projecto antes de se construir um sistema Beowulf.

Dependendo do tipo de aplicação que se pretende dar ao Beowulf, assim se deve decidir quanto ao número de nós a colocar no *cluster.* Existem implementações muito variadas desde 4 nós até implementações com várias centenas de nós. Para os nós pode optar-se por encomendar os componentes desmontados (memória, *drives*, discos, processadores, *motherbord,* caixa *minitower*, placas de rede, etc) e fazer a montagem dos componentes nas caixas *minitower* ou optar por encomendar as unidades já montadas, o que poupa bastante trabalho técnico.

3.1. ACONDICIONAMENTO DO MATERIAL

Depois de todos os nós estarem individualmente montados é necessário estudar a melhor forma de acondicionar o material. A solução encontrada pode ser bastante variada e depende essencialmente da quantidade de nós. Até oito ou nove PCs é possível acomodar dentro de um bastidor de 19'' (figura 3.1.b.). A utilização de bastidores tem vantagens, porque o equipamento fica protegido e a cablagem fica escondida. O equipamento activo de rede deve sempre que possível ficar dentro de um bastidor. Nas implementações com maior quantidade de nós, os PCs podem ser colocados em prateleiras especiais, construídas propositadamente para o efeito, como se pode observar no exemplo da figura 3.1.a..

A organização dos cabos e a forma como as máquinas são ligadas é fundamental. Se existir alguma falha a esse nível, pode levar a diagnósticos errados, dirigindo a atenção para questões de configuração.

Figura 3.1.a. – Solução com prateleiras.

Figura 3.1.b. – Solução com bastidor.

3.2. REFRIGERAÇÃO DO SISTEMA

O acondicionamento do material deve ter em conta eventuais problemas de sobre-aquecimento, resultantes da grande concentração de equipamento. Deve ser prevista instalação de ar condicionado na sala onde o Beowulf vai ficar instalado. No caso de se utilizarem bastidores estes devem incluir *kits* de ventilação, para facilitar a circulação de ar. Na versão com prateleiras, no caso de não haver ar condicionado poderão ser instaladas ventoinhas exteriores.

Para se conseguir monitorizar a evolução da temperatura no *cluster*, é importante que as *motherboards* à semelhança do que acontece nos sistemas maiores tenham um *chipset* com controlo e indicação de temperatura. Este pormenor pode ser importante, para evitar problemas de sobre-aquecimento não controlados.

3.3. TOPOLOGIAS E TECNOLOGIAS DE REDE

Conforme referido nos capítulos anteriores, um cluster é um tipo de sistema de processamento paralelo, que consiste num conjunto de computadores (nós), interligados entre si, funcionando de uma forma integrada como se fosse um único sistema de computação. Um nó de um cluster pode ser um sistema mono-processador ou multiprocessador, PC, *workstation* ou SMP, com memória, facilidades de I/O e sistema operativo. Os clusters Beowulf em particular são pilhas

de PCs, consistindo no mínimo em dois ou mais nós (PCs) interligados através de uma rede local privada. Tipicamente utilizam-se dois segmentos Ethernet, como se pode observar na Figura 1.3.a. , para interligar os nós.

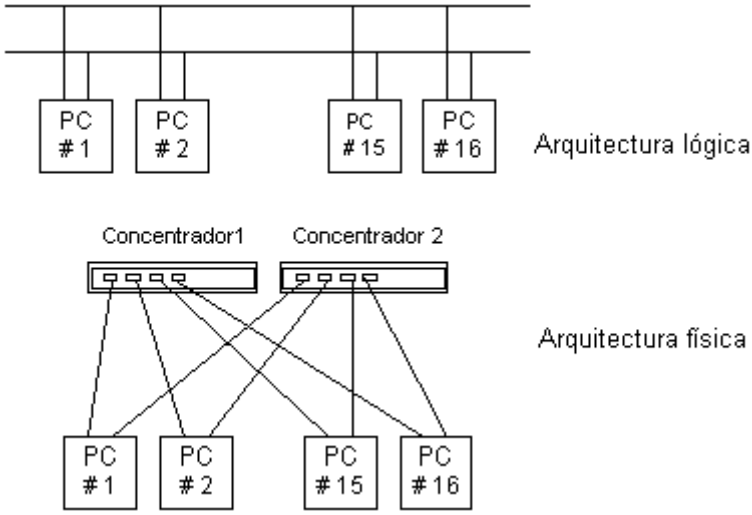


Figura 1.3.a. - Configuração de rede com dois segmentos Ethernet em paralelo.

A interligação clássica entre os nós no Beowulf é feita através de dois segmentos Ethernet em paralelo. Os processadores comunicam utilizando TCP/IP. Nesta topologia, a performance das comunicações inter-processador é severamente limitada pela performance (latência e largura de banda) da tecnologia Ethernet.

O Beowulf tem sido usado para explorar a viabilidade de utilizar múltiplas redes Ethernet em paralelo para satisfazer a largura de banda requerida para a transferência de dados interna. Nesta arquitectura, cada nó do Beowulf tem acesso, de forma transparente para o utilizador, a várias redes, que constituem domínios de difusão distintos. No exemplo da figura 3.3.a. podemos ver que cada PC do cluster tem acesso a dois segmentos Ethernet. Esta arquitectura é designada *channel bonding* e está implementada como um melhoramento ao nível do kernel do Linux (que é o sistema operativo tipicamente utilizado nos clusters Beowulf).

Vários projectos Beowulf mostraram que podem ser utilizadas várias redes em paralelo para obter um aumento significativo de largura de banda e portanto validar a utilização da técnica de *channel bonding*.

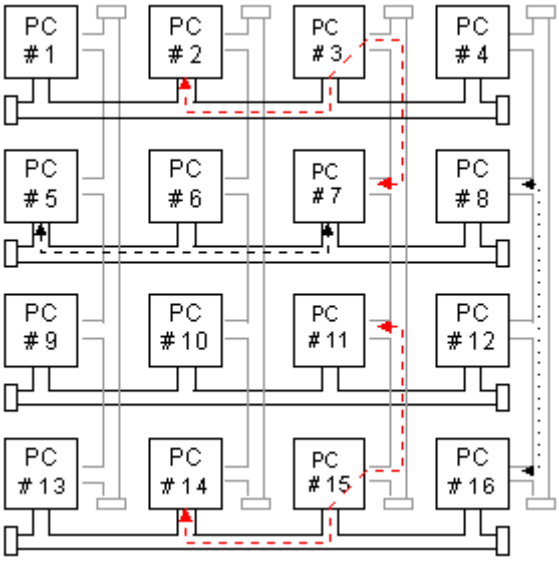


Figura 3.3.b. – Exemplo de configuração de rede com *routing* por *software*

A topologia de rede do Beowulf original (figura 3.3.a.) consiste num par de segmentos Ethernet que interliga todo o cluster operando em paralelo como se fosse um único barramento virtual. Esta configuração apresenta uma largura de banda agregada de 20 Mbps, que constitui um factor bastante limitativo para a performance do cluster.

Outra configuração alternativa consiste em explorar a tecnologia Ethernet 10 Mbps com dois adaptadores de rede por nó, mas criando agora oito segmentos separados. Cada nó é ligado a dois segmentos e cada segmento interliga quatro nós como se pode observar no exemplo da figura 3.3.b.. Os quatro segmentos verticais estão interligados entre si assim como os segmentos horizontais. Embora a largura de banda agregada teórica tenha quadruplicado com este esquema, existem restrições à largura de banda agregada efectiva e à performance.

A topologia de rede do Beowulf original (2 segmentos Ethernet) interliga cada nó com uma rota directa para todos os nós do cluster não havendo necessidade de fazer encaminhamentos na rede.

Na topologia da figura 3.3.b. podemos observar que cada um dos nós no *cluster* comunica directamente com apenas seis nós (nós locais), os três nós com quem partilha o segmento vertical e os três nós com quem partilha o segmento horizontal. Para um nó comunicar com os outros nove nós (nós remotos), é necessário que um nó intermédio que é local ao emissor e ao receptor funcione como *router*, encaminhando os pacotes através dos segmentos de rede. Desta forma dois nós podem sempre comunicar usando não mais que um nó intermédio que é o *router*. Desta topologia com *routing* por *software* resultam duas consequências:

- O *routing* consome largura de banda. Um pacote para passar de um segmento para outro tem de ser replicado nos dois segmentos e consome o dobro da largura de banda agregada do que um pacote que não precise de passar pelo *router*;
- O *routing* não é grátis. Existe um custo adicional a pagar em latência associado com o *routing*, quer seja feito por hardware quer seja por *software*, sendo este custo mais elevado no *routing* por *software.*

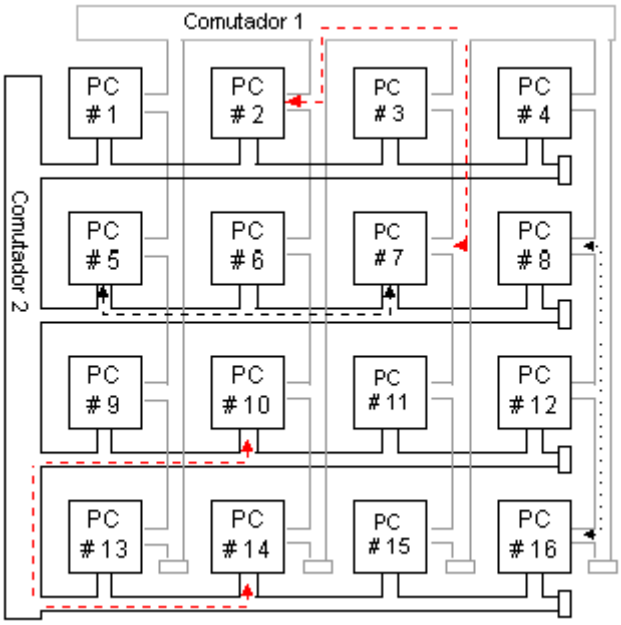
A topologia com *routing* tira vantagem do encaminhamento IP, que é uma funcionalidade implementada no kernel do Linux. Isto permite que os nós funcionem como *routers*, aceitando um pacote numa interface, determinando se deve ser reencaminhado, podendo também ser fragmentado e transmitindo-o pela segundo interface. Apesar dos adaptadores de rede utilizados no Beowulf serem do tipo *Bus master,* o que permite reduzir a carga no CPU porque copiam os dados de e para a memória principal autonomamente, a máquina que faz o *routing* continua a ter que carregar o *software* para atender um *interrupt* de recepção, alocar espaço temporário para o pacote a ser reencaminhado, copiá-lo, consultar a tabela de *routing* e colocar o pacote na fila de transmissão do interface respectivo. Este processo que também é feito pelos *routers* designa-se *store and forward* e provoca latência na rede.

Uma variação alternativa a este esquema de interligação pode ser observado na figura 3.3.c.. Este novo esquema mantém a ligação física dos oito segmentos do esquema com *routing* anterior, mas adiciona dois comutadores Ethernet de quatro portas para fazer a comutação entre os segmentos. Os comutadores Ethernet de alta velocidade encarregam-se de fazer a comutação entre os segmentos, libertando os nós das tarefas de encaminhamento. Para evitar estrangulamentos na comutação, o comutador deve ter como especificações técnicas uma largura de banda no *bus* superior à largura de banda dos segmentos e uma latência baixa.

