UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

TRABALHO DE LICENCIATURA

TEMA:

...DELO DE ALTA DISPONIBILIDADE EM "CLUSTER" USA... FERRAMENTAS OPEN SOURCE

ESTUDANTE: DOMINGOS FERNANDO CHITLHANGO

MAPUTO, OUTUBRO DE 2006

UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

Trabalho de Licenciatura

Tema:

Modelo de Alta Disponibilidade em "Cluster" Usando Ferramentas Open Source

Estudante: Domingos Fernando Chitlhango

Maputo, Outubro de 2006

UNIVERSIDADE EDUARDO MONDLANE
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

Trabalho de Licenciatura

Tema:

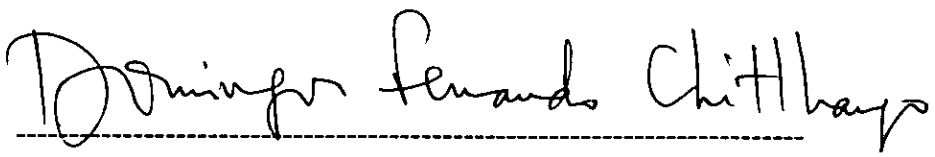
Modelo de Alta Disponibilidade em "Cluster" Usando Ferramentas Open Source

Supervisor: dr Afonso Tzandzana
Co-Supervisor: Eng. Rogério Lam
Estudante: Domingos Fernando Chitlhango

Maputo, Outubro de 2006

Declaração Sob Palavra de Honra

Este trabalho foi efectuado, somente, com base nos recursos que, ao longo do mesmo se faz referência.



(Domingos Fernando Chitlhango)

Dedicatória

Dedico este trabalho a minha mulher Florentina, ao meu filho Lino, ao meu irmão Salomão e ao resto da minha família.

Agradecimentos

Este pequeno espaço está reservado para tecer todos os agradecimentos a todos que directa ou indirectamente contribuíram para a realização deste trabalho.

Desejo agradecer em primeiro lugar aos meus supervisores, O dr. Afonso Tzandzana e o Eng. Rogério Lam pela sábia orientação, preciosas observações e sugestões e, sobretudo, pela paciência que tiveram para o sucesso deste trabalho e em segundo, ao meu irmão que também prestou um grande apoio.

Agradeço aos meus colegas de trabalho e amigos, Edgar Gemo, Florencia Marrão Suamade, Adriano Herberth, Ilídio Chunguane, Nazário Binana e Suzel Emílio. Amigos como vocês não se fazem num dia.

Esta lista alarga-se á minha família, os meus pais, a minha cunhada, aos meus sobrinhos, e ao Pessoal da Biblioteca do DMI, pela dedicação, amor e carinho que tiveram para a concretização deste trabalho.

A TODOS, MUITO OBRIGADO

Terminologia

***Backup*:** É um processo usado para a recuperação de informação, gravando-as em fitas, ou em disquetes com o propósito de a ter sempre disponível caso ocorra algum problema que ocasiona a sua perda total ou parcial, em outro tipo de armazenamento, normalmente arquivo de trabalho.

***Boot*:** Procedimentos iniciais que o computador tem que cumprir antes de estar apto a responder a comandos. Inclui diagnósticos, testes nele próprio, conferência de tabelas e preparação para carga das rotinas indispensáveis para operação dos dispositivos do computador.

***Byte*:** É um termo genérico que indica um determinado número de dígitos binários consecutivos, operando com uma unidade de medida. Também é identificado como um conjunto de 8 bits operando como unidade mínima de medida.

***Cluster*:** É um sistema que compreende dois ou mais computadores, denominados nó, no qual trabalham em conjunto para executar aplicações ou realizar outras tarefas, de tal forma que os utilizadores tenham a impressão que somente um único sistema responde para eles, criando assim ilusão de único recurso (computador virtual).

Confiabilidade: É a capacidade de um item desempenhar uma função requerida sob determinadas condições para um dado período de tempo. Também pode ser referido como a probabilidade de que um sistema ou componente funcione de acordo com especificações, durante um dado intervalo de tempo, e em determinadas condições de operar, é identificado pelo MTBF (tempo médio entre falhas).

Consola: É a unidade do computador usada para controlar a máquina por métodos manuais, como corrigir erros, determinar o estado dos circuitos, registos e contadores.

Dado: Esta designação é utilizada para descrever qualquer conjunto de factos ou operandos constituídos por algarismos, caracteres alfabéticos ou símbolos especiais que descrevem: procedimentos, condições, objectos, ideia, situações ou outros estados.

Default: É uma acção ou um valor que o computador assume automaticamente, a menos que uma instrução modifique esse estado.

Disponibilidade: É o estado de um sistema quando pronto para processamento de dados no momento em que é solicitado.

Downtime: É o período de tempo durante o qual o computador está em estado de mau funcionamento, ou seja, não está a operar correctamente devido a um defeito mecânico ou electrónico, em oposição ao termo permissível (available time).

Drivers: É uma rotina de um programa operacional que dirige as unidades periféricas individuais no computador. É necessário que uma rotina deste tipo trate os detalhes íntimos da estrutura de cada unidade e do seu comportamento em tempo real.

Fallback: Após que uma falha de um servidor for detectada, além do *failover* é necessário que se faça a manutenção do mesmo. Na recuperação de uma falha, este servidor será recolocado em serviço e, tem-se a opção de realizar o processo inverso do *failover*, que se chama *fallback*. O *fallback* é o processo de restauro de um determinado serviço, ou de uma outra máquina para o seu estado de origem.

Failover: É o processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha, podendo ser automático ou manual, sendo automático aquele que normalmente se espera de uma solução de alta disponibilidade.

Falha: É a incapacidade de terminar uma operação devido a uma razão qualquer. Uma falha acontece no universo físico, ou seja, no nível mais baixo do hardware. São exemplos de falha uma flutuação da fonte de alimentação, ou interferência electromagnéticas. Estes são dois eventos indesejados, que acontecem no universo físico e afectam o funcionamento de um computador ou de partes dele.

***Hardware*:** Refere-se a parte física de um computador incluindo os componentes eléctricos e electrónicos (por exemplo, dispositivos e circuitos), os componentes electromecânicos (por exemplo, uma unidade de disco) e os componentes mecânicos.

***Inodes*:** São objectos usados em Unix para o registo de informação sobre os ficheiros. No geral, contêm duas partes, sendo que a primeira armazena informação sobre ficheiros, incluindo o dono do ficheiro, suas permissões e seus tamanhos e, a segunda parte do *inode*, contêm apontadores aos blocos de dados associados com os ficheiros dentro do sistema de ficheiros.

***Interface*:** Neologismo utilizado para definir a ligação física, lógica ou funcional entre duas unidades ou sistemas distintos.

Journaling: É um sistema de ficheiros que regista as mudanças a um *journal* dentro do sistema de ficheiros, antes de as escrever ao sistema de ficheiro principal.

Keep Alive: Este parâmetro controla a frequência que o TCP verifica se uma conexão está ainda activa dentro do sistema.

Kernel: É entendido como o núcleo deste, representa a camada mais baixa de interface com o *Hardware*, sendo responsável por administrar os recursos do sistema operativo como um todo. É nele que estão definidas funções para operação com dispositivos periféricos (*mouse*, discos, impressoras, interface serial, interface paralela), administração

de memória entre outros. Resumidamente, o *kernel* é um conjunto de programas que fornece aos utilitários um interface para utilizar os recursos do sistema.

Largura de Banda: É a taxa à qual um sistema de comunicações pode transmitir dados, em outras palavras, é o intervalo de frequência a que um sistema electrónico pode transmitir dados. É medida em bits por segundo (bps), por exemplo: 256 Mbps.

Linux: é um sistema operativo para PCs e outros equipamentos, distribuído gratuitamente e compatível com o *UNIX*.

Log: É um arquivo contendo todos os dados relativos a execução de um programa na máquina. Ele contêm um conjunto de informações tais como, tempo de execução, números de vezes que o ficheiro ou programa foi executado, acessos e decisões tomadas na ocorrência das mesmas (arquivo de eventos).

Online: É uma técnica de operação em que a transmissão de uma informação é enviada ao mesmo instante em que é digitada no transmissor.

Open Source: Refere-se a programação de código aberto, permitindo os seus utilizadores alterações de programação de modo que personalizem as aplicações de acordo com as suas necessidades.

Perfomance: Junto com a flexibilidade, é um dos maiores factores dos quais depende a produtividade total dum sistema. O desempenho é normalmente determinado pela combinação de três factores: confiabilidade, tempo de resposta e viabilidade.

Placa Mãe – É Placa principal de circuitos integrados de um computador, contendo CPU e a memória. Alguns computadores já incluem mais dispositivos como portas de impressoras, rede, e outros.

RAID: *Redundante Arrays of Inexpensive* - são vários discos combinados para aumentar a perfomance, e a um nível mais complexo, o RAID pode ser usado também para a confiabilidade do equipamento por meio de espelhamento ou paridade.

Redundância: É uma precaução em ter informação extra ou hardware adicional para aumentar a fiabilidade

Servidor: Computador que fornece serviços a outro computador (o cliente), por exemplo, um servidor de base de dados, de comunicação, de impressão, etc.

***Single point of failure* (SPOF):** Representa uma componente do sistema, no qual se o seu funcionamento correcto falhar permitirá a falha do sistema num todo.

Sistema: Um conjunto de factos, equipamentos, acontecimentos ou procedimentos relacionados ou interactuantes que constituem uma unidade.

Sistema Operativo: é um programa que controla um computador e especifica o modo como o utilizador deve introduzir e executar os seus próprios programas.

***Software*:** É a totalidade de programas e rotinas usados para aumentar a performance de um computador, com os compiladores, assembladores, narradores, rotinas e sub-rotinas.

Tolerância: É o valor ou grandeza da variação permissível em torno de um padrão preestabelecido, normalmente expressa em percentagem.

Transações: É um conjunto de dados que identificam as alterações e as actualizações que foram feitas em registos de um arquivo.

Uptime: Refere-se ao tempo em que o computador ou outro equipamento está a operar ou em estado de poder operar quando for solicitado (tempo produtivo, tempo de operação).

Abreviaturas:

Siglas	Descrição
ACL	Lista de Controle de Acesso
CPU	Unidade de Processador Central
DNS	*Domain Name Service*
DRDB	*Distributed Replicated Block Device*
EFS	*Extended File System*
Eth0	*Ethernet 0*
Eth1	*Ethernet 1*
Fsck	*File System Check*
FTP	*File Transfer Protocol*
GB	*Gigabyte*
GPL	*General Public License*
HA	*High Availability*
HP	*Hewllet Packard*
IP	*Internet protocol*
JFS	*Journal File system*
KB	*Kilobytes*
LAN	*Local Area Network*
MB	*MegaByte*
Mon	*Service Monitoring Daemon*
MTBF	Tempo Médio Entre Falhas
MTTR	Tempo Médio Para Reparo
NFS	Sistema de Arquivos em Rede

SAD	*Sistema de Alta Disponibilidade*
SCSI	*Small Computer System Interface*
SMB	*Server Message Block*
SO	Sistema Operativo
SPOF	*Single Point Of Failure*
TB	*TeraByte*
UPS	*Uninterruptible Power Supply*
VGA	*Vedeo Graphics Array*

Resumo

O sistema de alta disponibilidade (SAD) é uma tecnologia que permite a eliminação dos pontos de falhas para as aplicações e garantir a continuidade dos seus negócios. Em Moçambique já existem empresas que usam esta tecnologia, estão distribuídas pelos sectores de Comunicações, Finanças, Banca, Transporte e Serviços.

O Factor que leva diversas instituições a aderirem aos serviços de alta disponibilidade são vários, sendo os principais a competitividade no mercado e a necessidade de aumento do rendimento.

O presente trabalho debruça-se sobre todos os aspectos de alta disponibilidade e as diversas técnicas que podem ser encontradas nestes sistemas. São descritos os sistemas de ficheiros que podem ser encontrados em sistemas UNIX, cujo maior foco é o *journalling*. É feito o desenho de um sistema desta natureza tendo em conta todos os pontos de falhas que possam ocorrer, sem tomar em conta a componente corrente eléctrica e, por fim é implementado o sistema de alta dispobilidade.

Dentre as principais razões que levam à aderência da tecnologia de alta disponibilidade em ambiente *Open Source*, destaca-se o facto da infra-estrutura tecnológica ser de baixo custo, os benefícios resultantes dos serviços de alta disponibilidade constituírem motivação para a sua adopção e, o facto da maioria das empresas, de pequeno e grande porte, considerarem que os seus clientes são tão exigentes e não podem esperar tanto tempo para a reposição dos seus serviços quando estes não estiverem disponíveis quando necessários. É ilustrado até que ponto as pequenas empresas também podem implementar estes sistemas dentro dos seus negócios a baixo custo.

Prefácio

O trabalho está dividido em 8 capítulos, que a seguir são descritos:

Capítulo 1, 2 e 3: Referem-se à introdução, objectivos do trabalho e metodologia para o trabalho em curso.

Capítulo 4: Debruça-se sobre aspectos de alta disponibilidade de uma forma geral, dando os tipos de Cluster e os conceitos básicos que compõem o sistema de alta disponibilidade.

Capitulos 5 e 6: Faz-se a análise das diversas formas de contornar falhas de sistemas computacionais e uma descrição comparativa dos diversos tipos de sistemas de ficheiros para sistemas de alta disponibilidade.

Capítulo 7: É feita uma descrição passo a passo do processo de implementação do ambiente de alta disponibilidade proposto, desde a preparação das máquinas, instalação dos aplicativos e sua configuração final.

Capítulo 8: Este é reservado as conclusões referentes ao trabalho em curso e as respectivas recomendações.

Convenções usadas

A seguir descreve-se a lista de convenções usadas neste trabalho:

```
Alert for group $opt_g, service $opt_s
```

Esta foi usada para mostrar o conteúdo de ficheiros de configuração no trabalho e o resultado da execução de comandos na aplicação.

Auth <number> <number> <authmethod> [<authkey>]

Esta representa a sintaxe de execução de comandos.

Italic

É usado para descrever os nomes de ficheiros, directórios, comandos e opções de comandos. Adicionalmente é usado no neologismo dos termos de casos em inglês.

Resultado de execução de comandos:

```
<item>          <Designação>    <Descrição detalhada e outra
                                informação>
```

Índice

1. Introdução

Nas últimas décadas, com a evolução das tecnologias de informação e comunicações e com a crescente demanda de serviços informáticos virados ao cliente, várias mudanças têm-se verificado pelo uso da alta disponibilidade (*High Availability* ou HA) em várias instituições.

Segundo [Vigliazzi 2002], a alta disponibilidade está ligada directamente à crescente dependência dos sistemas computacionais. Com o avanço das actividades que dependem directamente de recursos computacionais, é cada vez maior o transtorno causado pela falha de sistemas. Desde os sistemas bancários até os de supermercados, os recursos computacionais têm um papel crítico. Mas não apenas em tais tipos de sistemas, principalmente em empresas cuja maior funcionalidade é a oferta de algum serviço computacional, para os serviços considerados críticos, que dele dependem as funções vitais da instituição.

Com a expansão do mercado de comunicações, as ideias e problemas tornam-se comuns. Sistemas oscilantes, suporte mal qualificado e demora na solução de problemas são comuns. A eficiência pode ser um grande diferencial. O sistema ao estar sempre disponível quando o cliente solicitar, certamente será um diferencial. E o contrário muito mais. Pesquisas mostram que um cliente bem atendido repassa a boa impressão a 2 ou 3 pessoas, mas um mal atendido repassa a 8 ou 10 Pessoas [Garcia, 2003]. Então, o cliente acredita que o sistema sempre disponível é um requisito básico, não é mais do que a sua obrigação.

Mas pelo contrário, sistemas computacionais falham, e um bom projecto prevê estas falhas e uma forma de contorná-las. Será que a alta disponibilidade pode solucionar ao mínimo as falhas computacionais?

Presentemente, quer seja pelo ambiente em que as empresas tem de operar, ou pela força das influências entre os sectores, todas as organizações tem sido afectadas pela nova realidade do mercado computacional e a alta disponibilidade não podia ser excepção. Essa situação tem exigido das organizações grande esforço para a assimilação

e utilização das tecnologias referentes a alta disponibilidade, tanto do ponto de vista da sua operacionalização, como em termos competitivos.

2. Objectivos

2.1. Objectivo Geral

- Construir um modelo de alta disponibilidade baseado no Sistema Operativo *Linux* e Ferramentas *Open Source*.

2.2. Objectivos Específicos

- Estudar um sistema de alta disponibilidade;
- Desenhar um sistema de alta disponibilidade;
- Implementar um sistema de alta disponibilidade;

3. Metodologia

- Para realizar o estudo do sistema de alta disponibilidade (SAD), foi feita a consulta bibliográfica a manuais, páginas da *Internet* e consultas a especialistas da área, como forma de determinar os principais pontos de análise do caso.

- Para implementar o SAD proposto, foram usados 2 computadores, com processador *Intel* 865GV *chipset*, 1024 MB de RAM, 2.4 GHz do processador, 73 GB de disco duro, seis portas de USB 2.0, uma porta serial padrão e uma opcional, uma porta paralela, 2 PS/2, quatro placas de rede RJ-45, uma VGA, *audio in/out*, *headphone and microphone*, e um *switch*, os quais serão configuradas usando aplicativos de alta disponibilidade em *Open Source*.

- Para o desenho do SAD, foi elaborado uma topologia de rede em estrela, para a comunicação entre os dois nós do *Cluster* e o resto dos clientes (ver fig. 5 pág 25). Os componentes da rede em *cluster* resumem se em:

  - Rede
    - Foi utilizado o endereçamento com o protocolo TCP/IP, o qual permitiu a definição de endereços IP, públicos e privados. Um endereço IP público, representa a comunicação entre a rede de utilizadores (rede pública) e aplicações em *cluster*, enquanto que um endereçamento IP privado representa a comunicação com a rede de *heartbeat* do *cluster,* e está completamente isolada da rede de utilizadores.
    - Parte Física, onde se pode encontrar os cabos de rede normais e cruzados, e um *switch*.
  - Serviços
    - Foi estabelecido uma consola que permitirá a administração da aplicação do *cluster*;
    - Foi utilizada uma página Web em *xml* para teste e será incorporada no *apache* com a versão *2.0.*

4. Aspectos de Alta Disponibilidade

Segundo Lewis (1999), a alta disponibilidade é definida como sendo a probabilidade de um sistema estar disponível (em perfeito funcionamento) num dado momento. Ela pode ser classificada em: básica, alta e contínua.

Disponibilidade Básica – A disponibilidade básica é aquela encontrada em computadores comuns, sem nenhum mecanismo especial, em *software* ou *hardware*, que visa de alguma forma prever eventuais falhas. Computadores nesta classe apresentam uma disponibilidade de 99%.

Disponibilidade Alta – Adicionando-se mecanismos especializados de detecção, recuperação e prevenção de falhas, pode-se aumentar a disponibilidade do sistema, por forma a que este venha a se enquadrar na classe de alta disponibilidade. Nesta classe os Servidores tipicamente apresentam disponibilidade na faixa de 99,99% a 99,999%, podendo ficar indisponíveis por um período de pouco mais de 5 minutos até uma hora em um ano de operação (ver tabela 1 na página 8).

Disponibilidade Contínua – Com a adição de noves se obtém uma disponibilidade cada vez mais próxima de 100%, diminuindo o tempo de inoperância do sistema de forma que este venha a ser desprezível ou mesmo inexistente. Chega-se então na Disponibilidade Contínua, o que significa que todas as interrupções planeadas e não planeadas são mascaradas, e o sistema está sempre disponível.

Em projectos de sistemas, a alta disponibilidade se encaixa em ambiente "*Cluster*" e em sub-área de tolerância de falhas, uma vez que objectiva a continuidade do sistema mesmo com ocorrência de falhas.

4.1 Matemática da Confiabilidade e Disponibilidade

Segundo Jones (2000), um ambiente que está sempre disponível é 100% de tempo disponível, independentemente de como é medido este tempo. Se um sistema está disponível 99% tem um significado imediato, significa que 1% de tempo não estará disponível para suportar as aplicações e, ainda não se sabe o seguinte:

- Com que frequência o sistema pode estar indisponível?
- Quanto tempo a rede pode estar indisponível?
- É relevante a rede estar a funcionar continuamente?
- O período de tempo da medida?

Vejamos, se o sistema for calendarizado para estar disponível continuamente (24 horas X 7 dias X 365 dias) e medindo sobre um período de tempo anual, a disponibilidade de 99% traduz em 87 horas e 4 minutos do *downtime*. Se o sistema é requerido apenas 8 horas de tempo por dia, 5 dias por semana, a disponibilidade de 99% traduz em 2.200 horas e em 488 minutos do *downtime*. No entanto, se as 20 horas estarem registadas somente durante as horas de funcionamento, o *downtime* iguala a 2.5 dias em que o sistema fica indisponível durante os dias de trabalho e de *downtime* ilimitado fora das horas de trabalho.

O sumário de *downtimes* permitidos anualmente para vários níveis de disponibilidades, segundo Jones (2000), é apresentado na tabela a seguir:

Disponibilidade	24horasX7diasX365 dias	Turnos
90%	3.65 Dias	25 Dias de trabalho
99%	87 Horas, 45 minutos	20.8 Horas (2.5 dias de trabalho)
99.9%	8 Horas, 45 minutos	2 Horas, 5 minutos
99.99%	52.6 Minutos	12.5 Minutos
99.999%	5 Minutos, 15 segundos	75 Segundos
99.9999%	31.6 Segundos	7.5 Segundos

***Tabela 1**: Disponibilidade requerida e downtime respectivo*

Na determinação de níveis aceitáveis de disponibilidade, deve ser considerado a duração máxima de um único incidente do *downtime*. Os 99% de disponibilidade são aceitáveis em média, se todos *downtimes* tiverem ocorrido em simples incidentes e como resultado um impacto negativo em algumas actividades organizacionais e consequente perda de credibilidade perante os seus clientes, sendo obrigado a mudar os requisitos perante o sistema. Obviamente que é necessário saber não somente os requisitos médios, mas também os limites até que a organização atinja o tal impacto.

***Mean Time Between to Failure* (MTBF) e *Mean Time to Repair* (MTTR)**

A disponibilidade pode ser caracterizada por: Tempo médio entre falhas *(*MTBF*)"* e tempo médio para reparo *(*MTTR*)"*.

O MTBF define o tempo, em média, em que um sistema é esperado a funcionar e livre de falhas, enquanto que o MTTR define o tempo em média, requerido para obter o sistema a funcionar outra vez depois de ter ocorrido uma falha. Deste modo a disponibilidade é medida segundo a equação:

Disponibilidade = MTBF*100%/(MTBF + MTTR)

(Jones, 2001)

EXEMPLO:

Supõe-se que um carro foi comprado à meia-noite do dia 1 de Janeiro do início do ano, no dia 15 de Fevereiro corrente a viatura teve um pneu furado que levou uma hora para a sua manutenção (tarefa não prevista), no dia 1 de Abril, faz-se a manutenção e levou duas horas para a mudança de óleos (tarefa calendarizada), no dia 5 de Julho, verificou-se que a bateria do carro ficou sem carga porque as luzes estiveram acesas, por engano durante toda a noite, e é necessário esperar uma hora para o arranque do carro (tarefa não prevista mas necessária). E em 2 de Setembro, mudou-se os óleos, mais uma vez, tendo esta operação levado duas horas de *downtime* previsto. A partir deste período o carro funcionou até ao início do ano seguinte sem nenhuma avaria.