3.4. NOVAS TECNOLOGIAS DE REDE ALTERNATIVAS PARA INTERLIGAR O CLUSTER

Já vimos que uma interligação do *cluster* com um único segmento Ethernet a 10 Mbps não é satisfatória em termos de performance, pois esta tecnologia de rede já está na fase de declínio e a largura de banda e latência não está balanceada com o poder computacional dos PC*s* actuais.

Figura 3.3.c. – Exemplo de configuração de rede comutada



Recentemente têm surgido novas tecnologias de rede de alta velocidade que podem ser exploradas para melhorar a performance da comunicação inter-processadores. Vamos de seguida analisar algumas delas e discutir a sua adequação para os clusters Beowulf.

Mecanismos que a rede deve suportar:

- Deve ter suporte para Linux;
- Adaptadores de rede PCI com *drivers* para Linux;
- Comunicação inter-processador Inter-nó e intra-nó;
- Protocolos eficientes;
- Primitivas de comunicação eficiente com baixa latência;
- Deve ser escalonável de forma a permitir o *upgrade* do equipamento activo.

Tecnologias de Rede local com suporte para Linux:

- Ethernet (10 Mbps);
- Fast Ethernet (100Mbps);
- Gigabit Ethernet (1Gbps);

- Myrinet (1.2Gbps);
- SCI (Dolphin – MPI – 12 μs latência);
- HIPPI;
- ATM;
- FDDI.

ETHERNET

A tecnologia Ethernet tornou-se sinónimo de rede para interligar PCs . Esta tecnologia tem um campo de utilização muito basto de tal forma que se pode encontrar em quase todos os sectores de actividade. Contudo, a sua largura de banda de 10 Mbps deixou de ser suficiente para utilização em ambientes onde exista necessidade de transferir grandes quantidades de dados ou exista uma grande densidade de tráfego.

FAST ETHERNET

Uma versão melhorada da Ethernet, surge com o nome de Fast Ethernet, que permite 100 Mbps de largura de banda. Principais características da tecnologia Fast Ethernet:

- Performance moderada (100 Mbps);
- Custo reduzido;
- Ferramentas de diagnóstico disponíveis e controlo a vários níveis;
- Cablagem em cabo de cobre UTP CAT-5;
- Adaptadores de rede com *drivers* para Linux disponíveis.

GIGABIT ETHERNET

Actualmente, o estado da arte na tecnologia Ethernet é o Gigabit Ethernet. Tem como características chave:

- Alta performance: 1 Gbps (Gigabit-por-segundo) *full-duplex*;
- Preserva a simplicidade da Ethernet;
- Cablagem em fibra ou cabo UTP CAT-6;
- Standard aprovado recentemente.

Nível físico do Gigabit Ethernet

- Ligações em fibra com conectores SC;
- Algumas empresas começaram a comercializar muito recentemente soluções Gigabit Ethernet com cablagem em cobre;
- As ligações *Half-duplex* têm distâncias muito limitadas;
- As distâncias para as ligações *Full-duplex* só são limitadas pelas perdas ópticas na fibra ou pela atenuação e *cross-talk* no cobre.

Topologia

- Estrela com repetidores *Full-duplex*;
- Existe já uma grande variedade de comutadores disponíveis.

Nova característica Ethernet: *frames* de controlo de fluxo

- Aparecem como *frames multicast*;
  - Geradas automaticamente pelo hardware;
  - Opcionalmente geradas pelo *software.*

Adaptadores de rede

- *Yellowf"* e *Hamachi*;
- Adaptadores PCI 64 bits;
  - Para utilizar com bus PCI 32 bits;
- O modelo *Hamachi* opera a 66 MHz e utiliza endereços de 64 bits;
- Consumo de potência moderado;
- Capacidade para manipular *frames* grandes;
  - Pode usar *frames* de 96 KB não standard;
- *Drivers* para Linux disponíveis.

ASYNCHRONOUS TRANSFER MODE (ATM)

O ATM é uma tecnologia de circuitos virtuais e foi originalmente desenvolvida para a indústria das telecomunicações.
O ATM é suposto ser usado tanto em redes LAN como WAN. É baseado em pequenos pacotes de dados de tamanho fixo a que se designam células.

Principais características:

- Requer comutadores sofisticados para interligar os nós;
- Todas as comunicações utilizam estabelecimento de conexões
- Cablagem em fibra ou cobre.

SCALABLE COHERENT INTERFACE (SCI)

SCI é um standard IEEE 1596-1992 desenvolvido com o objectivo de fornecer memória partilhada num ambiente distribuído. SCI foi projectado para suportar multi-processamento distribuído com uma grande largura de banda e latência reduzida. Fornece uma arquitectura escalonável que permite construir grandes sistemas utilizando componentes baratos.
O SCI é uma arquitectura ponto-a-ponto . Consegue reduzir o atraso nas comunicações inter-processador mesmo quando comparado com as melhores tecnologias actualmente disponíveis como o *Fiber Channel* e ATM.
Isto é conseguido porque elimina a necessidade de executar translações de protocolos. Uma comunicação remota ocorre como sendo um simples carregamento ou armazenamento de um processo num processador. Tipicamente um endereço remoto resulta em perda de memória cache. Isto por sua vez faz com que o controlador de cache enderece memória remota através do SCI para conseguir os dados. Os dados são capturados para a cache com um atraso na ordem dos poucos μs e o processador continua a execução.

Actualmente a empresa *Dolphin* produz placas SCI para os *SPARC's SBUS* e também já estão disponíveis placas SCI para PCI.

Esta tecnologia apresenta bons resultados em termos de latência e é especialmente recomendada para suporte a sistemas MPI. Tem como desvantagens o preço dos componentes relativamente altos e algumas restrições de implementação devido ao tipo de comutadores existentes.

MYRINET

Myrinet é uma tecnologia de rede local a 1.28 Gbps (Gibabits por segundo) *full-duplex* baseada na tecnologia usada para comunicação de pacotes nos supercomputadores MPP.
É uma tecnologia proprietária fornecida pela *Myri-com*, de alta performance e latência reduzida. Tem tolerância a falhas devido ao mapeamento automático da configuração da rede. Isto também facilita a tarefa de configuração. A tecnologia *Myrinet* suporta Linux e Windows NT.

Para além do TCP/IP suporta a implementação MPICH do MPI para alguns *packages* como *Berkeley active messages*, com latências inferiores a 10 μs.

A tecnologia Myrinet é mais dispendiosa quando comparada com Fast Ethernet, mas tem vantagens relativamente a esta. Apresenta latência muito baixa (5 μs, comunicação unidireccional ponto-a-ponto), grande largura de banda e as placas de rede têm um processador *on-board* programável que garante independência do CPU. Consegue saturar a largura de banda efectiva de um bus PCI a quase 120 Mbytes/s com pacotes de 4 Kbytes.

Uma das principais desvantagens da Myrinet é, como já foi mencionado, o preço comparado com Fast Ethernet. O custo dos componentes para uma LAN Myrinet, incluindo os cabos e comutadores, ronda os 1350 Euros por *host*. Também não existem comutadores disponíveis com mais de 8 portas, o que pode trazer alguns problemas de escalabilidade.

Uma LAN Myrinet é composta por placas de rede, comutadores e ligações. Os comutadores multi-porta podem ser ligados a outros comutadores ou a placas de rede instaladas nos nós, usando qualquer topologia. Ao contrário da Ethernet e do FDDI, que partilham a mesma largura de banda por todos os clientes, a largura de banda agregada da Myrinet pode ser escalonada com a configuração.

Uma ligação Myrinet consiste num par de canais *full-duplex*, cada um operando a uma taxa de transferência de 0.64 Gb/s (Gigabits por segundo), ou uma a uma taxa agregada para uma ligação *full-duplex* de 1.28 Gb/s. As ligações físicas são feitas com cabo flexível multi-condutor de 0,4'' de diâmetro. UL- CL2. O comprimento máximo por segmento do cabo é de 25 m. Podem ser usados comutadores e repetidores para permitir aumentar o diâmetro da rede. Adaptadores para ligações até 2 Km em fibra óptica estão a ser desenvolvidos.

O protocolo de ligação de dados permite controlo de fluxo, *packet framing* e controlo de erros. Apresenta uma taxa de erros muito baixa e é muito robusta no que concerne às ligações *host* – comutador e falhas nos cabos.

No caso de haver falhas nos cabos ou nos comutadores são automaticamente utilizadas rotas alternativas se estiverem disponíveis para contornar os problemas. O hardware processa e verifica um CRC para cada pacote em cada link.

As placas de rede *Myrinet/Sbus* incluem um processador *onboard* que executa um programa de controlo que mapeia e monitoriza a rede continuamente, e traduz os endereços usados pelas máquinas (como endereços IP) e as rotas usadas pelos comutadores Myrinet.

Um comutador Myrinet é um pequeno componente, com pequeno consumo de potência e *software* de auto-inicialização. Actualmente existem comutadores de 4 ou 8 portas. A figura 3.4.a. mostra um comutador de 8 portas. Comutadores de 16 e 32 portas estão em desen-volvimento.

Devido ao facto de poder transportar pacotes de qualquer tamanho, a Myrinet suporta IP directamente sem nenhuma camada de adaptação, e todos os protocolos acima do IP, tais como TCP e FTP.

Devido à sua elevada largura de banda, baixa latência e escalabilidade, a tecnologia Myrinet é particularmente interessante para interligar PCs em *clusters* usados para aplicações de computação paralela. Isto não surpreende, porque a tecnologia Myrinet desenvolveu-se com a tecnologia multi-computador. Esta tecnologia também é recomendada para construção de backbones, e para aplicações que utilizam muita largura de banda como transferência de imagens de elevada resolução ou vídeo.

Figura 3.4.a. – Comutador de 8 portas Myrinet

HIPPI

O acrónimo HIPPI significa *high-performance parallel interface* . A tecnologia HIPPI é um standard ANSI X3T11. Esta tecnologia funciona com velocidades de 800 Mbps ou 1.6 Gbps e foi desenvolvida para interligar *mainframes* ou supercomputadores em *cluster*. A versão paralela do HIPPI usa 50 pares de cabo STP com comprimentos até 25 m, enquanto o novo interface série permite ligações em fibra com conectores SC e distâncias até 300m.
Para esta tecnologia existe pouco *hardware* e suporte para *drivers* e só se refere no contexto deste trabalho por uma questão de complementação.
Contudo, de todas as tecnologias em análise, esta é a que apresenta maior largura de banda.

FDDI

O FDDI é uma tecnologia de rede a 100 Mbps, normalmente utilizada em backbones. Tem uma configuração em duplo anel e suporta vários modos de transmissão que são importantes para a comunicação de dados.
O Modo síncrono permite reserva de largura de banda e define um atraso máximo *end to end* para cada pacote de dados; o modo assíncrono não tem restrições temporais, é similar ao protocolo *Token Ring* e algumas implementações suportam apenas este modo.

Análise Comparativa

O principal elemento chave no desenvolvimento de novos *clusters* Beowulf reside na possibilidade de se utilizar redes de alta performance para servir de base de sustentação ao *cluster*. Recentemente emergiram várias tecnologias de rede *Gigabit* com suporte para Linux, abrindo novas perspectivas para a evolução e crescente expansão dos clusters Beowulf.