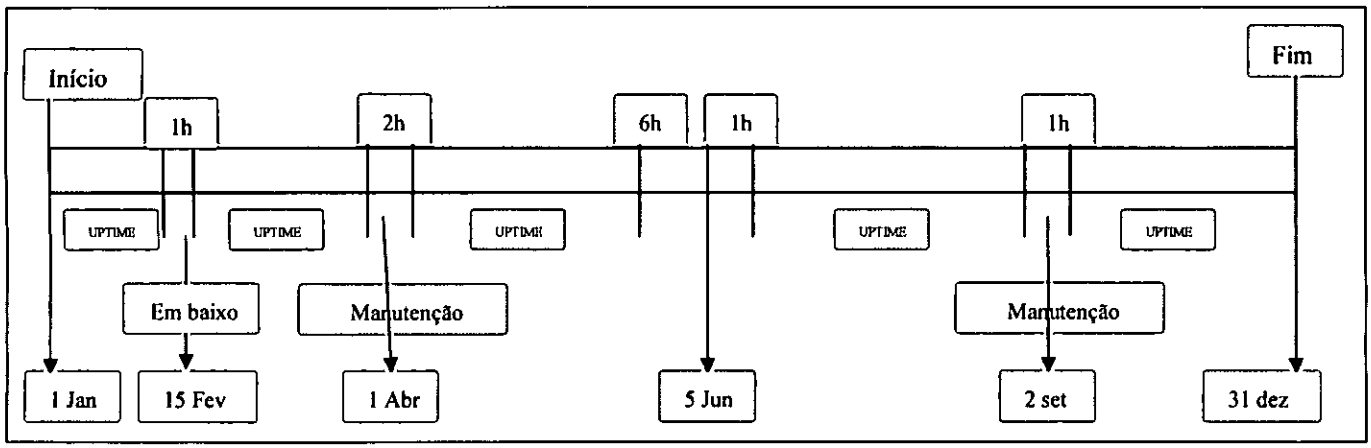


***Figura 1**: MTBF e MTTR para um carro novo*

A figura 1 mostra uma experiência com um carro em funcionamento durante um ano. O *downtime* relativo a descarga da bateria não é de 1 hora, porque mesmo que não tivesse necessidade de usar o carro no período em que a bateria descarregou na noite do dia 4 de Junho até a manhã de 5 julho, o carro esteve indisponível para o uso dentro do período de 4 a 5 de junho.

Segundo a Fig.1, o carro ficou disponível 8.748 horas e não disponível 12 horas num período de um ano, que corresponde a uma média de 99.6%.

Disponibilidade = 100%Uptime*/(*uptime* + *downtime*) = 100*8.748/(8.748 + 12) = 99.86%**

Similarmente, no período de um ano (8.760 horas), verificaram-se quatro períodos de interrupção do uso do carro, dois planeados e dois não planeados, e 8.748 horas de tempo útil do carro em funcionamento, assim o Tempo médio entre falhas (MTBF) fica:

MTBF = 8.748/4 = 2.187 horas

O cálculo do MTTR pôde ser considerado não real, porque inclui no tempo de reparação não apenas o tempo requerido para a manutenção não calendarizada (1h para o pneu e 5 minutos para a bateria sem carga), mas também todo o tempo em que o sistema não esteve disponível para a operação, a saber:

- Uma Hora para o pneu no dia 15 de fevereiro;
- Cinco Minutos para o arranque do carro em 5 de Junho;
- Cinquenta e cinco Minutos para o mecânico realizar a manutenção em 5 de Julho;
- Seis Horas em que a bateria esteve inoperacional; e
- Quatro Horas de manutenção calendarizada.

Assim:

MTTR = (1:00 + 2:00 + 6:00 + 0:55 + 0:05 + 2:00) /4 = 12 /4 = 3 horas

Para os casos em que o SAD inclui uma *Uninterruptible power supply* (UPS) para promover disponibilidade, o cálculo do MTTR não seria calculado pela forma acima, mas

sim pela determinação da probabilidade em que esta componente estará em funcionamento quando for necessário:

$$\textbf{Probabilidade}_{\text{Estar a funcionar}} = e^{-\lambda t}$$

Onde:

- t é o tempo decorrido desde o último período em que a UPS esteve a funcionar;
- λ Representa a taxa média de falhas da UPS

$$\textbf{MTBF} = 1/\lambda$$

Então:

$$\textbf{Probabilidade}_{\text{Estar a funcionar}} = e^{-t/MTBF}$$

(Jones, 2001)

4.2 Tipos de *Clusters*

Cluster é o nome dado a um sistema montado com mais de um computador, cujo objectivo é fazer com que todo o processamento das aplicações seja distribuído aos computadores e, parecendo que é um único computador.

Cada computador de um *cluster* é denominado nó, ou nodo. Todos devem ser interconectados, de maneira a formarem uma rede de qualquer topologia. Esta precisa de ser criada de uma forma que permita o acréscimo ou a retirada de um nó (em casos de danos, por exemplo), mas sem interromper o funcionamento do *cluster*. O sistema operativo usado nos computadores deve ser do mesmo tipo, isto é, somente Windows, *Linux*, Solaris, etc. Isso porque existe particularidades em cada sistema operativo que podem impedir o funcionamento do *cluster*. Independente do sistema operativo usado, é preciso usar um *software* que permita a montagem do *cluster* em si. Assim, este *software* será responsável pela distribuição do processamento. Esse é um ponto crucial na montagem de um *cluster*. É necessário que o *software* trabalhe por forma que os erros e defeitos sejam detectados, oferecendo meios de providenciar a sua manutenção, mas sem interromper as actividades do *cluster*. Deste modo, os tipos de *cluster* são:

- **Alta Disponibilidade ("*High Availability and Failover*")** - Estes modelos de "*clusters*" são construídos para prover uma disponibilidade de serviços e recursos de forma ininterrupta através do uso da redundância implícita ao sistema. A ideia geral é que se um nó do "*cluster*" vier a falhar, as mesmas aplicações ou serviços possam estar disponíveis em outro nó (*failover*). Estes tipos de "*cluster*" são utilizados para aplicações de missões críticas como correio electrónico, Servidores de bases de dados, ficheiros e aplicações.

- **Balanceamento de carga ("*Load balancing*")** - Este modelo distribui o tráfego ou requisições de recursos provenientes dos nós que executam os mesmos programas entre os Servidores que compõem o "*cluster*". Todos os nós são responsáveis em controlar os pedidos. Se um nó do *cluster* falhar, as requisições são redistribuídas entre os outros nós disponíveis no momento. Este tipo de solução é normalmente

utilizado em Servidores de páginas, de aplicações e outros (ver figura 2), podendo ser efectuado ao nível do *swhich* ou dos Servidores.

- **Combinação Alta Disponibilidade & Balanceamento de Carga** - Este tipo de *cluster* combina as características dos dois tipos de "*cluster*", aumentando assim a disponibilidade e escalabilidade de serviços e recursos. Este tipo de configuração de "*cluster*" é bastante utilizado em Servidores de páginas, Correios electrónicos, notícias ou transferência de ficheiros (FTP) e base de dados.

- **Processamento Distribuído ou Processamento Paralelo** - Este modelo de "*cluster*" aumenta a disponibilidade e perfomance para as aplicações, particularmente as grandes tarefas computacionais. Uma grande tarefa computacional pode ser dividida em pequenas tarefas que são distribuídas ao redor das estações (nó), como se fosse um supercomputador massivamente paralelo. Estes "*clusters*" são usados para computação científica ou análises financeiras, tarefas típicas para exigência de alto poder de processamento.

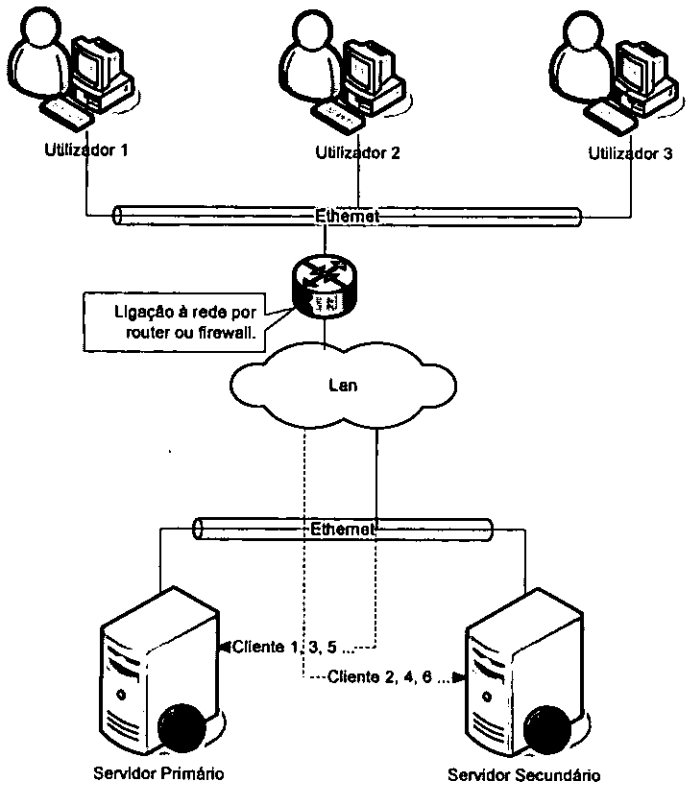


Figura 2: Exemplo de balanceamento de carga por Round Robin

4.3 Técnicas de *Cluster*

O *cluster* pode ser classificado quanto a:

- **Classe (I e II)** - O *cluster* de classe I é construído inteiramente usando-se *hardware* e *software* de tecnologia padrão, como SCSI (*small computer system interface*), *Ethernet*, fibra óptica e IDE (*integrated device electronics*), ou seja, comum e, são tipicamente mais baratos do que o *cluster* de Classe II, que utilizam *hardware* e *software* especializado e tem mais desempenho.

- **Arquitectura (Simétrica e Assimétrica)** - Na arquitectura simétrica, todos os nós do *cluster* são homogéneos, visto que possuem a mesma velocidade e capacidade de processamento, além de possuírem a mesma quantidade de recursos computacionais (Exemplo: memórias). Os *clusters* com este tipo de arquitectura possibilitam uma verdadeira análise de desempenho, por esta razão foi aplicada ao trabalho em curso.

E na arquitectura assimétrica, os nós que constituem este tipo de *cluster* são diferenciados uns dos outros. Podem possuir homogeneidade, mas com velocidades e capacidades de processamentos diferentes, ou nós homogéneos com diferentes recursos computacionais. Este tipo de arquitectura dificulta possíveis análises de desempenho.

4.3.1 *Single Point of Failure* (SPOF)

Segundo Brookman (2003), tem-se a visão de que os *clusters* de alta disponibilidade são muito utilizados para manter serviços disponíveis de forma ininterrupta. Portanto, na construção de *cluster*, deve haver redundância de *hardware* e a reconfiguração do *software* de *cluster* em todos os nós que compõem, afim de evitar único ponto de falha (*Single Point of Failure* - SPOF), que poderá ocasionar a queda do sistema e a indisponibilidade de serviços providos pelo *cluster*.

O SPOF (ou o único ponto de falha) é uma situação crítica e de interesse essencial ao funcionamento de SAD. Uma falha em um SPOF pode representar a paragem do sistema, por ele definido. Para atingir uma alta disponibilidade, é preciso eliminar quantos SPOFs possíveis, fazendo com que o recurso deixe de ser único e essencial.

A eliminação de SPOF´s geralmente fica perante a redundância de recursos para deixarem de ser únicos como os discos duros, placas de rede, processador, fonte de alimentação, etc. Mas os discos duros são identificados como os principais recursos, visto que a perda de dados é mais prejudicial do que a do *hardware*.

Uma solução de SPOF´s é a realização de cópias de segurança, entretanto esta não caracteriza a alta disponibilidade, porque não aumenta directamente o *uptime* do sistema. Um exemplo de eliminação de SPOF´s em alta disponibilidade é a utilização de *software* ou *hardware* redundante, como o RAID.

4.3.2 *High Availability* (Alta Disponibilidade)

São *clusters* desenvolvidos para gerar alta disponibilidade de serviços e recursos, de maneiras a tornar o uso por intermédio da redundância, que é subentendido o sistema. De modo geral, se um nó do *cluster* tiver algum problema ou falha, os serviços e as aplicações, transparentemente sem percepção do utilizador, estarão disponíveis em outro nó. Esta técnica foi desenvolvida a partir da necessidade de não deixar o sistema parar caso um servidor, de forma inesperada, deixar de funcionar.

Esta é a técnica aplicada para o trabalho em curso, porque os *clusters* de alta disponibilidade possuem uma ligação directa com a necessidade referida acima, em função da crescente dependência que as organizações e pessoas têm dos computadores. Eles assumiram, cada vez mais, um papel de extrema importância.
Estes sistemas de *cluster* de HA mantêm a disponibilidade dos recursos e serviços prestados em um sistema computacional. Baseiam-se na replicação dos recursos em outro servidor, por redundância e configurações de *softwares* para realizar esta tarefa. Na realidade, a ideia principal de *cluster* baseia-se em vários Servidores que interagem como um único servidor, tendo a função de monitorar uns aos outros.
A grande complexidade está no *software*, que tem a função de monitorar os recursos no *cluster*, verificando se os serviços estão sempre *online* e, saber como realizar a substituição do serviço em caso de falha.

Segundo Fesurv (2004), periodicamente este *software*, conhecido como *keep alive*, envia uma mensagem de um nó ao outro, se o primário responder ao secundário, e vice-versa, significa que estão em perfeito funcionamento ou estão disponíveis. Caso contrário, constata-se que um falhou e este será substituído pelo se encontra em funcionamento. Para que a substituição seja bem sucedida, todos os clientes comunicam-se com o *cluster* por um endereço IP público (como a figura 5 na página 24 ilustra), para facilitar a migração do nó com falha para o que está em funcionamento, afim de garantir a continuidade do processamento das aplicações.

4.3.3 Load Balance (Balanceamento de Carga)

O tipo de *cluster* baseado em balanceamento de carga tem a função de distribuir o tráfego e requisição de Servidores que estão no sistema. Pode ocorrer de duas formas:

- O sistema é exclusivo para a realização do balanceamento de carga, ou seja, os nós não são utilizados em processamentos de aplicativos individuais. Nessa situação existe um nó responsável por manter e controlar as aplicações e pedidos;
- O sistema realiza o balanceamento de carga e executa os aplicativos dos utilizadores, desse modo os nós também são responsáveis em manter e controlar as solicitações e pedidos.

Nesta estrutura de *cluster* são interligados os nós para suprir todas as solicitações de recursos e as requisições enviadas de clientes, sendo distribuídas de uma forma rápida e equilibrada entre os nós. Este tipo de *cluster* não trabalha ao mesmo tempo em um único processo, fazendo uma espécie de redireccionamento das solicitações de forma uniforme, definido em um algoritmo específico de escalonamento.
Caso algum nó deste *cluster* falhar, o tráfego deve ser rapidamente redistribuído entre os Servidores que estão mais disponíveis, caso ocorra a falha referenciada no algoritmo utilizado.
Este tipo de *clusters* aplica-se em Servidores de páginas, DNS (*Domain Name Service*), e também ao nível de *routers e switches*.

Para evitar SPOF´s neste tipo de *cluster* existem vários tipos de algoritmos de balanceamento de carga, nomeadamente: *Least Connection, Round Robin, Weighted Fair, e* Combinação de Alta disponibilidade e balanceamento de carga.

a) Algoritmo *Least Connection*

Com esta técnica, são redireccionadas as solicitações para o servidor virtual, baseado no menor número de solicitações por conexões. Por exemplo, se nesse instante o servidor 1 estiver a processar 30 solicitações por conexões, e o 2 estiver processar 20 solicitações por conexões, com certeza a próxima solicitação será redireccionada automaticamente

para o servidor dois, desde que, nesse mesmo instante, o servidor não tenha um número maior de processos activos.

b) Algoritmo *Round Robin*

Com esta técnica, as requisições são sempre redireccionadas para a próximo servidor do *cluster,* que estará na sua vez de receber estas solicitações, disponibilizando de forma circular.

Exemplificando: se existirem dois Servidores que atendem as solicitações no sistema, e a primeira solicitação de conexões for encaminhada para o servidor 1, posteriormente a próxima para o servidor 2, a seguir para o servidor 1, e assim sucessivamente, como mostra a figura 2 (*pag. 15*), melhora o desempenho do sistema e elimina a concentração de solicitações em um único servidor, este tipo de técnica é maioritariamente aplicado em Servidores de DNS.

c) Algoritmo *Weighted Fair*

Nesta técnica, as solicitações de conexões são dirigidas para os Servidores do *cluster* referencialmente baseadas na capacidade de carga e requisições em cada um dos Servidores e, na disposição de resposta deles mesmo. Exemplo: se o servidor 1 é cinco vezes mais rápida para a realização de atendimento às solicitações do que o servidor 2, os administradores referenciam uma capacidade de carga maior para o servidor 1 do que referenciado ao servidor 2.

d) *High Availability* (Alta Disponibilidade) e *Load Balancing* (Balanceamento de Carga)

Esta técnica, faz a combinação das duas técnicas de *cluster* (HA e balanceamento de carga), com o objectivo de aumentar a capacidade de disponibilidade e de escalabilidade das requisições, dos recursos e dos serviços. Este tipo de estrutura também é muito utilizada em organizações que dispõem de Servidores de páginas *Web, correio electrónico, ou Ftp.*

Exemplo: Haverá o redireccionamento das requisições do nó com problema para os nós de cópia de segurança, melhoria na qualidade das aplicações e serviços disponíveis na rede, mantendo uma transparência na relação das aplicações em uma única rede virtual, e uma estrutura que mantêm o sistema altamente escalável.

e) Processamento paralelo

O *cluster* de processamento paralelo requer aplicações paralelas para se obter um bom desempenho, mas também podem ser aplicações sequenciais, porém nesse caso a maioria dos recursos do *cluster* não é actualizada. Entretanto, para a utilização de aplicações que são paralelas, tem-se os API´s (*Application programming Interface*) que representam um conjunto de funções e sub-rotinas utilizadas pelos programas que informam ao sistema operativo como executar determinadas tarefas, utilizando diferentes ambientes de programação, visto que estes correspondem principalmente a bibliotecas paralelas (MPI[1], PVM[2], etc.) com a utilização de depuradores, monitores e linguagens de programação adequadas a interagir com a estrutura para fazer a comunicação na rede e fornecer um bom desempenho nas aplicações.

[1]MPI – *message passing interface*, biblioteca para comunicação paralela, para troca de mensagens.

[2]PVM – *Paralele Virtual Machine*, biblioteca pública para comunicação paralela, para troca de mensagens.

4.4 Que Benefícios de *Cluster*?

Dentre vários benefícios que o *cluster* pode disponibilizar, o destaque vai para os seguintes:

- Segundo Christensen (2003), o *cluster* em Servidores permite que os clientes continuem a aceder os aplicativos e os recursos na rede no caso de falhas de aplicações ou *hardware* e interrupções planeadas. Se um dos Servidores no *cluster* não estiver disponível devido a uma falha ou aos requisitos de manutenção, os recursos e os aplicativos serão movidos pelos *software* do *cluster* para o outro nó disponível, mesmo que o sistema esteja em produção e não afectará o utilizador final.

- Oferece um nível mais alto de flexibilidade e recuperação. Os Servidores com tolerância a falhas normalmente usam um alto grau de redundância de *hardware*, além de *software* especializado, para fornecer uma recuperação quase instantânea de qualquer falha de *hardware* ou *software*. Claro que se o próprio *software* de *cluster* falhar o SAD não estará disponível.

- O "*Cluster*" pode ser desenhado com escalabilidade, muitos sistemas podem ser adicionados com o aumento dos requisitos, que espalha a carga através dos subsistemas múltiplos ou dos Servidores.

- **Alta disponibilidade**: Com os *clusters* de servidor, a propriedade dos recursos, como unidades de disco e endereços IP público, é automaticamente transferida de um servidor com falhas para um servidor disponível, sendo que os endereços de *heartbeat* mantêm-se.

Quando há falha num sistema ou num aplicativo no *cluster*, o *software* reinicia este aplicativo com falha no nó disponível ou distribui o trabalho do nó com falha para os restantes, consequentemente, o usuário terá apenas uma pausa momentânea no serviço.

- ***Fallback***: O serviço de *cluster* atribuirá novamente, de forma automática, a carga de trabalho em um *cluster* quando um servidor com falha estiver *online* para seu proprietário predeterminado. Esse recurso pode ser configurado, mas está desactivado por padrão.

- **Capacidade de Administração**: É possível usar ferramentas específicas para a administração do *cluster* a partir da consola de gestão, permitindo mover os recursos de um servidor para o outro no *cluster*. Esta ferramenta permite também equilibrar as cargas de trabalho do servidor manualmente e libertar os Servidores para uma manutenção planificada. É possível também monitorar o estado do *cluster* em todos os nós e os recursos envolvidos neste ambiente.

- **Escalabilidade**: Os serviços de *cluster* podem aumentar para atender à crescente demanda dos serviços, para o efeito pode-se usar a escalabilidade Vertical (*Scale-up*) e Horizontal (*Scale-out*).

  A finalidade da escalabilidade horizontal é aumentar o desempenho, por exemplo, em Servidores de páginas, ou a disponibilidade (*Cluster* de HA). Os sistemas horizontalmente escalados são caracterizados pela capacidade de adição de mais Servidores no ambiente. Embora as redes e o *Cluster* continuem a melhorar o seu desempenho em termos de largura de banda elevada e da latência baixa, a sua velocidade ainda é baixa pelas potencialidades de barramento e as interconexões de utilizadores via *Swhich*.

  A finalidade da escalabilidade vertical, é diminuir o tempo de execução de processos computacionais. Estes sistemas são caracterizados por um único utilizador, vários CPU's múltiplos, a funcionar em uma única cópia do sistema operativo por exemplo. Não há nenhuma sincronização entre processos, todos os recursos são compartilhados.

  Resumidamente, a escalabilidade vertical, tem a haver com o transporte das aplicações a um sistema maior, com mais capacidade, atende prioritariamente às necessidades de perfomance. E a escalabilidade horizontal, atende a adição de mais recursos de alta disponibilidade, além de maior perfomance.

4.5 Necessidades de Alta Disponibilidade

Uma rede de dados é uma chave fundamental para um sistema de Alta Disponibilidade. Ora, actualmente muitos fabricantes garantem o tempo médio entre falhas (MTBF) para o funcionamento do *hardware* que compõem o equipamento computacional, incluindo a própria cablagem. Para garantir melhores serviços, é necessário que os serviços disponibilizados aos clientes sejam fiáveis. Uma rede estruturada tem vários tipos de componentes: os passivos e os activos. Enquanto os passivos (cablagem propriamente dita blindada ou não, ou com troços de fibra óptica) já têm normalmente alguma redundância de pontos terminais, os activos (*Switches* e *Routers*) embora bastante fiáveis, são normalmente instalados sem qualquer duplicação. Se falha um elemento activo numa rede falham, obviamente, todos os postos de trabalho que dele dependem. Nestes ambientes, não está prevista nenhuma redundância de *hardware*, se a placa mãe, o processador, a fonte de alimentação, o disco duro ou a carta de rede do servidor em uso avariar por alguma razão, todos os utilizadores na rede não poderão acedê-lo (ver fig. 3).