Tecnologia de Rede	Topologia	Largura de Banda	Latência	Meio Físico
Ethernet	BUS ou Estrela	10 Mbps	Alta	Cobre
Fast Ethernet	Estrela	100 Mbps	Média	Cobre
Gigabit Ethernet	Estrela	1 Gbps	Baixa	Fibra e cobre
Myrinet	Estrela	1.2 Gbps	≈5 µs	Cobre
SCI	Ponto-a-Ponto	1.6 Gbps	Muito baixa ≈600 ns	Cobre
HIPPI	Ponto-a-Ponto	800 Mbps ou 1.6 Gbps	Baixa	Fibra e Cobre
ATM	Estrela	25 ou155 Mbps	Baixa ≈10 µs	Fibra e cobre
FDDI	Duplo Anel	100 Mbps	Média	Fibra

Tecnologia de Rede	Suporte para Linux	Protocolos	Equipamento Activo	Preço
Ethernet	Sim	CSMA/CD	Comutadores e concentradores	Baixo
Fast Ethernet	Sim	CSMA/CD	Comutadores e concentradores	Médio
Gigabit Ethernet	Sim	CSMA/CD	Comutadores	Alto

Myrinet	Sim	Propriétário	Comutadores	Alto
SCI	Não	Propriétário	Comutadores	Alto
HIPPI	Sim	Propriétário	Comutadores	Alto
ATM	Não	ATM – AAL	Comutadores	Alto
FDDI	Não	*Timed Token Rotation*	Comutadores	Médio

CAPÍTULO IV

SOFTWARE

Neste capítulo pretende-se oferecer uma panorâmica do *software* mais usualmente utilizado nos clusters Beowulf, que não é de forma alguma exaustiva. Para o efeito referencia-se brevemente as ferramentas que podem ser utilizadas para gestão dos recursos do sistema e escalonamento, programação e *debug* em ambiente paralelo e sistema operativo utilizado. Descreve-se também com algum detalhe duas ferramentas desenvolvidas especificamente para clusters Beowulf com o objectivo de criar uma imagem única de sistema: o PFVS e o BPROC que se podem considerar parte da camada de Middleware anteriormente descrita. Por fim, refere-se uma aplicação de gestão da totalidade do cluster de uma forma integrada, o SCMS.

A quase totalidade do software descrito é de domínio público e encontra-se sob a GPL – GNU Public License. Para facilitar a consulta de mais informação, nomeadamente na *World Wide Web*, inclui-se em cada subcapítulo um conjunto de referências e apontadores para informação acerca dos assuntos tratados.

Apesar de todo o *software* se encontrar disponível de forma separada existem algumas colecções de ferramentas que incluem praticamente tudo aquilo que é necessário para instalar um cluster Beowulf, como por exemplo o pacote Beowulf incluído na distribuição SuSE do Linux.

Referências:

- GNU - http://www.gnu.org/;
- SuSE - http://www.suse.de/en/index.html.

4.1. SISTEMA OPERATIVO

O sistema operativo constitui a base de todo o cluster em termos de *software*, sobre o qual todos os restantes módulos vão funcionar.

Os clusters Beowulf utilizam usualmente o sistema operativo Linux. Trata-se de um sistema operativo fiável, de bom desempenho, gratuíto e *open source*. Consiste num kernel monolítico com suporte para multiprocessador, multitask e multiutilizador.

Todas estas características são essenciais quando se pretende construir um sistema computacional ao mesmo tempo de custo reduzido e de elevado desempenho. O facto de ser *open source* tem permitido uma rápida evolução efectuada por um grande número de programadores em todo mundo. Este desenvolvimento distribuido em larga escala tem gerado um enorme conjunto de aplicações e ferramentas de desenvolvimento poderosas. O controlo de qualidade é mantido pelo facto de as novas versões de kernel serem disponibilizadas a partir de um único ponto.
O Linux corre em plataformas de baixo custo como os PCs e oferece o desempenho e fiabilidade do Unix. Pode ser obtido a partir da internet sem custos adicionais e pode ser modificado conforme as necessidades do cluster. Além disso, as aplicações escritas em ambiente Unix são facilmente transpostas para Linux.

Referências - Linux:

- http://www.linux.com/;
- http://www.redhat.com;
- http://www.suse.de/en/index.html.

4.2. GESTÃO DE RECURSOS E ESCALONAMENTO

A gestão de recursos e o escalonamento (GRE) consiste em distribuir os processos pelos diversos nós de forma a minimizar o tempo total de execução e equilibrar a carga dos nós. A gestão de recursos encarrega-se da localização de recursos, da autenticação, e da criação e migração de processos. O escalonador encarrega-se da gestão da fila de processos e da atribuição de recursos aos processos.

A arquitectura básica da GRE é o modelo cliente-servidor. Cada nó que partilha recursos computacionais corre um processo servidor que mantém informação acerca dos recursos que disponibiliza e que é responsável pela gestão e escalonamento das tarefas. Para submeter uma tarefa é necessário utilizar os processos cliente, que interagem com os servidores de recursos, indicando por exemplo a localização dos ficheiros executáveis, os dados de entrada, a localização de destino dos dados de saída, entre outros parâmetros. A GRE oferece serviços de *Middleware* que incluem:

- Migração de processos: deslocamento de processos entre nós;
- *Checkpointing*: garante a fiabilidade armazenando o estado do processo em certos instantes de forma a ser possível retomar a sua execução em caso de falha;

- Distribuição de Carga: determinação da melhor forma de distribuir os processos ao longo do sistema de forma a garantir uma utilização equilibrada dos diversos nós.

Referências – Sistemas de GRE de domínio público:

- CONDOR – www.cs.wisc.edu/condor/;
- GNQS – www.gnqs.org/;
- DQS – www.scri.fsu.edu/~pasko/dqs.html.

4.3. ESPAÇO DE PROCESSOS DISTRIBUIDO

Neste capítulo abordar-se-á o pacote de *software* BPROC – *Beowulf distributed PROCess space*. O BPROC tem como objectivo criar a imagem de sistema virtual único para um cluster Beowulf. Para o efeito oferece mecanismos para gerir processos nos diversos nós do sistema através do *front-end* do cluster sem ser necessário efectuar *remote shell* (rsh) nem *remote login* (rlogin).
O espaço de processos distribuído permite representar processos de qualquer nó na tabela de processos do *front-end*, de forma a ser possível comunicar com esses processos como se de processos locais se tratassem. Salienta-se que o BPROC não se ocupa da gestão de recursos nem da distribuição de carga. O seu único objectivo consiste em criar um espaço de processos único para todo o cluster Beowulf.

As máquinas que pretendem distribuir o seu espaço de processos devem correr um *master daemon*. As máquinas que aceitam processos de outros nós devem correr um *slave daemon*. Cada nó pode correr múltiplos *slave daemon* mas apenas um *master daemon*. Ou seja, cada máquina pode contêr processos em diversos espaços diferentes mas apenas pode gerir um espaço de processos. Num cluster Beowulf o servidor ou *front-end* deve correr um *master daemon* e os restantes nós devem correr *slave daemons*.

Existem diversas primitivas que permitem iniciar um novo processo em qualquer nó a partir de um processo em qualquer outro nó:

- bproc_rexec: permite iniciar um processo num nó remoto desde que o executável se encontre nesse nó. Portanto, requer a instalação prévia do executável nesse nó. A sintaxe desta função é a seguinte:

```
bproc_rexec( node, file, argv, envp );
```

Os argumentos identificam o executável (file), o nó onde se pretende iniciar o processo (node), os parâmetros de entrada (argv) e as variáveis de ambiente (envp).

• bproc_move: permite deslocar o processo que invoca esta primitiva para um determinado nó especificado (node):

```
bproc_move( node, flags );
```

Esta primitiva tem a vantagem de transportar o executável para o nó de destino de uma forma automática, o que reduz o espaço ocupado em disco nos nós mas apresenta a desvantagem do *overhead* que resulta da utilização da rede de comunicações para transferir o executável.

• bproc_rfork: permite criar um processo filho do processo que a invoca num determinado nó (node) seleccionado. A sua sintaxe é idêntica à de fork( ):

```
if ( bproc_rfork( node, flags ) == 0)
{
        // processo filho a ser executado
        // pelo nó remoto
} else {
        // processo pai
}
```

O BPROC consiste em três componentes básicos, nomeadamente os Processos *Ghost* no *front-end* , o Mascaramento de PID e os *Daemons*, que são descritos de seguida.

Processos Ghost

Estes processos representam, no *front-end*, os processos remotos, que correm nos restantes nós sob a monitorização dos *slave daemons*. Estes processos não têm espaço de memória, ficheiros associados nem contexto. No entanto, possuem *signal handlers* e filas de mensagens. A sua funcionalidade é a seguinte:

- Encaminham os sinais que recebem para os respectivos processos remotos que representam;

- Efectuam fork( ) a pedido do respectivo processo remoto. Quando um processo remoto pretende efectuar fork( ) é necessário obter o PID do processo filho a ser criado a partir do *master daemon*, que é quem gere o espaço de processos distribuido. Isto é conseguido através do seu processo *ghost* , que efectua fork ( ) e retorna o PID do filho ao nó remoto. Este PID será atribuído ao processo filho remoto através do Mascaramento de PIDs efectuado pelo *slave daemon* no nó de destino;
- Sempre que um processo efectua wait( ) o respectivo *ghost* deve fazer o mesmo para impedir a acumulação de processos *ghost zombies* na árvore de processos;
- Sempre que um processo remoto efectua exit( ) o seu código de terminação é enviado ao respectivo *ghost* que também efectua exit( ) com o mesmo código de terminação, o que permite que wait( ) no processo pai remoto receba a informação acerca do término do respectivo processo filho (que poderia encontrar-se a correr em qualquer outro nó do sistema).

Em resumo, estes processos espelham o estado dos processos remotos que representam. A informação acerca desses processos remotos á apresentada no /proc *file system* em vez da informação acerca dos próprios processos *ghost*, que são invisíveis para o utlizador. Essa informação é actualizada a pedido (por consulta de /proc utilizando o comando *top* por exemplo) e também periodicamente. Neste sentido, /proc passa a ser designado *Unified /proc file system*.

Mascaramento de PID

Os nós remotos correm uma parte da árvore de processos do *front-end*. Para além desses processos podem existir outros processos iniciados por rsh por exemplo, que devem coexistir com os processos remotos controlados pelo BPROC, iniciados utilizando as primitivas atrás indicadas. Quando se desloca um processo de um nó para outro o seu PID não deve mudar devido às dependências que existem entre os diversos processos. Para que tal seja possível o *slave daemon* cria e controla um espaço de processos local que é um subconjunto do espaço de processos do *master daemon*. Neste subespaço, os processos têm o mesmo PID dos processos *ghost* correspondentes – isto designa-se de Mascaramento de PID. Podem eventualmente existir outros processos no nó associados a diferentes *daemons* e consequentemente a diferentes espaços de processos que não interferem entre si.

Daemons

O *master daemon* armazena a informação acerca da configuração dos nós e conhece a localização de todos os processos no seu espaço de processos. É responsável pela criação da imagem de uma única árvore de processos.
Os *slave daemons* actuam como intermediários entre os processos remotos e o *master daemon*. Representam um subconjunto do espaço de processos global e efectuam o Mascaramento de PID nesse subespaço.

A figura 4.3.a. ilustra o conceito do BPROC.