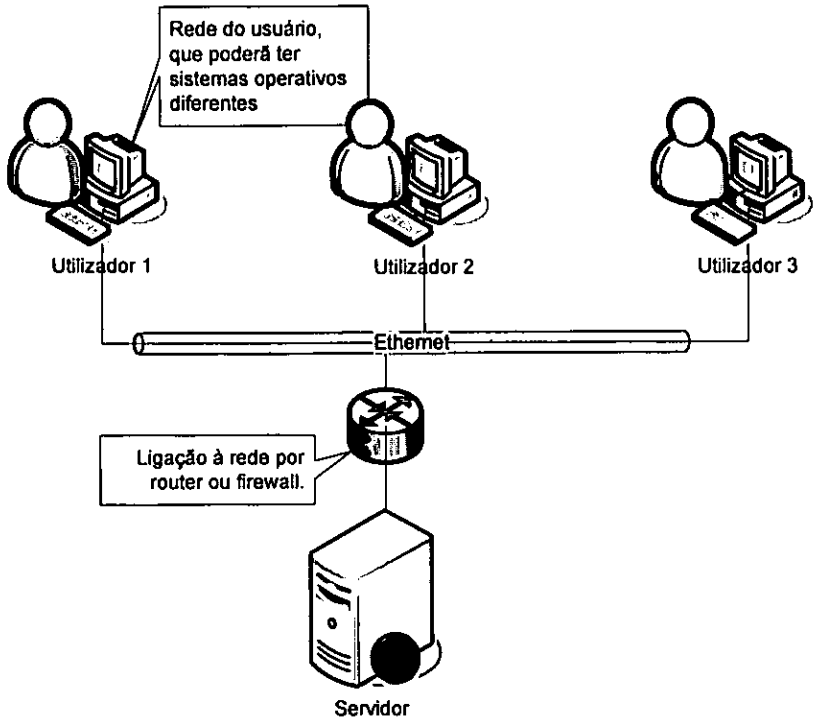


Figura 3: Rede sem nenhuma redundância

Se for feito redundância ao nível da placa mãe e do processador central (CPU), verifica-se que o sistema apresenta ainda falhas, que podem ser de: disco duro ou carta de rede, possibilitando ainda que se um dos componentes falhar, o sistema poderá não estar disponível aos utilizadores na rede. (ver figura 4)

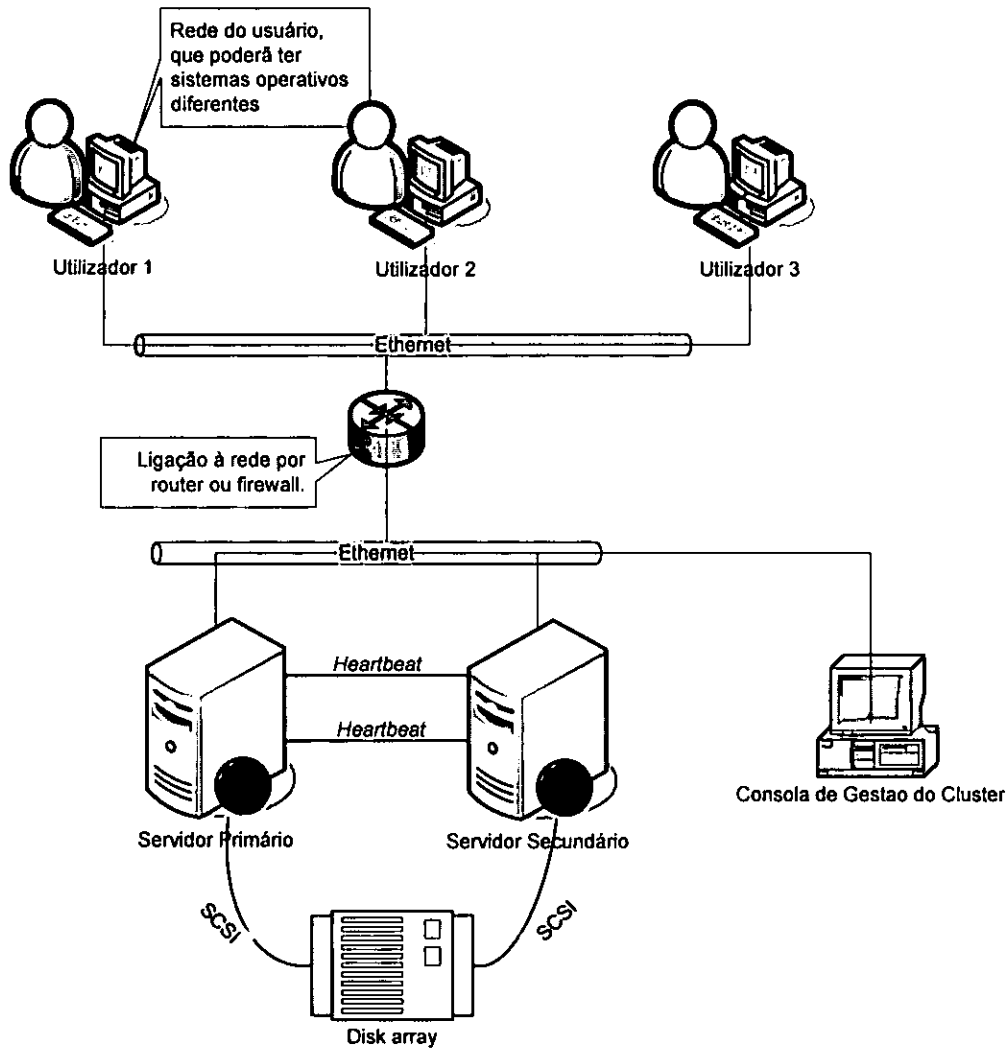


Figura 4: Rede com redundância de placa mãe e processador

Neste caso, se for feita a redundância do disco duro e da carta de rede, pode resultar numa disponibilidade completa do sistema, como a figura 5 ilustra.

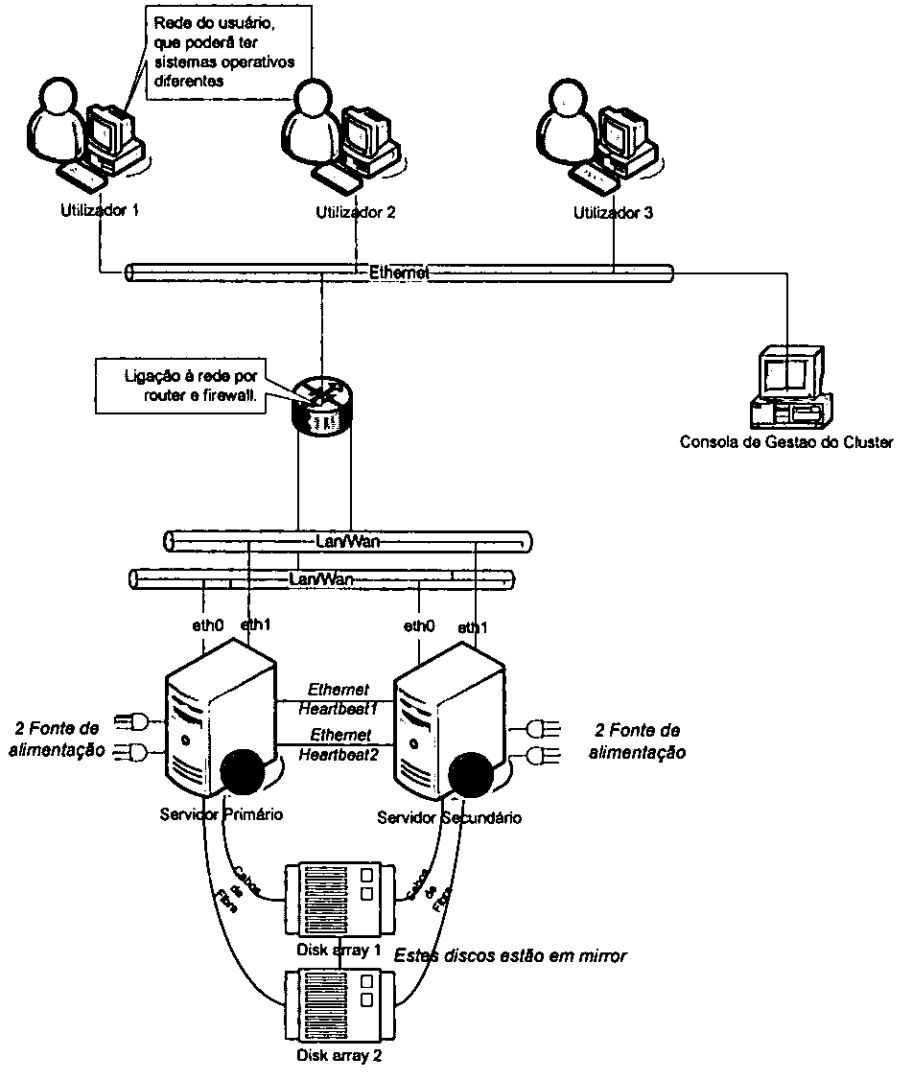


Figura 5: Rede com disponibilidade completa

4.6 Vantagens de Usar *Linux* para o *clustering*

Entretanto, existem inúmeras vantagens em utilizar *clusters*, e Pitanga (2004) destaca o seguinte:

- ***Alto Desempenho*** – possibilidade de resolver problemas complexos através do processamento paralelo, diminuindo o tempo de resolução do mesmo;
- **Escalabilidade** – possibilidade de adicionar novos componentes a medida que cresce a carga de trabalho;
- ***Tolerância a Falhas*** – o aumento da confiabilidade do sistema como um todo, caso alguma parte falhe;
- ***Baixo Custo*** – a redução de custo para se obter processamento de alto desempenho utilizando simples computadores; e custo zero para o sistema operativo e ferramentas de configuração final (são retirados da Internet gratuitamente);
- ***Independência de fornecedores*** – utilização de *hardware* aberto, *software* de uso livre e independência de fabricantes e licenças de uso.
- O *Linux* é *OpenSource*, embora actualmente exista o *Linux Redhat Enterprise Server* que possui licença.
- A maior parte dos principais Sistemas Operativos (SO) suportam o "*clustering*", com a excepção da *Sun Microsystems*, HP e outros que possuem uma linhagem própria de seus produtos.
- O *Linux* é independente do tipo de *hardware* e fabricante e é conhecido pela sua portabilidade.
- Implementar um *cluster* é bem fundamental, e garante segurança de dados.
- O *Linux* obedece os padrões OSI da ISO e é compatível com outros sistemas operativos e diferentes fabricantes de *hardware*.

4.7 Desvantagens de Usar *Linux* para o *clustering*

Embora o *Linux* tenha muitas vantagens para o "*cluster*", também possui falhas que podem torná-lo não atractivo a certos tipos de "*clusters*", como:

- O nível de suporte ao *cluster* não é tão robusto como pode ser encontrado em outros sistemas operativos com linhagens próprias para *cluster*.
- Alguns erros ainda são inerentes com o sistema e o *kernel*. O sistema de ficheiros nativo, não suporta o "*journaling*" (ext2). Tendem a ser uma quantia menor de "*drivers*" para o *Linux* em comparação com outros sistemas operativos, embora as soluções mais comuns possam ser obtidas.
- No *Linux*, o *cluster* é configurado em *hardware* que por natureza não foi idealizado para efeito;
- Existe pouco *software* para o *cluster*;
- Nem todas as aplicações podem ser paralelizadas;
- O nível de suporte neste ambiente é oneroso e pode não estar disponível ou demorar mais tempo.

5. Como Aumentar a Disponibilidade de Sistemas

Há várias soluções para contornar falhas de sistemas computacionais, das quais o destaque vai para:

a) ***Hardware Especializado***: Existem no mercado várias alternativas de *hardware* mais especializado que prevê uma possível falha, tratando isto como redundância de recursos. Outros são voltados para a disponibilidade, com *software* que executa balanceamento de carga, redundância de *links*, tomada de serviços, monitorização, *backup* em tempo real, etc. Mas tais sistemas, quanto mais especializados, mais expansivos são. E também, têm poderes computacionais frequentes muitas vezes maior do que o necessário.

b) ***Sistemas de Tolerância a Falhas***: Os aplicativos evitam que falhas se tornem erros e, erros se tornem defeitos. As falhas dizem respeito ao universo físico. Os erros ocorrem quando estas falhas mudam o valor de algum dado. Acredita-se que a melhor maneira de tratar um problema é logo no seu começo e, o melhor é tratar o erro na hora em que ocorre, evitando danos maiores, como a corrupção e/ou perda de dados. Porém, sistemas assim são extremamente trabalhosos e complexos de serem desenvolvidos, e requerem um profundo e especializado conhecimento.

c) ***Alta Disponibilidade***: Prevê a redundância total do sistema, actuando em caso de defeito, quando o servidor trava. A isso é chamado de defeito, onde um erro causa queda do sistema, afectando os seus clientes. Trabalha com a hipótese de completa paragem do sistema, através de uma solução em redundância de *hardware* e controlo via *software*. Um sistema de alta disponibilidade prevê tal comportamento com *hardware* simples, deixando a complexidade e o controlo da disponibilidade do sistema para o *software*.