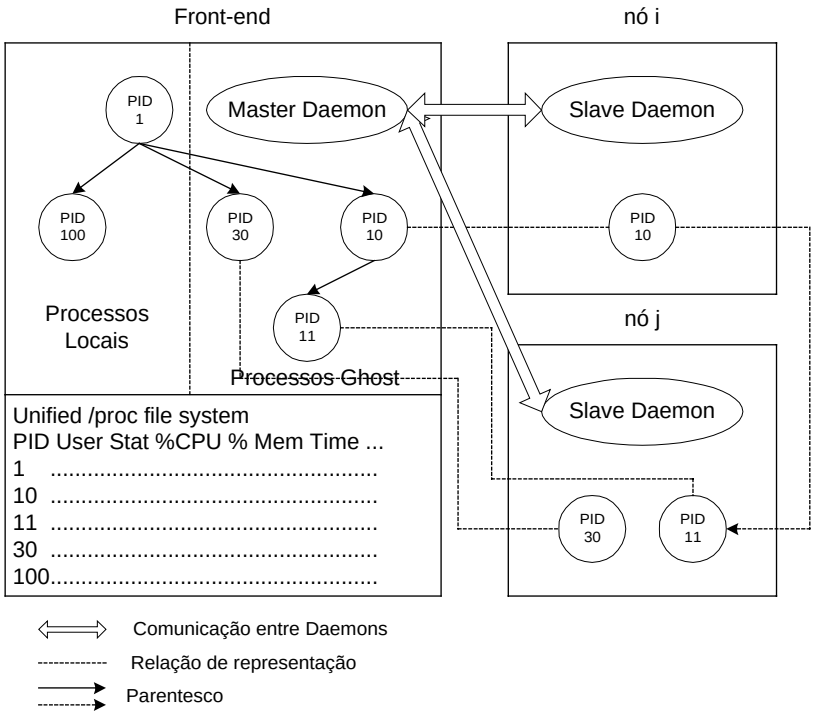


Figura 4.3.a. - BPROC

Referências – BPROC:

- http://www.beowulf.org/software/bproc.html.

4.4. PARALLEL VIRTUAL FILE SYSTEM

O desenvolvimento do *Parallel Virtual File System* (PVFS) deve-se à crescente disparidade que se tem verificado nos últimos anos entre a evolução do desempenho dos dispositivos de

armazenamento de dados e dos processadores, sendo notoriamente superior neste últimos, o que origina um ponto de estrangulamento naqueles dispositivos especialmente em aplicações que manipulem grandes conjuntos de dados. Com o objectivo de minimizar este problema surgiu o projecto PVFS na NASA, intimamente ligado ao desenvolvimento do Beowulf.

A ideia base consiste em dividir um ficheiro em diversas partições e armazenar essas diferentes partições ao longo de diversos nós do sistema de forma a conseguir uma capacidade de input/output agregada sobre o ficheiro múltipla da que estaria disponivel se o ficheiro se encontrasse armazenado num único nó. Desta forma é possível paralelizar as operações de leitura e de escrita sobre um ficheiro.
O PVFS é utilizado em ambiente Linux sobre o protocolo de transporte TCP. Actualmente encontra-se implementado em *user-level* pelo que não é necessário modificar o kernel. Suporta a interface de input/output do Unix e inclui uma biblioteca partilhada que permite a portabilidade dos programas escritos para ambiente Unix sem necessidade de recompilação. Os comandos da *shell*, como *ls*, *cp* e *rm* podem ser utilizados nos ficheiros e directórios abrangidos pelo PVFS. Oferece ainda uma interface nativa que permite um maior controlo sobre os parâmetros que regem o sistema de ficheiros paralelo.

Os nós do sistema de ficheiros paralelo podem ser de três tipos:

- De Gestão: corre um *daemon* de gestão que gere os acessos aos ficheiros com base em permissões e mantém metadados acerca dos ficheiros. Actualmente o PVFS utiliza o NFS para obter um único espaço global de i-nodes visível a todos os nós;
- De I/O (input/output) ou De Armazenamento: são utilizados para armazenar os ficheiros. O acesso aos ficheiros é efectuado pelo I/O *daemon* utilizando as primitivas read( ) e write( ) do Linux;
- De Processamento: correm as aplicações paralelas que utilizam a biblioteca de funções que permite comunicar com os *daemons* de gestão e I/O.

Usualmente existe apenas um nó de gestão, o *front-end* do cluster, sendo todos os restantes nós de processamento e I/O em simultâneo. A figura 4.4.a. representa o relacionamento entre os diversos nós.

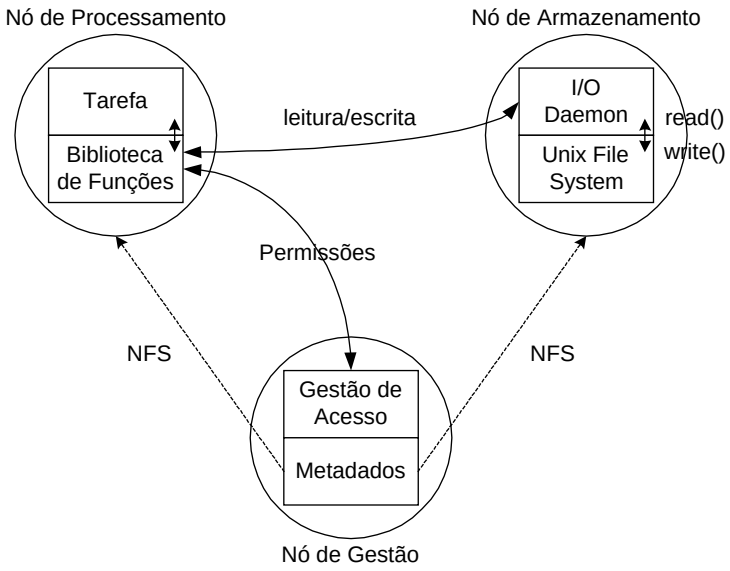


Figura 4.4.a. – Arquitectura do PVFS

Como já se referiu, cada ficheiro a ser armazenado é fragmentado em diversas partições que são armazenadas ao longo de diversos nós. Quando a aplicação necessita de determinados segmentos de um determinado ficheiro envia pedidos de leitura aos diversos nós de armazenamento que contêm esse ficheiro. Em cada nó o I/O daemon respectivo efectua a intersecção entre as partições do ficheiro localmente armazenadas e os segmentos pretendidos pela aplicação enviando de seguida o resultado ao nó de processamento que efectuou o pedido. A figura 4.4.b. ilustra este procedimento num nó de armazenamento, 1, que envia o resultado ao nó de processamento 2.

As características da *stream* de dados entre nós de I/O e nós de processamento são determinadas por negociação entre o I/O *daemon* e a interface de programação no nó de processamento na fase inicial da ligação de forma a minimizar o *overhead* de informação de controlo.

Para as aplicações que utilizam as funções de I/O de Unix, as características de fragmentação de ficheiros são as escolhidas por omissão durante a instalação do PVFS. A figura 4.4.c. representa a fragmentação efectuada por defeito para um sistema com n nós de armazenamento.

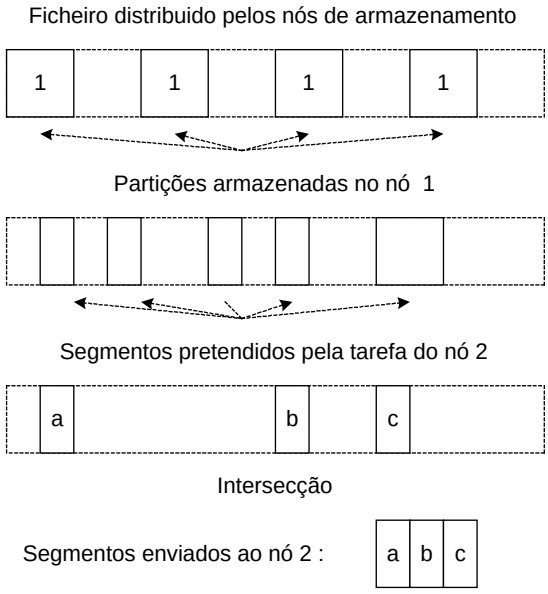


Figura 4.4.b. – Leitura de ficheiros com PVFS

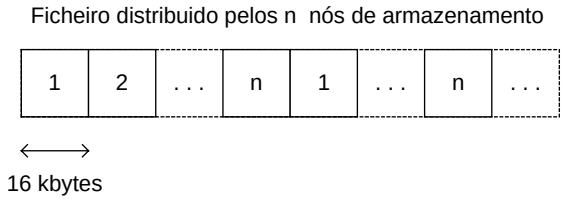


Figura 4.4.c. – Fragmentação utilizando I/O de Unix

Por outro lado, as leituras são indicadas por um offset e número de bytes a ler, conforme indicado na figura 4.4.d..

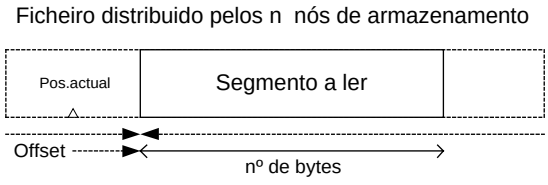


Figura 4.4.d. – Leitura utilizando I/O de Unix

Como alternativa as aplicações podem utilizar as funções de I/O nativas do PVFS. Desta forma é possível especificar qual o nó de I/O onde se inicia o armazenamento do ficheiro (base), o número de nós de I/O a começar em “base” ao longo dos quais o ficheiro vai ser distribuido de uma forma circular (pcount), e o tamanho de cada fragmento (ssize). A figura 4.4.e. ilustra este procedimento para base igual a 5 e pcount igual a 3.

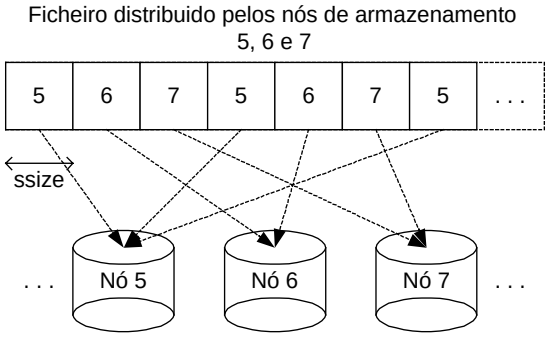


Figura 4.4.e. – Fragmentação nativa de PVFS

A estrutura pvfs_stat deve ser preenchida com os parâmetros atrás indicados sendo depois utilizada na função pvfs_open, que efectua a fragmentação do ficheiro:

```
struct pvfs_stat {
        int base;
        int pcount; // -1 utiliza todos os nós I/O
        int ssize;
}

pvfs_open( char *pathname, int flag, mode_t mode, struct pvfs_stat *dist );
```

Os restantes parâmetros de pvfs_open são idênticos a open( ) de Unix. Para conhecer a distribuição física de um ficheiro é possível utilizar a seguinte função:

```
pvfs_ioctl( int filedescriptor, GET_META, struct pvfs_stat *dist );
```

A aplicação pode também utilizar a função pvfs_ioctl para escolher os segmentos do ficheiro de uma forma mais granular que o que as funções de Unix permitem. A estrutura fpart permite escolher o desvio a partir do início do ficheiro (offset), o número de bytes consecutivos de cada segmento (gsize) e o número de bytes entre o início de dois segmentos consecutivos (stride), conforme ilustrado pela figura 4.4.f.:

```
struct fpart {
        int offset;
        int gsize;
        int stride;
}

pvfs_ioctl( int filedescriptor, SET_PART, struct fpart *part );
```

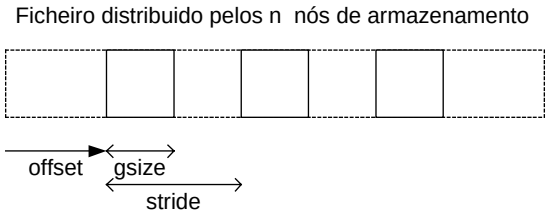


Figura 4.4.f. – Leitura nativa de PVFS

Referências – PVFS:

- http://ece.clemson.edu/parl/pvfs/.

4.5. AMBIENTES DE PROGRAMAÇÃO

A comunicação entre os nós dos clusters pode ser efectuada de duas formas diferentes: utilizando o *software* Distributed Inter-Process Communication (DIPC) ou *software* de *message passing* como o PVM (Parallel Virtual Machine) e o LAM (Local Area Multicomputer).

Distributed Inter-Process Communication

Consiste numa implementação dos mecanismos de comunicação entre processos (IPC) do System V, nomeadamente semáforos, filas de mensagens e memória partilhada, modificados de forma a funcionar em ambiente multicomputador. A sintaxe é análoga à do System V de forma a garantir a portabilidade das aplicações. As suas funcionalidades são garantidas por um *daemon* (dipcd) em *kernel space*, o que obriga a uma recompilação do núcleo do Linux de forma a incluir o DIPC.

A Distributed Shared Memory (DSM) consiste na implementação de memória partilhada do DIPC, que utiliza a semântica múltiplos leitores – escritor único, e um modelo de consistência *strict consistency* que garante que o resultado de uma leitura é sempre o valor mais actual. No entanto este modelo origina diminuição de desempenho.
Apesar de o DIPC se encontrar integrado no kernel, o seu desempenho é actualmente inferior aos mecanismos de *message passing*. Note-se que o DIPC baseia-se num mecanismo de troca de mensagens subjacente uma vez que os clusters são sistemas multicomputador.

Referências – DIPC:

- http://wallybox.cei.net/dipc/.

Parallel Virtual Machine

O PVM tem como objectivo utilizar um conjunto de nós de um cluster para executar uma determinada aplicação como se estes se tratassem de uma única máquina virtual. Utiliza o paradigma de comunicação *message passing*. As aplicações são encaradas como uma colecção de tarefas colaborantes que utilizam as bibliotecas de funções do PVM para sincronização e comunicação. A transparência é conseguida através do Identificador de Tarefa (TID) único para cada tarefa de cada máquina virtual. O PVM tem as seguintes características:

a) O conjunto de nós que constitui a máquina virtual é escolhido pelo utilizador;
b) O modelo de mensagens é explicito (ao contrário do DIPC);
c) Permite Heterogeneidade de arquitecturas de *hardware* (desde PCs até máquinas multiprocessador), formato de dados e velocidade de processamento.

O *software* que integra o PVM divide-se em duas partes: um *daemon* (pvmd3) colocado em todos os nós do cluster e uma biblioteca de funções de interface utilizadas pelas aplicações (libpvm3.a).
Cada utilizador pode criar uma máquina virtual PVM que englobe qualquer conjunto de nós do cluster, ou seja, em simultâneo podem existir diversas máquinas virtuais a correr nos mesmos nós do cluster. O ficheiro ".rhosts" identifica os restantes servidores da máquina virtual (*daemons* nos outros nós) e autoriza o acesso a esses servidores.
O PVM utiliza os seguintes mecanismos de comunicação entre processos:

- Sockets UDP para comunicação entre os *daemons*. O protocolo de transporte UDP oferece um serviço de mensagens não fiável pelo que esta tem de ser garantida ao

nível da camada protocolar constituída pelo PVM. O TCP não é utilizado devido ao elevado número de conexões que seria necessário estabelecer entre os diversos nós (N(N-1)/2 para N nós ou seja $O(N^2)$);
- Sockets TCP para comunicação entre tarefas e entre um *daemon* e uma tarefa, que oferecem um serviço *stream oriented* fiável.

Pode ainda utilizar Unix sockets para comunicações locais, o que diminui a latência.

Existem duas técnicas de programação:

- MIMD (Multiple Instruction Multiple Data): diversos blocos de instruções diferentes são aplicados a diferentes conjuntos de dados;
- SPMD (Single Program Multiple Data): um único bloco de instruções é simultaneamente aplicado a múltiplos conjuntos de dados (os processadores podem no entanto seguir diferentes ceminhos de acordo com as instruções de controlo de fluxo do programa).

A técnica utilizada depende da aplicação. SPMD é eficiente quando a mesma parte de um determinado trabalho computacional pode ser paralelizada. MIMD destina-se a ter diferentes processadores a executar tarefas distintas.

O XPVM é um interface gráfico para ambiente X-Windows que permite monitorizar o desempenho e debug ao nível das chamadas do PVM. Foi desenvolvido em Tcl/Tk e é distribuído livremente sob a GPL.

Referências – PVM:

- http://www.netlib.org/pvm3/book/pvm-book.html;
- http://www.netlib.org/pvm3/index.html;
- http://www.epm.ornl.gov/pvm/pvm_home.html .

Local Area Multicomputer

O LAM implementa o standard MPI (Message Passing Interface) de comunicação por mensagens, que surgiu no âmbito de um esforço de normalização de diversas interfaces de programação proprietárias de forma a permitir a portabilidade de aplicações entre os diversos sistemas. Desta forma um programa escrito numa arquitectura pode ser transposto para outra

bastando efectuar nova compilação. No, entanto, ao contrário do PVM, este princípio não se extende à possibilidade de integração e cooperação de diferentes equipamentos de *hardware* nem à possibilidade de interacção entre aplicações escritas em diferentes linguagens de programação. O MPI não tem presente o conceito de máquina virtual do PVM. No entanto o desempenho constituiu uma preocupação central no seu desenvolvimento, pelo que teoricamente é mais rápido e eficiente que o PVM. Por este motivo, se a arquitectura é homogénea como acontece com os clusters Beowulf a melhor forma de comunicação é o LAM.

Referências – MPI:

- http://www.erc.msstate.edu/labs/hpcl/projects/mpi/;
- http://www.mpi.nd.edu/lam/

.

4.6. COMPILAÇÃO E DEBUG

Não existem ainda compiladores que paralelizem automaticamente o código de uma aplicação de forma a ser executado em paralelo num cluster. No entanto existem algumas ferramentas que auxiliam a paralelização manual do código. Estas ferramentas destinam-se essencialmente a aplicações escritas em Fortran (por exemplo o *software* BERT da Prologic).

Usualmente os fabricantes de sistemas computacionais HPC fornecem ferramentas de *debug* para os seus sistemas, no entanto, o número de *debuggers* capazes de serem utilizados numa arquitectura heterogénea é bastante limitado. Como exemplo refere-se o TotalView, que é um produto comercial da Dolphin Interactive Solution, actualmente o único que suporta múltiplas plataformas de HPC, sistemas operativos (SunOS, Solaris, AIX, Digital Unix, SGI Irix), linguagens de programação (C, C++, Fortran) e modelos de comunicação (PVM e MPI).
No entanto, apesar de poder correr em múltiplas plataformas o sistema operativo deve ser o mesmo em todas elas.

Referências:

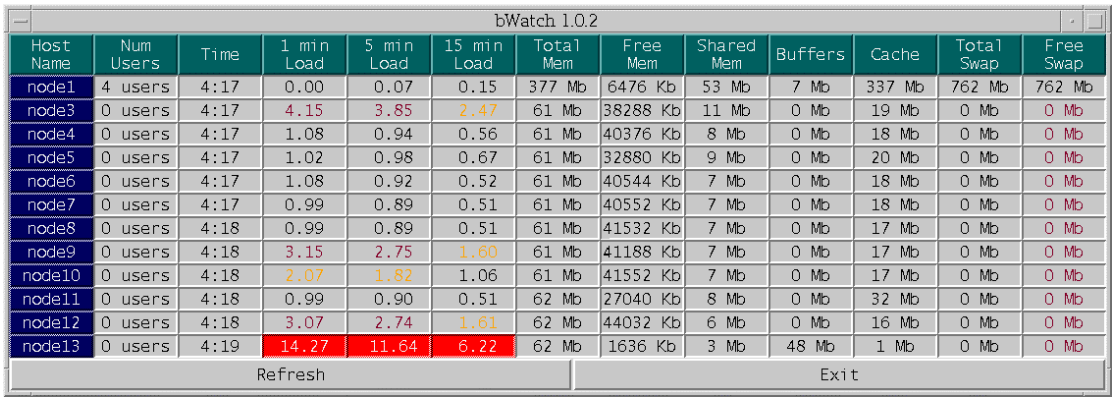
- Prologic – www.prologic.com;
- Dolphin Interconnect Solutions – www.dolphinics.com.

4.7. APLICAÇÕES DE GESTÃO DO CLUSTER

É necessário dispôr de bom software de gestão para poder explorar o cluster de uma forma eficiente. As ferramentas de gestão permitem monitorizar e configurar o cluster a partir de uma aplicação com interface gráfica amigável colocada no *front-end*.

bWatch

O bWatch é um programa escrito em Tcl/Tk que produz uma tabela com estatísticas de ocupação de processadores e memória em cada nó do sistema. É apenas uma ferramenta de monitorização e análise do desempenho do cluster. A figura 4.7.a. mostra a interface oferecida por esta aplicação. As referências contêm apontadores para este e outros programas do mesmo género.



Host Name	Num Users	Time	1 min Load	5 min Load	15 min Load	Total Mem	Free Mem	Shared Mem	Buffers	Cache	Total Swap	Free Swap
node1	4 users	4:17	0.00	0.07	0.15	377 Mb	6476 Kb	53 Mb	7 Mb	337 Mb	762 Mb	762 Mb
node3	0 users	4:17	4.15	3.85	2.47	61 Mb	38288 Kb	11 Mb	0 Mb	19 Mb	0 Mb	0 Mb
node4	0 users	4:17	1.08	0.94	0.56	61 Mb	40376 Kb	8 Mb	0 Mb	18 Mb	0 Mb	0 Mb
node5	0 users	4:17	1.02	0.98	0.67	61 Mb	32880 Kb	9 Mb	0 Mb	20 Mb	0 Mb	0 Mb
node6	0 users	4:17	1.08	0.92	0.52	61 Mb	40544 Kb	7 Mb	0 Mb	18 Mb	0 Mb	0 Mb
node7	0 users	4:17	0.99	0.89	0.51	61 Mb	40552 Kb	7 Mb	0 Mb	18 Mb	0 Mb	0 Mb
node8	0 users	4:18	0.99	0.89	0.51	61 Mb	41532 Kb	7 Mb	0 Mb	17 Mb	0 Mb	0 Mb
node9	0 users	4:18	3.15	2.75	1.60	61 Mb	41188 Kb	7 Mb	0 Mb	17 Mb	0 Mb	0 Mb
node10	0 users	4:18	2.07	1.82	1.06	61 Mb	41552 Kb	7 Mb	0 Mb	17 Mb	0 Mb	0 Mb
node11	0 users	4:18	0.99	0.90	0.51	62 Mb	27040 Kb	8 Mb	0 Mb	32 Mb	0 Mb	0 Mb
node12	0 users	4:18	3.07	2.74	1.61	62 Mb	44032 Kb	6 Mb	0 Mb	16 Mb	0 Mb	0 Mb
node13	0 users	4:19	14.27	11.64	6.22	62 Mb	1636 Kb	3 Mb	48 Mb	1 Mb	0 Mb	0 Mb

Figura 4.7.a. – Interface do bWatch

Referências:

- http://www.sci.usq.edu.au/staff/jacek/bWatch/.