6. Sistemas de Ficheiros para Alta Disponibilidade

Em alta disponibilidade, recomenda-se o uso de sistemas operativos que possuam sistemas de ficheiros seguros e relativamente independentes de acções por agentes humanos na recuperação de falhas. Actualmente, as principais abordagens empregues na obtenção de alta disponibilidade em sistemas de ficheiros são:

- *Soft update* (disponível em sistemas UNIX-BSD); e
- *Journaling (usada* pela maioria dos sistemas operativos).

6.1 Abordagem Baseada em Journaling

O conceito de "*Journaling*" foi introduzido na área de base de dados com o propósito de assegurar a consistência e a integridade dos dados durante as transações, na ocorrência de falhas. Eles armazenam cada uma das operações realizadas sobre os seus registos em uma área do disco, denominada de *journal ou log*, antes de serem armazenados permanentemente no disco, e na ocorrência de falha conseguirem retornar ao seu último estado consistente.

Sistemas baseados em "*Journaling*" usam o mesmo princípio, fazendo um controlo sobre as mudanças realizadas nos meta-dados associados ao sistema de ficheiros.

6.2 Implementação de *Journaling*

Na implementação de "*journaling*", reserva-se uma região "*log ou journal*", onde as operações de escritas realizadas sobre o sistema de ficheiros devem ser registadas como sendo transações. As mudanças sempre devem ser registadas, primeiramente, na região de *log* – isto é "*journaling*", e após este registo as transações devem ser aplicadas ao sistema de ficheiros

6.3 Vantagens do Sistema de Ficheiros Baseados em *Journaling*

Os sistemas de ficheiros baseados em *journalling* incluem as seguintes vantagens:

- A recuperação é extremamente rápida na ocorrência de falhas, pois esta tarefa consiste em examinar as transações pendentes no *log*.
- Flexibilidade, pois o número de *inodes* em cada unidade é criado dinamicamente (sem a necessidade de manutenção em tabelas).
- O desempenho é optimizado, pois o registo em *log* das transações de escrita pode ser feito sequencialmente.
- O desempenho pode ser ainda mais optimizado se o *log* for mantido em outra unidade de disco.
- Acesso optimizado aos arquivos e directórios, através de algoritmos *B+Tree* (para JFS, XFS e *ReiserFS*).

6.4 Características dos Principais Sistemas de Ficheiros baseados em *Journaling*

Pode-se encontrar os seguintes sistemas de ficheiros em *journaling*: *ext3*, *ReiserFS*, XFS e JFS.

6.4.1 Sistema de Ficheiros *ext3*

O sistema de ficheiro ext3 é uma alternativa do sistema de ficheiros ext2 que deseja obter as vantagens do "*journaling*", permite a conversão do sistema de ficheiro ext2 para ext3 sem necessitar de reformatar o disco. Desde a versão 2.4, encontra-se disponível como parte integrante do "*kernel*" no sistema operativo *Linux* e o seu suporte encontra-se em várias distribuições *Linux*. O suporte pode também ser adicionado através de uma recompilação do "*kernel*" com os seguintes parâmetros:

CONFIG_EXT3_FS=m ou CONFIG_EXT3_FS=y.

Permite o uso de "*journal*" para dados, meta-dados ou ambos e não suporta a alocação dinâmica de "*inodes*".

Ext3 = ext2 + journal

É o sistema de ficheiros que está implementado no trabalho por razões que a seguir são descritas.

Vantagens

- **Disponibilidade**

O ext3 não requer uma verificação do sistema de ficheiros (fsck) após o reinicio do servidor, excepto em casos previstos. Isto é, porque os dados são escritos ao disco de tal maneira que o sistema de ficheiros é sempre consistente. O momento de recuperar um sistema de ficheiros ext3 depois de uma interrupção programada, não depende do tamanho ou do número de ficheiros, mas sim depende do tamanho do "*journal*" usado para manter a sua consistência. O tamanho do *journal* por defeito leva aproximadamente um segundo para restauro (dependendo da velocidade do *hardware*).

- **Integridade**

Usar o sistema de ficheiros ext3 pode fornecer garantias mais fortes sobre a integridade de dados em casos de uma interrupção programada de sistema. Permite escolher o tipo e o nível de protecção que os dados devem ter e, também manter o sistema consistente.

- **Velocidade**

Apesar de escrever alguns dados mais de uma vez, o ext3 é frequentemente mais rápido (*throughput* mais elevado) do que ext2 porque o *journaling* de ext3 optimiza o movimento da cabeça do disco duro. Pode-se escolher de entre as três modalidades de *journaling* para optimizar a velocidade:

Primeira modalidade: *Data=writeback* limita as garantias da integridade de dados, permitindo que os dados velhos mostrem acima nos ficheiros corrompidos, para um aumento potencial na velocidade sob algumas circunstâncias. Esta modalidade, que vem por defeito para a maioria de sistemas de ficheiros *journaling*, fornece

essencialmente as garantias mais limitadas da integridade de dados do sistema de ficheiros ext2 e evita, meramente, a verificação do sistema de ficheiro no tempo do reinicio.

Segunda modalidade: *Data=ordered* (a modalidade por defeito), garante que os dados sejam consistentes com o sistema, e os ficheiros recentemente escritos nunca serão visualizados após o *crash*.

Terceira modalidade: *Data=journal* requer um *journal* maior para a velocidade razoável na maioria dos casos e leva muito tempo para a sua recuperação em casos de interrupção forçada, mas as vezes é mais rápido para determinadas operações da base de dados.

- **Transação fácil**

É fácil de mudar de ext2 a ext3 e ganhar os benefícios de um sistema de ficheiro *journaling* robusto, sem reformatar o sistema, que são:

Primeiro: A instalação do sistema operativo *Linux Redhat* oferece a transação dos seus sistemas de ficheiros ao fazer *upgrade* do sistema.

Segundo: Existe um programa nos sistemas operativos *Linux* de nome *tune2fs*, o qual pode adicionar um *journal* a um sistema do tipo ext2 existente. Se estiver montado enquanto decorrerem transações, o *journal* será visível como um ficheiro com o nome *journal* no directório */root*. Se o sistema de ficheiros não estiver montado, o *journal* estará escondido e não aparecerá no sistema de ficheiros. Executar *tune2fs – j /dev/hda1* e mudar de ext2 a ext3 dentro do ficheiro */etc/fstab*.

6.4.2 Sistema de Ficheiros *ReiserFS*

O sistema de ficheiros *ReiserFS*, apresenta as seguintes características:

- Projectado por *Hans Reiser*, tem maior vantagem de ser o primeiro sistema de ficheiros baseado em *journaling* a ser incluído como parte integrante do *kernel* de *Linux*.

- Utiliza uma abordagem radicalmente diversa da alocação baseada em blocos usada no *ext2* para armazenar as informações, usando árvores balanceadas.
- Ainda não oferece suporte a quotas para usuários e grupos.
- Possui optimizações para trabalhar com ficheiros pequenos, reduzindo a fragmentação.
- O suporte a ele já se encontra disponível em várias distribuições *Linux*, incluindo conectiva (desde a v6.0), *RedHat 7.1*, SuSe Linux 7.0 e *Mandrake 8.0* .
- O suporte também pode ser adicionado através de uma recompilação do *kernel* v2.4 com os seguintes parâmetros:
  – *CONFIG_REISERFS_FS=m* ou
  – *CONFIG_REISERFS_FS=y*

6.4.3 Sistema de Ficheiros XFS

O sistema de ficheiros XFS, apresenta as seguintes características:

- Criado em 1994 pela SGI para substituir outro sistema de ficheiros (EFS), tornou-se disponível para *Linux* em Julho de 2001 pela licença GPL;
- Projectado especialmente para trabalhar com ficheiros grandes (de até 9 mil "peta *bytes*") e directórios com vários ficheiros;
- Pode trabalhar com tamanho de bloco variando de 512 byte até 64 kb;
- Oferece suporte a quotas para usuários e grupos;
- Suporta ACLs no padrão POSIX;
- Suporte ao NFS (*Network File System*) versão 3.0 para cliente e servidor;
- O suporte a ele já se encontra disponível em distribuições *Linux*, incluindo *RedHat 9* e *Mandrake 8.0*;
- Não é parte integrante do *kernel* do *Linux*, mas pode ser adicionado com a aplicação de Pacotes;
- O sistema do tipo XFS fornecem as vantagens do *journaling* ao minimizar o impacto do desempenho de *journaling* na leitura e escrita de transações dos dados. As suas estruturas e algoritmos são ajustados para o registo rápido das transações.

6.4.4 Sistema de Ficheiros JFS

O sistema de ficheiros JFS, apresenta as seguintes características:

- Criado pela IBM para uso em Servidores corporativos;
- Permite o emprego de blocos com tamanho de 512,1024, 2048 e 4096 byte;
- Para um tamanho de bloco de 512 byte, o tamanho máximo de um ficheiro é de 512 Tb;
- Para um tamanho de bloco de 4Kb, o tamanho máximo de um ficheiro é de 4 pentabytes;
- Incorporado na versão 2.5.6 do *kernel* para o sistema operativo *Linux*;
- O suporte a ele pode ser adicionado em versões anteriores do *kernel* com a aplicação de *patches;*
- Já se encontra disponível em distribuições *Linux*, incluindo *Mandrake 8.0.*

6.4.5 Sistema de Ficheiros NILFS

O NILFS (*New Implementation of a Log-structured File System*) é um sistema de ficheiros estruturado em *log*, desenvolvido para a versão de *kernel* 2.6 no *Linux*.
Um sistema de ficheiro de registo estruturado tem a característica que todos os dados do sistema de ficheiros possuem, sendo que, os meta dados são escritos em formato de *log.* Isto melhora extremamente o desempenho, porque há poucas despesas gerais a respeito das buscas do disco. O NILFS tem também as seguintes características específicas:

- É um sistema operativo também livre;
- A administração está baseada no protocolo B para o ficheiro e o *inode* (ver tabela 2);
- Recuperação imediata após o servidor encravar;
- A estrutura de dados é de 64 bits, suporta muitos ficheiros, de tamanhos grandes e discos;
- O módulo de *kernel* é lido sem nenhuma recompilação;

Porém, o NILFS apresenta ainda os seguintes *bugs*:

- O comando *df* indica a quantidade errada de espaço de disco disponível no sistema de ficheiros de NILFS;
- A execução dos comandos *ls* e *du*, mostram um número incorrecto dos blocos do disco alocados a um ficheiro;
- O sistema não suporta grandes processamentos simultaneamente.

7. Implementação de Um ambiente de Alta Disponibilidade

A regra geral para implementar um ambiente de alta disponibilidade, é fazer redundância de componentes do sistema onde são necessários. Todas as partes num servidor são sujeitas a ser único ponto da falha (*SPOF*), nomeadamente, o processador central (CPU), a fonte de alimentação, a placa mãe, a memória principal, os dispositivos internos, isto é, todos os dispositivos que possam tornar o sistema inoperacional.

Entretanto, é possível montar um ambiente de HA com redundância de *hardware* em dois Servidores, em que um é chamado de nó principal e o outro de secundário, sendo que cada um é o espelho do outro.

Para que um servidor espelho assuma o lugar quando o principal falhar, ambos devem possuir os mesmos dados e/ou recursos envolvidos no tipo de serviço oferecido, devem ser configuradas de forma a espelhar os dados, para que em caso de substituição de um sistema por outro, os dados sejam actualizados e consistentes.

Contudo, a montagem deste ambiente envolve:

a) *Hardware*: Redundância de Servidores (requisito), de ligações, de conexão dedicada e de alta velocidade (recomendada).

b) Instalação do Sistema: Os *softwares* devem ser independentes de distribuição. A identidade dos Servidores sobre quem será o nó principal fica a cargo do administrador de sistemas.

c) Consistência dos dados: Os sistemas de ficheiros "*journaling*" armazenam em um "*journal*" *(log)* as acções antes de serem efectuadas. Desta forma, havendo uma interrupção no sistema, seja por queda do próprio sistema ou corte de energia, a recuperação é muito mais rápida.

d) Espalhamento de dados: Com os dados espelhados, os serviços podem ser passados de um servidor para o outro mesmo que o sistema esteja em produção e não afectará o usuário final. Porém, o sistema deve ser autónomo nesta transferência, e ser capaz de reconfigurar para continuar atender os usuários.

e) Monitorização: O sistema deve monitorar os seus serviços para detectar defeitos, disparando assim a reconfiguração.