SCMS

O SCMS – SMILE Cluster Management System – foi concebido de forma a simplificar as tarefas de gestão de clusters Beowulf de elevada dimensão. É constituido pelos seguintes componentes:

- Camada de *software* que implementa uma arquitectura do tipo cliente-servidor que recolhe e centraliza num único ponto os parâmetros de desempenho da totalidade do cluster;
- Interface de programação que permite a programas escitos em C, Java ou Tcl/Tk aceder à informação recolhida;
- Componente designado de *Parallel Unix Command* que implementa comandos com funcionalidades idênticas a ps, cp, rm e outros da shell de Unix mas sobre os recursos distribuidos do sistema;
- Sistema de Alarmes que permite a definição de condições associadas a certos eventos que originam alarmes. É possível gerar alarmes de acordo com limites estabelecidos para a ocupação dos processadores, da memória, a temperatura e muitas outras variáveis. Permite ainda definir acções de resposta à ocorrência desses alarmes. Inclui módulos de síntese de voz e interface com a rede telefónica de forma a ser possível sinalizar alarmes através deste meio.
- Interface gráfica com os seguintes componentes:
  - Sistema de Gestão: permite conhecer o estado dos nós, espaço em disco, ficheiros, utilizadores, efectuar reboot, shutdown, seleccionar acções e nós onde se pretende actuar;
  - Sistema de Monitorização: permite a monitorização dos nós em tempo real;
  - Interface para o *Parallel Unix Command*;
  - Interface para configurar o sistema de alarmes.

A figura 4.7.b. mostra diversas componentes da interface gráfica.

Figura 4.7.b. – Interface do SCMS

Referências:

- http://smile.cpe.ku.ac.th/software/scms/.

CAPÍTULO V

IMPLEMENTAÇÕES

Neste capítulo apresenta-se resumidamente alguns cluster Beowulf, com especial incidência no *hardware* e *software* utilizados e no próprio aspecto físico que estes clusters tipicamente assumem. Em particular, para o cluster Vitara apresenta-se uma análise de custos.
Todos os clusters apresentados encontram-se na lista de clusters Beowulf do CESDIS (http://www.beowulf.org/).

5.1. MAGI

As figuras 5.1.a. e 5.1.b. mostram a constituição física do cluster acondicionado em prateleira.

Figura 5.1.a. - Parte da frente do *cluster*

Figura 5.1.b. - Parte de trás do *cluster*

Aplicação:

Máquina multi-utilizador para experiências em reconhecimento de voz.

Hardware:

Sem disquetes, CDROM ou rato. A única ligação com o exterior são dois cabos: cabo de alimentação e cabo de rede a 100 Mbps.

Os oito nós (clientes e servidor) consistem em:

- *CPU* Intel Pentium Pro a 200 MHz com cache nível 2 integrada de 256K, com um grande refrigerador passivo;
- *motherboard* Intel VS440FX (Venus) com Intel 82440FX PCIset;
- 64MB de *RAM* em 2 *SIMMs* de 32MB *EDO RAM* 60ns 72pin 32bit;
- 2 discos *IDE* de 4,3GB (IBM DCAA-34330 alias Capricorn) em prateleiras removíveis;
- adaptador *Fast Ethernet* 100Mb/s 3c905 (alias Etherlink XL 10/100 TX PCI alias Boomerang);
- placa *SVGA* (*cheap PCI* 1MB S3 trio 64V+);
- Caixas ATX (4 *Minitowers* Intel CC5 200W e 4 *Bigtowers* 235W).

Rede de dados:

- comutador a 100Mb/s 3Com SuperStack II Switch 3000 usado para a interligação dos nós;

Componentes adicionais:

- 2 APC Smart-UPS 700;
- um teclado e um pequeno monitor ligados através de comutadores a qualquer nó;
- *hardware* usado para ligar/desligar os nós por *software*.

Software:

- Red Hat Linux 4.0, usado como sistema operativo nos nós individuais;
- *Software* especifico usado para ligar os nós, que inclui:
    - definição da identidade do nó (*hostname, IP*) na altura do *boot*;
    - criação do mapa do sistema de ficheiros distribuídos quando os discos duros são movidos;
    - sistema de *Job Spooling*;

- *scripts* a substituir vários programas por versões paralelas;
- ferramentas especiais de gestão do *cluster*.

- HTK (*hidden Marcov model toolkit*), que pode ser convertido em processamento paralelo usando umas *scripts* simples que substituem programas originais do HTK;
- Matlab, que não pode ser convertido da mesma forma, mas pode ser substituído por uma *script* que dedica toda a potência de um nó a cada processo Matlab.

Performance:

A largura de banda da rede é 81*10^6bits/Seg na comunicação entre nós, usando o teste *netperf.*

5.2. HIVE

A figura 5.2.a. mostra o cluster acondicionado em três bastidores. Pode observar-se a interligação dos nós e os sistemas de refrigeração.

Figura 5.2.a. - Interior das três partes constituintes do Hive

Aplicação**:**

Aplicações multi-utilizador para ciências terrestres e espaciais na NASA:

- Segmentação de imagens;
- Uso de PPM (*Piecewise Parabolic Method*): resolve equações de *Euler*;
- *Adaptative Mesh Refinement* paralelo: estende o código série existente em código paralelo com AMR (*Adaptative Mesh Refinement*);
- Diagnósticos plasmáticos e Interferometria electromagnética;
- *Cosmological Particle-Mesh N-body Code*: resolve equações de *Poisson* para estimar acelerações de partículas.

Hardware:

O Hive consiste em 4 *subclusters* (E, B, G, DL), contendo 332 processadores:

- 3 comutadores de rede com 72 portas *fast ethernet* e 8 portas *gigabit ethernet* cada. 4 portas das *gigabit ethernet* estão truncadas umas com as outras para interligar os três comutadores;
- o *cluster* B, baseado no HIVE original, consiste em:
  - 2 nós servidores Dual Pentium Pro;
  - 66 *PCs* Dual Pentium Pro (132 processadores);
  - um comutador *fast ethernet* de 72 portas;
  - 28 GBytes de *RAM*;
  - 900 GBytes de disco.
- o *cluster* E, baseado num Beowulf da CESDIS, consiste em:
  - 63 Dual Pentium Pro PCs (128 processadores);
  - um comutador *fast ethernet* de 72 portas;
  - 8 GBytes de *RAM*;
  - 1.2 TBytes de disco.
- o *cluster* G consiste em:
  - 16 *PCs* servidores de ficheiros Dual Pentium III Xeon (32 processadores);
  - ligados a um comutador *fast ethernet* de 72 portas e Myrinet;
  - 8 Gbytes de *RAM*;
  - 72 Gbytes de disco.
- o *cluster* DL consiste em:
  - 10 *PCs* servidores de ficheiros Quad Pentium III Xeon (40 processadores);
  - ligados a um comutador *fast ethernet* de 72 portas e Myrinet;
  - 5 Gbytes de *RAM*;
  - 45 Gbytes de disco.

Software:

- Sistema Operativo: Red Hat Linux;
- Linguagens: C, C++, aCe, Fortran77;
- *Software* de comunicação: PVM, MPI, BSP.

Performance:

- Entre um par de processadores (em um único comutador, sem mais tráfico de comunicações):

A taxa máxima de comunicações é 11.32Mbytes/seg/processador.

A taxa média de comunicações para todos os pares é 11.25Mbytes/seg/processador.

- Entre oito pares de processadores (simultaneamente, num único comutador, sem mais tráfego de comunicações):

A taxa máxima de comunicações é 11.30Mbytes/seg/processador.

A taxa média de comunicações para todos os pares é 11.25Mbytes/seg/processador.

- Entre dezasseis pares de processadores (simultaneamente, num único comutador, sem mais tráfego de comunicações):

A taxa máxima de comunicações é 11.35Mbytes/seg/processador.

A taxa média de comunicações para todos os pares é 10.1Mbytes/seg/processador.

- Entre comutadores (16 processadores num comutador comunicando com 16 processadores noutro, sem mais tráfego de comunicações):

A taxa média de comunicações entre comutadores adjacentes (C0-C1 ou C3-C4) é 998Kbytes/seg/processador.

A taxa média de cada par de comutadores de 127Mbits/seg.

5.3. VITARA

A figura 5.3.a. mostra o cluster Vitara em bastidor e o pormenor da ligação dos nós.

Figura 5.3.a. – Parte frontal e traseira do cluster

Aplicação:

O VITARA está inserido num projecto académico, de análise de viabilidade económica, mas para testes foram desenvolvidas as seguintes aplicações:

- *Rendering* de imagens;
- *Webserver.*

Arquitectura:

A figura 5.3.b. mostra a arquitectura do Vitara. A arquitectura física efectivamente utilizada está representada a tracejado com a interligação dos nós através de um comutador.

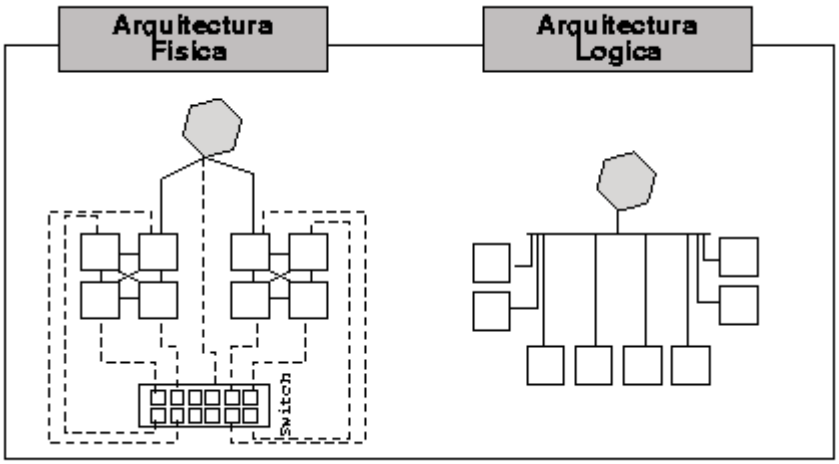


Figura 5.3.b. - Arquitectura do cluster Vitara

O VITARA usa configuração de clientes sem disco rígido.

Hardware:

- 1 bastidor;
- 8 nós clientes Pentium II 350MHz, sem disco;
- 1 nó servidor Pentium II 450MHZ;
- 1 comutador com interfaces a 100Mbps;
- 10 placas de rede a 100 Mbps;
- 1 leitor de *CDROM*;
- 9 *drives* de disquetes;
- 1 monitor.

Software:

- Sistema operativo no servidor: *S.u.S.E. 6.0*;
- Sistema operativo nos clientes: Slackware (mais flexível e mais simples);
- Software de comunicação: LAM.