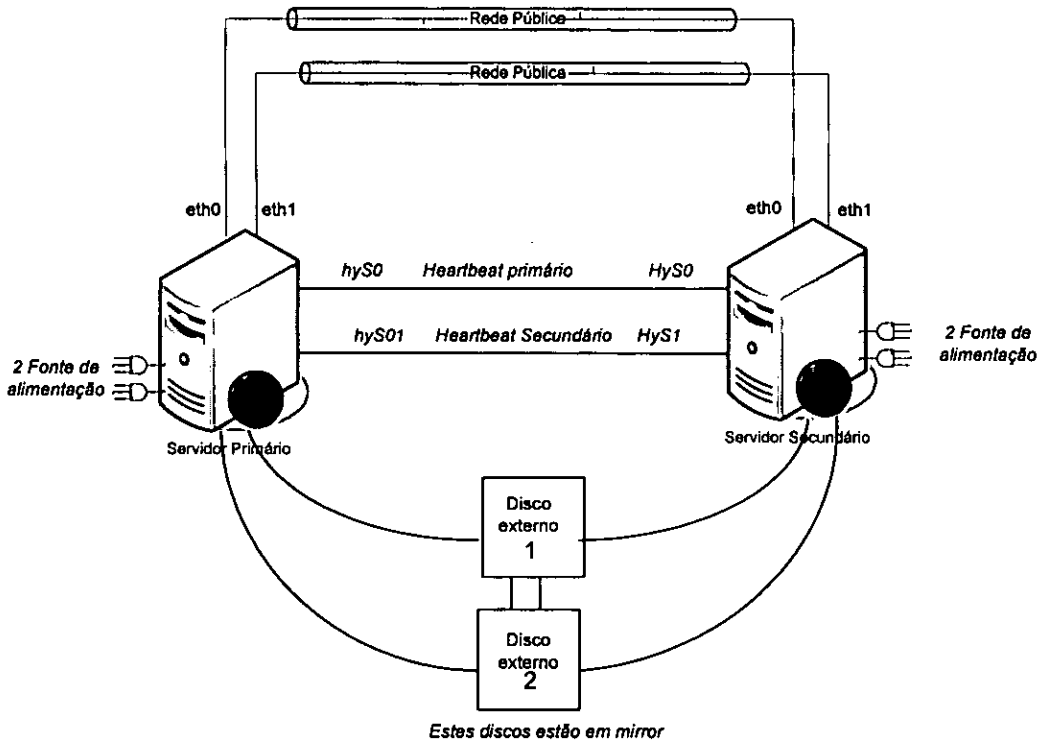


Figura 6: Configuração óptima da Alta disponibilidade

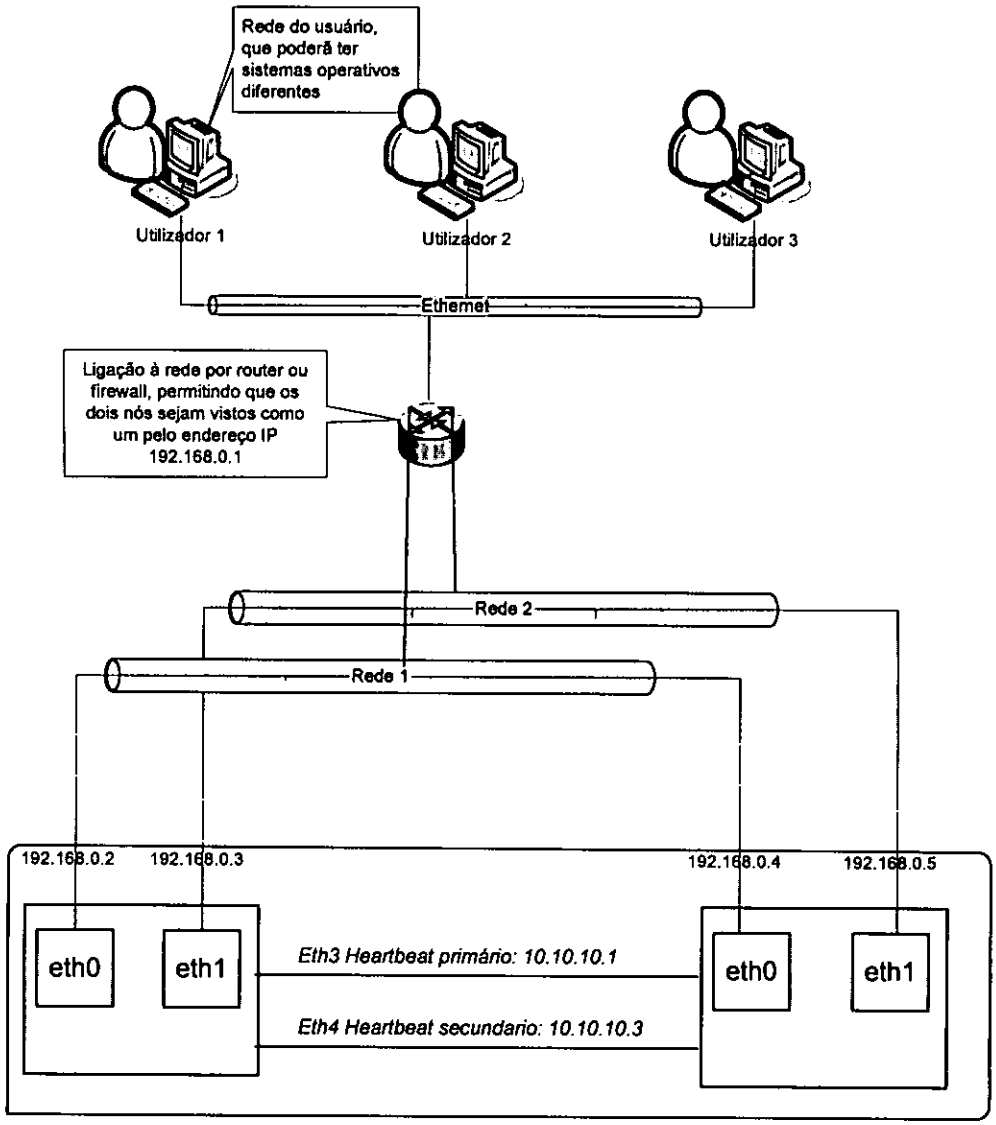


Figura 7: Alocação de endereços virtuais no sistema de alta disponibilidade (SAD)

f) Características de *hardware* e configurações de rede (nó1 e nó2)

PCI *Board* nó1 e nó2

Slot#	Frequência	Descrição
0	1G	Capacidade de memória
-	1Ghz	Capacidade de CPU

Discos locais (nó1 e nó2)

ID	Tipo	Tamanho do disco
0	IDE	73GB

Partição dos discos (nó1 e nó2)

Sistema de ficheiros	Tamanho	*Mount point*
/dev/mapper/VolGroup00-LogVol00	72G	/
/dev/hda1	100M	/boot
None	502M	/dev/shm
/dev/hda2		Swap

Nó1 (cluster1): Configurações de Rede

Interface	Endereço IP	Netmask	Default Router	Observações
eth0	192.168.0.2	255.255.255.0	Opcional	Endereço IP público
eth1	192.168.0.3	255.255.255.0	Opcional	Endereço IP público de redundância
eth2	10.10.10.1	255.255.255.0	N/A	Endereço IP de heartbeat
eth3	10.10.10.3	255.255.255.0	N/A	Endereço IP de heartbeat redundante

Nó2 (cluster2): Configurações de Rede

Interface	Endereço IP	Netmask	Default Router	Observações
eth0	192.168.0.4	255.255.255.0	Opcional	Endereço IP público
eth1	192.168.0.5	255.255.255.0	Opcional	Endereço IP público de redundância
eth2	10.10.10.2	255.255.255.0	N/A	Endereço IP de heartbeat
eth3	10.10.10.4	255.255.255.0	N/A	Endereço IP de heartbeat redundante

7.1 Preparação de Servidores

As características de *hardware* na implementação deste sistema são: Dois Servidores *Pentium IV*, 1GB de memória, 1Ghz de CPU, 1 disco de 73 GB cada.
Esta fase marca o início de uma descrição de procedimentos dos *softwares* na sequência que eles devem ser iniciados no "*boot*" do servidor. Primeiro o *DRBD*, de seguida o "*Heartbeat*", e por último o *Mon* que a seguir serão descritos.

Tendo os dados em uma partição ext2, transformá-las para ext3 é bastante simples, bastando o comando *'tune2fs -j /dev/hdXX'*, onde hdXX é a sua partição. A grande maioria das distribuições actualmente já vêm com suporte a "*ext3 e reiserfs*".

Entretanto, para converter um sistema de ficheiros ext3 a ext2, em que esteja montado ou não, pode-se usar o comando "*debugfs*" de *e2fsprogs-1.17* ou versão anterior
(ver anexo B).

7.2 Instalando e configurando o *DRBD*

DRBD – "*Distributed Replicated Block Device*" – É um módulo de "*kernel*" e "*scripts*" associados que oferecem um dispositivo de bloco projectado para construir "*clusters*" de alta disponibilidade. Desta forma, isto é feito espelhando conjunto de blocos via rede (dedicada). Ele pode ser visto como um RAID via rede local [Garcia 2003].

O DRBD é responsável dos dados escritos no disco rígido local e envia-os a um dos nós, que posteriormente são escritos no disco.

7.3 Funcionamento do DRBD

Cada dispositivo (*drbd* providencia mais de um destes dispositivos) tem um estado, que pode ser primário ou secundário. No nó com o dispositivo primário, a aplicação está em execução e tem acesso ao dispositivo (/dev/nbX). Toda a escrita é enviada para o "dispositivo de bloco do nível mais baixo" e para o nó com o dispositivo secundário. O

secundário apenas escreve o dado no "dispositivo de bloco do nível mais baixo", as leituras são sempre realizadas localmente.
Se o nó primário falhar, o *software* de *heartbeat* permutará o dispositivo secundário em primário e iniciará a aplicação neste dispositivo, caso esteja a ser usado um sistema de ficheiros que não seja *journaling*, implicará a execução do comando "*file system check – fsck*", o que será um processo manual, aumentando deste modo o *downtime* do sistema.

Se o nó que falhou for restaurado no sistema, ele torna-se o secundário e tem que sincronizar os seus dados com o primário. Esta operação é feita sem nenhuma interrupção do serviço em *background*.

Como se encontra o DRBD em "*clusters*" HA actualmente?
A maioria dos actuais "*clusters*" HA (HP, *Compaq*,...) usam dispositivos de armazenamento compartilhados e, desta forma, estes dispositivos são conectados a mais de um nó. A isso pode-se executar com barramento SCSI compartilhado ou usando fibra óptica.
O DRBD usa a mesma semântica de um dispositivo compartilhado, mas ele não necessita de nenhum *hardware* não comum. Ele trabalha no topo de redes ao nível de endereçamento IP, camada de rede do modelo OSI.

Actualmente o DRBD garante acesso de leitura-escrita apenas em um nó de cada vez, o que é suficiente para o "*failover*" usual de um "*cluster*" HA.

7.3.1 Instalação do DRBD

Neste momento, a última versão estável é a 0.6.1 onde seguem os passos normais da sua instalação:

```
root@linux: ~/# tar zxvf  drbd-0.6.1.tar.gz
root@linux: ~/# cd drbd
```

No directório do *drbd*, compilar e instalar o programa:

```
root@linux:~/drbd# make
root@linux:~/drbd# make install
```

Em seguida são feitas as configurações essenciais para a criação dos volumes no ficheiro de configuração "/etc/*drbd.conf*", e este deve ser o mesmo em ambos os nós (ver anexo C).