Problemas:

Alguns problemas iniciais no arranque do cluster vieram a revelar-se consequência de sobreaquecimento dos processadores pelo que se procedeu à aquisição de componentes de refrigeração adicionais.

Preço:

O equipamento apresentado na tabela abaixo constituem o *cluster* implementado. Os valores são preços de Novembro de 1998.

Valorização dos componentes utilizados			
	N.Comp.	**Preço Unitário**	**Total**
Motherboard	9	17,600.00 PTE	158,400.00 PTE
Pentium II 350	8	39,100.00 PTE	312,800.00 PTE
Pentium II 450	1	10,820.00 PTE	10,820.00 PTE
Memória	10	32,900.00 PTE	329,000.00 PTE
Caixas Desktop	9	8,500.00 PTE	76,500.00 PTE
Leitor CDROM	1	7,900.00 PTE	7,900.00 PTE
Drive de 3 1/2	9	2,600.00 PTE	23,400.00 PTE
Placas de rede	10	7,500.00 PTE	75,000.00 PTE
Monitor	1	104,320.00 PTE	104,320.00 PTE
Rack	1	190,000.00 PTE	190,000.00 PTE
Switch	1	295,000.00 PTE	295,000.00 PTE
			1,583,140.00 PTE
		Iva 17%	269,133.80 PTE
			1,852,273.80 PTE

CONCLUSÃO

Os clusters Beowulf apresentam uma alternativa economicamente viável às tradicionais arquitecturas de processamento paralelo de elevado desempenho. Desde o seu aparecimento esta arquitectura tem vindo a tornar-se cada vez mais popular devido ao seu custo reduzido e enorme flexibilidade de evolução. É possível escolher a dimensão do cluster que mais se adequa a cada aplicação. Desta forma existem sistemas que vão de alguns nós até aos sistemas que possuem centenas de nós. O grau de granularidade de capacidade de processamento é superior ao de qualquer outra arquitectura uma vez que se pode escolher o exacto número de nós que se pretende. Existe um grande conjunto de arquitecturas de *hardware* e de ferramentas de *software* que permitem implementar clusters deste tipo. Existe ainda uma enorme quantidade de informação disponível na World Wide Web para quem pretender criar um sistema Beowulf. Além disso, uma vez que se baseia em componentes de *hardware* vulgares e *software* que é na sua totalidade gratuito, o risco de construir um destes clusters é mínimo.

A avaliar pela crescente popularidade do sistema operativo Linux, em que se baseia, certamente que nos próximos anos se observará um grande crescimento do número de sistemas Beowulf.

BIBLIOGRAFIA

Referem-se aqui os principais locais visitados durante a realização do trabalho. As referências indicadas podem ser úteis para quem estiver interessado por este tema. A totalidade das fontes de informação foram obtidas na World Wide Web. O capítulo 4 inclui referências específicas para os assuntos aí tratados para facilitar a consulta.

Livros

- High Performance Cluster Computing: Architectures and Systems, Vol. 1
  Rajkumar Buyya - Prentice Hall - PTR, NJ, USA, 1999
  http://www.phptr.com/ptrbooks/ptr_0130137847.html

Documentação

- Beowulf Homepage. (Página oficial do Beowulf no CESDIS).
  http://www.beowulf.org

- Última versão do Beowulf HOWTO
  http://www.sci.usq.edu.au/staff/jacek/beowulf

- Building a Beowulf System.
  http://www.cacr.caltech.edu/beowulf/tutorial/building.html

- Jacek's Beowulf Links
  http://www.sci.usq.edu.au/staff/jacek/beowulf

- Beowulf Installation and Administration HOWTO (DRAFT)
  http://www.sci.usq.edu.au/staff/jacek/beowulf

Artigos

- Chance Reschke, Thomas Sterling, Daniel Ridge, Daniel Savarese, Donald Becker, e Phillip Merkey. *A Design Study of Alternative Network Topologies for the Beowulf Parallel Workstation.* Apresentação no Quinto Simpósio Internacional do IEEE em Computação Distribuída de Alta Performance, 1996.
  http://www.beowulf .org/papers/HPDC96/hpdc96.html

- Daniel Ridge, Donald Becker, Phillip Merkey, Thomas Sterling Becker. *Harnessing the power of Parallelism in a Pile-of-Pcs*. IEEE Aerospace, 1997.
  http://www.beowulf.org/papers/AA97/aa97.ps

- Thomas Sterling, Donald j. Becker, Daniel Savarese, Michel R. Berry, e Chance Reschke.*Achieving a Balanced Low-Cost architecture for Mass Storage Management through Multiple Fast Ethernet Channels on the Beowulf Parallel Workstation.*
  Apresentação no Simpósio Internacional de Processamento Paralelo, 1996.
  http://www.beowulf.org/papers/IPPS96/ipps96.html

- Donald J. Becker, Thomas Sterling, Daniel Savarese, Bruce Fryxell, Kevin Olson. *Communication Overhead for Space Science Applications on the Beowulf Parallel Workstation.*
  High Performance and Distributed Computing, 1995.
  http://www.beowulf.org/papers/HPDC95/hpdc95.html

- Donald J. Becker, Thomas Sterling, Daniel Savarese, John E. Dorband, Udaya A. Ranawak, Charles V. Packer. *Beowulf: A Parallel Workstation for Scientific Computation*.
  Apresentação, Conferência Internacional sobre Processamento Paralelo, 95.
  http://www.beowulf.org/papers/ICPP95/icpp95.html

- Mais Papers nas páginas oficiais do Beowulf
  http://www.beowulf.org/papers/papers.html

Software

- PVM - Parallel Virtual Machine.
  http://www.epm.ornl.gov/pvm/pvm_home.html

- LAM/MPI (Local Area Multicomputer / Message Passing Interface).
  http://www.mpi.nd.edu/lam

- BERT77 - FORTRAN conversion tool.
  http://www.plogic.com/bert.html

- Beowulf software from Beowulf Project Page.
  http://beowulf.gsfc.nasa.gov/software/software.html

- Jacek's Beowulf-utils (disponíveis por ftp anónimo).
  ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils

- bWatch - cluster monitoring tool.
  http://www.sci.usq.edu.au/staff/jacek/bWatch

Tecnologias de rede

- Boden et. al. Myrinet - A Gigabit-per-Second Local-Area Network. IEEE Micro, February 1995.
  http://www.myri.com/

- Hardware e software para a Tecnologia SCI.
  http://www.dolphinics.com

- Tecnologias de rede de alta velocidade com suporte para Linux .
  http://cesdis.gsfc.nasa.gov/linux/drivers/index.html

Máquinas Beowulf

- Avalon, consiste em 140 processadores Alpha, 36 GB de RAM, e é provavelmente a máquina Beowulf mais rápida atingindo os 47.7 Gflops e situando-se em 114º lugar na lista das 500 máquinas de maior desempenho.
  http://swift.lanl.gov/avalon/

- Megalon-Massively PArallel CompuTer Resource (MPACTR).Consiste em 14 nós, cada um com quatro processadores Pentium Pro 200 MHz e 14 GB de RAM.
  http://megalon.ca.sandia.gov/description.html

- theHIVE - Highly-parallel Integrated Virtual Environment. É outro supercomputador Beowulf.
  http://newton.gsfc.nasa.gov/thehive/

- Topcat é uma máquina muito mais pequena e consiste em 16 CPUs e 1.2 GB de RAM.
  http://www.sci.usq.edu.au/staff/jacek/topcat

- MAGI cluster – Site com bastantes links.

http://noel.feld.cvut.cz/magi/

- ProjectoVitara – É um Beowulf português implementado no ISCTE. Consiste em 9 nós interligados por uma rede Fast Ethernet.
  http://vitara.adetti.iscte.pt

ANEXO - COMO IMPLEMENTAR UM BEOWULF

INSTALAÇÃO E CONFIGURAÇÃO DO HARDWARE

- Para configurar a *BIOS* de cada nó, ligar o teclado e a placa de vídeo a esse nó e configurá-lo individualmente. Apenas é necessário um teclado e uma placa de vídeo para todos os nós, mas pode-se usar um conjunto para cada nó;
- As configurações que normalmente têm de ser feitas são os discos rígidos *IDE* e fazer *"halt"* nos erros de teclado e placa de vídeo. Se os nós não tiverem teclado ou placa de vídeo eles não devem fazer o *"halt"* quando estes não forem detectados pela *BIOS*;
- Ligar todos os nós à energia e cabos *UTP* entre os nós e o comutador.

INSTALAÇÃO DO SISTEMA OPERATIVO

- Instalar o Linux no nó servidor (como referência fica o *Red Hat Linux 5.2*);
- Assegurar que se deixa espaço suficiente na partição de *root* para os sistemas de ficheiros *NFS-root* de todos os nós clientes, ou seja, cerca de 500 KB por cliente;
- /var/log do servidor vai requerer espaço de disco adicional para armazenar o seu *log* e também os dos clientes, porque todos os nós de clientes vão escrever os seus *logs* no *syslogd* do servidor;
- Na configuração clientes sem disco rígido /var, /lib, /bin, /sbin e /etc não devem ter *mounts* separados, ou seja, devem ser instalados na mesma partição;
- Certificar que todos os dispositivos de rede são suportados;
- Certificar que o *kernel* do Linux instalado suporta o *RARP* (CONFIG_INET_RARP), porque este é necessário para responder aos pedidos *RARP*.

INSTALAÇÃO DO SERVIDOR

O servidor vai servir sistemas de ficheiros por *NFS* aos nós clientes, vai ser usado para compilar o código fonte e iniciar trabalhos paralelos, e vai ser o ponto de acesso a partir do exterior.

Tamanhos das partições

- É muito importante escolher correctamente os tamanhos das partições de acordo com as necessidades do utilizador porque pode ser difícil alterá-los depois, quando o *cluster* já estiver a correr código de produção.

Como referência apresentam-se as partições para um disco de 4GB com o *Red Hat Linux 5.2*, e um *cluster* de 16 nós numa configuração de clientes sem disco rígido, sem incluir a partição /home (onde serão armazenados os ficheiros do utilizador):

- / : 500MB. Esta partição vai conter as directorias /bin, /boot, /dev, /etc, /lib, /root, /sbin, /var e /tftpboot e possivelmente a /tmp, e os seus conteúdos. Numa configuração de clientes sem disco rígido é muito importante que /tftpboot fique na mesma partição que / , porque senão não se poderá criar algumas das ligações necessárias para a configuração de *NFS-root* funcionar.

- /usr: 1,5GB. Maior parte dos *rmps* adicionais irão instalar em /usr e não em /usr/local. Se se estiver a pensar instalar pacotes grandes deve-se criar esta partição ainda maior.

- /usr/local: de 500MB a 2 GB. O tamanho exacto depende do *software* adicional que se pretende instalar.

- Área *de Swap*: não mais do que duas vezes o tamanho da *RAM* (sem ultrapassar 126Mbytes).

Configuração da rede

- Uma das placas de rede deve ter um endereço IP válido e a outra um IP privado (exemplo: 10.0.0.1) visível apenas pelos nós do *cluster*;
- A interface de rede pode ser configurada ou usando ferramentas *GUI* ou criando e editando ficheiros /etc/system/network-scripts/ifcfg-eth*;

Configuração de DNS

Ter um domínio e servidor DNS simplifica a administração do Beowulf, mas não é obrigatório.

- O servidor do *cluster* (no1) vai ser o servidor de DNS. Os ficheiros de configuração de DNS podem ser encontrados em ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils. Nestes ficheiros usa-se o intervalo 10.0.0.0/8 para os endereços IP privados, com uma máscara de 255.0.0.0;
- Há alguns ficheiros de configuração que terão de ser modificados para o DNS funcionar;
- Depois de instalados os ficheiros de configuração ter-se-á de reiniciar o *daemon named* executando /etc/rc.d/init.d/named restart.

/etc/hosts

- Se se decidiu não usar servidor DNS tem que se dar entrada de todos os nós (nomes e *aliases*) e dos seus endereços IP correspondentes no ficheiro /etc/hosts;
- Estando-se a usar a configuração de clientes sem disco rígido as *scripts sdct* e *adcn* vão criar ligações com este ficheiro, logo vai ser usado por todos os nós.

/etc/resolv.conf

- Se se tem um servidor DNS no servidor do Beowulf então o ficheiro resolv.conf deve apontar primeiro para o servidor de nomes local;
- Se não se tiver um servidor DNS ter-se-á de apontar para outros servidores de nomes.

/etc/hosts.equiv

- Para permitir *shells* remotas (rsh) de qualquer nó para outro no *cluster*, para todos os utilizadores, deve-se diminuir a segurança e listar todos os *hosts* em /etc/hosts.equiv.

Sincronização do relógio

A melhor solução para ter os relógios do *cluster* sincronizados é usar xntp e sincronizar a uma fonte externa.

Como usar xntp:

1. acertar todos os relógios do sistema pela hora corrente;
2. escrever a hora no CMOS RTC (real Time Clock) usando o comando clock –w;
3. montar o cdrom em cada sistema;
4. ir para /mnt/cdrom/RedHat/RPMS;
5. como *root*, fazer rpm –i xntp3-5.93-2.i386.rpm;
6. editar o ficheiro /etc/ntp.conf

   Em todos os sistemas pôr as linhas (como mostrado):

   ```
   #multicastclient                    # listen on default 224.0.1.1
   #broadcastdelay  0.008
   ```

   Em todos os sistemas excepto no *host* editar as linhas:

   ```
   server  HOSTNODE # local clock
   #fudge  127.127.1.0 stratum 0
   ```

(onde HOSTNODE é o nome do *host*);

7. Fechar o ficheiro /etc/ntp.conf em cada nó;
8. Iniciar o xntpd e todos os sistemas, "/sbin/xntpd".
   Pode-se iniciar isto cada vez que se faz o *boot* acrescentando-o ao ficheiro /etc/rc.d/rc.local. Será melhor, por prevenção, fazer o sync dos relógios uma vez por dia. Isto pode ser feito entrando como *root* no /etc/cron.daily e criando um ficheiro chamado "sync_clocks" que contenha o seguinte:

   # Assumindo que o ntp está a correr, sync o CMOS RTC ao relógio do SO
   /sbin/clock –w

INSTALAÇÃO DOS CLIENTES

Existem três métodos principais de instalação de nós clientes:

- o primeiro consiste em clonar os nós usando o comando dd;
- o segundo é instalar o sistema operativo em cada cliente separadamente e depois correr uma *script* de configuração no servidor que executa o resto da instalação;
- o terceiro é usar clientes sem disco rígido e neste caso toda a instalação e configuração é feita no servidor.

Clonar os clientes

A ideia básica do conceito de clonagem é fazer uma cópia exacta da partição de uma *drive* noutra *drive*.
A forma mais fácil de fazer isto é instalar um cliente, configurar esse cliente e fazer uma cópia exacta do disco, que será usada nos outros clientes fazendo apenas uma pequenas alterações: endereço *IP* e *hostname*.
Se os clientes tiverem disco próprio com o sistema operativo, esta é a forma mais fácil.

Existem duas formas para copiar de uma *drive* para outra:

- usar o comando dd em toda a *drive*. Isto tratará tanto das partições como dos *boot sectors* e dos ficheiros;
- se não se tiver o mesmo tipo de *drives* nos nós, criar ficheiros *tar* da partição de *root* e *usr* do sistema configurado que se instalar:

```
# mkdir /usr/images
# tar cvflz /usr/images/root.tar.gz /
# tar dvflz /usr/images/usr.tar.gz /usr
exclude /usr/images
```

Instalar o sistema operativo em cada cliente, separadamente

Se se estiver a fazer a instalação a partir do CD-ROM e apenas se tiver um para todo o sistema, tem de se mover o CD-ROM de cliente para cliente depois de cada instalação, ou então fazer uma instalação *NFS*.
Se só se tiver uma *drive* de disquetes também vai ser preciso move-la.

Usar clientes sem disco rígido

Por este método, toda a configuração dos clientes é feita no servidor uma vez que, como os clientes não têm disco rígido, todos os ficheiros dos clientes estão armazenados no nó servidor.

- Compilar uma disquete de arranque *NFS-root* para os clientes. A forma mais fácil é fazer um *Kernel* monolítico para os clientes, certificando-nos de que os sistemas de ficheiros *NFS* e *NFS-root* são compilados nele. As seguintes opções devem ser compiladas pois são o suporte para o *NFS-root*: CONFIG_ROOT_NFS, CONFIG_RNFS_BOOTP, CONFIG_RNFS_RARP. Não esquecer de compilar o suporte para a placa de rede:

```
make menuconfig
```

- Depois de configurar todas as opções no *Kernel* este tem que ser recompilado. Usar os seguintes comandos:

```
make dep && make clean && make zImage
```

- Alterar o dispositivo de *root* do *kernel* para *NFS-root*, e acrescentá-lo na disquete:

```
mknod /dev/nfsroot b 0 255
cd /usr/src/linux/arch/i386/boot
rdev zImage /dev/nfsroot
```

- Copiar este *kernel* para uma disquete. Fazer o *boot* a um cliente com esta disquete vai fazer o *broadcast* dos pacotes *RARP* e BOOTP à procura do servidor *NFS-root*.

- O passo seguinte, depois de criar a disquete de arranque, é criar directorias *root* para os clientes através de uma directoria modelo. Este modelo vai ser usado para o sistema de ficheiros dos clientes. Basta fazer o *"cut and paste"* da *script sdct* para um ficheiro e corrê-lo. A *script* vai criar todas as directorias e copiar todos os ficheiros necessários;

Esta *script* não cria a directoria *root* para nenhum cliente, mas sim um modelo que vai ser usado por outra *script* para criar essas directorias. Pode ser necessário fazer pequenas alterações ao modelo depois de ser criado para se ajustar às nossas necessidades, ou mesmo alterar a própria *script* para reproduzir as alterações mais facilmente.

- Agora que está criada a directoria *NFS-root* modelo, vai-se criar um sistema de ficheiros *NFS-root* para cada cliente. A forma mais fácil de o fazer é correndo a *script adcn* para cada cliente:

adcn –i 10.0.0.2 –c node2 –d my.beowulf.domain –l -D eth1

onde eth1 é a interface conectada ao *cluster*;
- Colocar a disquete do *NFS-root kernel* na *drive* do cliente e fazer o *reboot*.
(Estas *scripts* referidas podem ser encontradas em: ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less)

Se um cliente sem disco não obtiver uma resposta *RARP* do servidor

Se se fizer o *boot* ao cliente e ele estagnar coma mensagem: *"Sending BOOTP and RARP requests..."* deve-se verificar o seguinte:

- os cabos de rede e a configuração do comutador;
- que o *RARP* é suportado pelo *kernel* do servidor;
- que o servidor tem uma entrada *RARP* para o cliente problemático: rarp -a
- que o endereço de hardware do cliente está correcto;
- correr o 'tcpdump –i eth1 rarp' no servidor e fazer o *boot* no cliente (sendo eth1 a interface ligada ao *cluster*).

SEGURANÇA

Não é necessária muita preocupação em relação à segurança dentro do *cluster* porque nenhum dos nós clientes é directamente ligado ao exterior. Para aceder aos nós clientes tem de se passar pelo nó servidor. O servidor deve confiar no clientes mas não no mundo exterior. É preciso proteger o *cluster* em relação ao exterior.

No servidor

- *TCP wrappers*

É a primeira linha de defesa e a forma mais simples de limitar o acesso ao sistema. Vem no *Red Hat Linux* e é bastante simples de configurar. Há três ficheiros de configuração:

/etc/hosts.allow que verifica os *hosts* que têm permissão para se ligarem;
/etc/hosts.deny que é lido se o cliente não for encontrado em /etc/hosts.allow e verifica os *hosts* que têm proibição de ligação;
/etc/inetd.conf que não será necessário modificar para configurar o *tcpd*.

A *'man page'* hosts_access(5) mostra-se uma ajuda bastante útil para a sintaxe do /etc/hosts.allow e do /etc/hosts.deny

- Parar *daemons* não usados

Desactivar serviços desnecessários é uma boa forma de melhorar a segurança no servidor. A maior parte dos *daemons* são iniciados pelo super servidor *inetd* e podem ser desligados pondo em comentário as respectivas linhas no inetd.conf.
Depois de modificar o ficheiro de configuração tem de se reiniciar o *daemon inetd*. A melhor forma de o fazer é enviar ao *daemon* um sinal de desligar que vai forçá-lo a reler o seu ficheiro de configuração.

- Desactivar serviços iniciados por *scripts rc*

Serviços como o *Web* (httpd) e o *Samba* (smbd) iniciam com *scripts rc*. Normalmente são desactivados apagando a ligação correspondente na directoria /etc/rc.d/rc.3d. Estas ligações apontam para *scripts* de iniciação em /etc/rc.d/init.d.

- *ipfwadm*

ipfwadm é um programa que permite bloquear pacotes específicos de endereços *IP* para portas específicas.

Nos clientes

- .rhosts versus hosts.equiv

É necessário permitir aos utilizadores o *login* e a execução de *shells* remotas entre os nós sem introduzir a senha. A maior parte do *software* e dos utilitários para Beowulf assumem que se possam executar *shells* remotas (rsh) para pelo menos todos os nós clientes sem precisarem de senha.
Existem duas formas de eliminar as senhas dentro do *cluster*:
Ou acrescentando uma entrada no ficheiro /etc/hosts.equiv;
Ou então acrescentar um .rhosts na directoria de *'home'* de cada utilizador.
A preferível é a primeira porque a informação de hosts.equiv aplica-se a todo o nó, enquanto .rhosts é por utilizador.

- Acesso do *root* por *rlogin*

Para permitir ao *root* fazer *rlogin* a qualquer nó no *cluster*, acrescenta-se um ficheiro .rhosts na directoria de *root* em cada nó. Este ficheiro lista todos os nós no *cluster*, é importante que só seja de leitura/escrita para o dono. Nunca fazer isto no servidor. Para além disto, trocar a posição entre si das duas primeiras linhas do ficheiro /etc/pam.d/rlogin.

- Acesso do *root* por *telnet*

Uma forma de permitir o *telnet* a qualquer nó no *cluster*, excepto ao servidor é acrescentar ao ficheiro /etc/security as seguintes linhas:

ttyp0
ttyp1
ttyp2
ttyp3
ttyp4

- Acesso do *root* por *ftp*

Em qualquer sistema que seja necessário que o *root* aceda por *ftp* o ficheiro /etc/ftpusers tem de ter a entrada para o *root* em comentário.