Para a reinicialização destes serviços, deve-se executar os seguintes comandos em ambos nós:

```
root@linux: ~/# /etc/init.d/drbd stop
root@linux: ~/# /etc/init.d/drbd start
root@linux: ~/# drbdadm all up
```

7.3.2 Configuração via *drbdsetup script*

O "*drbdsetup*" é uma ferramenta de configuração do ambiente do *drbd*. É usado para associar dispositivos *drbd* com dispositivos de bloco de nível mais baixo, configurar pares de dispositivos *drbd* para espelharem os seus dispositivos de bloco, e verificar as configurações dos dispositivos *drbd* que estão configurados. É conveniente primeiro levantar um ambiente via *drbdsetup*, e depois partir para o ficheiro de configuração, para auxiliar na depuração de erros que possam vir a acontecer, além de maior controlo na administração da configuração.

Assume-se que os dois Servidores se chamam nó1 e nó2 com endereço IP 10.10.10.1 e 10.10.10.3 respectivamente, e se usa /dev/hda6 como dispositivo de bloco de nível mais baixo em ambos os Servidores. Entretanto, no nó2, deve-se executar os seguintes comandos:

```
# modprobe drbd
# drbdsetup /dev/nb0 disk /dev/hda6
# drbdsetup /dev/nb0 net 10.10.10.3 10.10.10.1 B -r 10M
```

Na primeira linha é carregado o módulo do *drbd*. Após isto, o primeiro comando indica que o *drbd* vai utilizar a partição hda6 para espelhar. No segundo comando é indicado que o espelhamento ocorre através do interface associada ao IP 10.10.10.3 no servidor, e remotamente com o IP 10.10.10.1, através do protocolo B (*ver tabela 2*).
O parâmetro **-r 10M** indica que a taxa de transmissão (largura de banda) pode chegar a 10Mbs. A vantagem é proporcional ao tamanho da partição, oferecendo uma disponibilidade de 100% de largura de banda. O *drbd* infelizmente leva bastante tempo para sincronizar os dados, pois trata de uma parte bastante sensível do seu sistema. E por padrão, ele vem configurado com uma taxa de transmissão de 256 bits/s. Desta forma, economiza mais tempo.

No nó1 deve-se executar os mesmos comandos, apenas mudando as informações relativas ao nó:

```
# modprobe drbd
# drbdsetup /dev/nb0 disk /dev/hda6 –d 10G
# drbdsetup net 10.10.10.1 10.10.10.3 B -r 10M
# drbdsetup /dev/nb0 primary
```

Este último comando, indica que este nó será o primário. Com a linha de comando "*cat /proc/drbd*", por exemplo, é possível verificar o progresso da sincronização dos dados nos dois nós. Se os nós estiverem em estado de *SyncAll*, é sinal que estão sincronizandos. Quando a mensagem mudar para *Connect*, a sincronização está feita.

Neste ponto, a partição /dev/nb0 pode ser usada como um qualquer outro dispositivo. Mesmo antes da sincronização terminada. Caso seja uma partição nova, deve-se criar a partição com o comando *mkfs* (tamanho de bloco igual a 4096 é recomendado).

```
# mkfs -j -b 4096 /dev/nb0        - para ext3
# mkreiserfs /dev/nb0             - para reiserfs
```

Chama-se a atenção que para transformar a partição em *reiserfs*, primeiro deve ser efectuado a cópia *de segurança* dos dados, e a partir daí pode-se criar a partição novamente, depois passar os dados para a partição nova. Em seguida, montar onde achar conveniente:

```
# mount /dev/nb0 /mnt/mountpoint
```

Nota: Nesta descrição, foi implementado o protocolo B, como pode ser visto na tabela 2. Drbd permite seleccionar o protocolo com o qual irá controlar como os dados serão escritos no dispositivo secundário.

Protocolo	Descrição
A	Uma operação de escrita é considerada completa imediatamente após o dado seja escrito no disco e enviado para a rede.
B	Uma operação de escrita é considerada completa imediatamente após que a confirmação de recepção pelo outro nó chegue.
C	Uma operação de escrita é considerada completa imediatamente após que a confirmação de escrita venha do outro nó.

Tabela 2. *Protocolos DRBD*

O protocolo A é mais leve e mais rápido, porém os protocolos B e C são mais rígidos com os dados. Se o espalhamento for realizado em uma rede fiável e isolada, os protocolos B e C são mais recomendáveis para maior segurança. Se forem utilizados numa rede comum a outros Servidores, o consumo vai ser pesado, sendo melhor o protocolo A.

7.3.3 Configurando via *drbd.conf*

O Drbd também permite a sua configuração pelo ficheiro "*drbd.conf*", organizando os seus parâmetros usando o "*script /etc/rc.d/drbd*", o drbd poderá funcionar perfeitamente mesmo quando os Servidores forem reiniciados (Ver anexo D).

7.3.4 Instalando e configurando o "*Heartbeat*"

O "*Heartbeat*" é usado para definir os pulsos enviados entre dois Servidores, que indicam que estão "vivas", ou seja, estão em funcionamento e disponíveis para executarem tarefas. O código "*heartbeat*" do *Linux*-HA é usado para construir "*clusters*" de altíssima disponibilidade. Ele pode fazer dois nós com capacidade de assumirem os recursos e serviços de um número ilimitado de interfaces. Ele funciona enviando um '*heartbeat*' entre dois Servidores através de uma ligação serial, "*ethernet*", ou ambas. Se o "*heartbeat*" falhar, o servidor secundário irá assumir que a primária falhou, e tomar os serviços que estavam a correr no servidor primário.
O "*Heartbeat*" verifica se o outro nó está a funcionar. Ele pode emitir uma comunicação enviando "*heartbeats*" de um nó para outro, por interfaces de rede. Então, se optar pela serial, temos que verificar se a comunicação serial está bem ou não, depois de conectar os computadores via um cabo cruzado ou serial, é testada a comunicação como a seguir é ilustrado:

```
Linux1# cat </dev/ttyS0
Linux2# echo teste> /dev/ttyS0    (ou substitua ttyS0 pela interface serial)
```

Deverá ser recebido o texto no nó 2, se isso não acontecer, verificar se está a enviar para a serial que realmente está conectada, ou verificar a conexão do cabo.

Para a sua instalação foi utilizado o aplicativo de *heartbeat,* que se encontra na versão 0.4.9.2. Inicialmente é descompactado e compilado como o exemplo a seguir:

```
# tar zxvf  heartbeat-0.4.9.tar.gz
# cd heartbeat-0.4.9
# make
# make install
```

Se ocorrer um erro na compilação do tipo:

file.c: In function `ExpectToken':

file.c:64: `CLK_TCK' undeclared (first use in this function)

Editar o ficheiro *file.c* e acrescentar #include <time.h> (a função usada não está no *sys/time.h*, que deve já estar incluído).

7.3.5 *Scripts* do *Heartbeat*

O "*heartbeat*" traz um "*script*" para integração do *drbd*, cujo nome é "*datadisk*", que se encontra pelo caminho */etc/ha.d/resource.d/datadisk*. O *datadisk* negoceia a troca do dispositivo *drbd* do secundário para o primário, monta as partições *drbd* especificadas no ficheiro *fstab* e pode ser chamado do ficheiro "*haresources*". Ele trata das partições *drbd* especificadas no *fstab*.

Então, se pretender que o "*heartbeat*" tome a tarefa de montar as partições *drbd*, certifique se incluiu a sua configuração no ficheiro *fstab*, como explicado anteriormente.

7.3.6 Configuração

Será criado um directório */etc/ha.d*, onde devem ficar os ficheiros de configuração. Do directório de instalação, copiar os ficheiros *ha.cf*, *haresources e authkeys* do directório *doc* para */etc/ha.d*. Primeiro configurar o *ha.conf* como o anexo E ilustra.

Uma vez configurado o *ha.cf*, é necessário configurar o ficheiro *haresources*. Este ficheiro especifica os serviços para o "*cluster*" e o proprietário padrão. Este ficheiro deve ser igual nos dois nós.

Para o trabalho em curso assume-se que os serviços são o *drbd* e o apache. O endereço IP para o "*cluster*" é obrigatório, e não configurar o IP do "*cluster*" fora do ficheiro "*haresources*". O ficheiro irá precisar de uma linha:

```
Cluster1 192.168.0.2 datadisk httpd
```

Então, esta linha diz que ao iniciar, o servidor cluster1 com o endereço IP 192.168.0.2 inicia o apache e o *drbd*. Ao desligar o "*heartbeat*" irá primeiro parar os serviços do apache, o *drbd* e depois libertar o endereço IP. Isto supõe que o comando '*uname – n*' tem como saída *cluster1*.

Nota: "*datadisk*" e "*httpd*" são os nomes dos "*scripts*" que iniciam o *drbd* e o apache, respectivamente. O "*heartbeat*" irá procurar por "*scripts*" com o mesmo nome nos seguintes lugares:

```
/etc/ha.d/resource.d
/etc/rc.d/init.d
```

Estes *scripts* devem iniciar os serviços via "*nome_script start*" e pará-los via "*nome_script stop*". Então, com estes *scripts* disponíveis segundo as especificações dadas, pode-se usar qualquer serviço no servidor.
Talvez seja necessário passar argumentos para um *script* customizado, o formato deve ser: *Nome_script: argument*.

Desta forma há mais flexibilidade com o endereço IP de serviço. Está-se a usar uma notação abreviada. Esta linha pode ser lida assim:

```
Cluster1 IPaddr: 192.168.0.2 datadisk httpd
```

Onde *IPaddr* é o nome do nosso *script* do serviço, tendo como argumento 192.168.0.2. Sem dúvida, se olhar no directório */etc/ha.d/resource.d*, poderá encontrar um *script* chamando *IPaddr*. Este irá também permitir manipular o endereço de *broadcast* e a máscara de rede do seu serviço IP. Para especificar uma sub-rede com 32 endereços, pode-se definir o serviço como (deixar de fora o *IPaddr* é possível):

```
Linuxha1 192.168.0.3 httpd smb
```

Este procedimento configura o endereço do serviço IP para 192.168.0.3

Às vezes tem criado confusão entre quais das alternativas acima são necessárias para o uso. Para isso depende, se existe uma rota estabelecida (independente do *heartbeat*) para os endereços IP do serviço, com a máscara e *broadcast* correctos, então esta operação não é necessário. Em adição, pode ter-se mais de uma *interface* possível que pode ser usada para o serviço IP.
Não é preciso fazer autenticação neste ambiente, porque não existe nenhum elemento activo ou passivo que possa ser incorporado, visto que trata-se de uma rede completamente fechada e nenhum utilizador irá precisar autenticar-se.

O formato do ficheiro do *sha1* é o seguinte:
Auth <number> <number> <authmethod> [<authkey>]

Então, para sha1, um exemplo do */etc/ha.d/authkeys* pode ser:

```
Auth                                    1
1 sha1 key-for-sha1-any-text-you-want
Para md5, pode se usar o mesmo, apenas trocando 'sh1' por 'md5'.
Finalmente para crc, pode ser:
auth                                    2
1 crc
```

Ter certeza de que as permissões deste ficheiro são seguras e estão com os valores 600, e existe um limite para o número de caracteres que pode se usar.

7.3.7 Seleccionando uma interface

Iniciar o *heartbeat*, cujo *script* está em */etc/init.d/*:

```
#/etc/init.d/heartbeat start
```

Os *logs* dos *scripts* de *heartbeat* são armazenados em */var/log/ha-log*. Caso ocorra algum erro, verificar neste ficheiro as suas causas.

O *heartbeat* irá cuidar agora da inicialização dos serviços configurados no *haresources*. Caso o servidor principal congele, a secundária irá assumir e iniciar os mesmos recursos. Mas se a rede cair, e se estiver a usar *firewall*, ou um servidor Web? Para isso, precisamos de um programa que seja capaz de detectar e executar uma acção de monitorização destes serviços.

7.3.8 Instalando o "*Mon*"

O *software* escolhido para monitorização dos serviços do *cluster* é o "*Mon – Service Monitoring Daemon*" e a sua versão é a 0.99.2. Seguindo a configuração, inicialmente é necessário que alguns módulos do *perl* sejam instalados.

A instalação destes módulos segue o seguinte exemplo:

```
#bzip2 –cd modulo.bz2 | tar xvf –
# cd modulo
# perl Makefile.pl
# make
# make test
# make install
```

O *mon* trabalha com monitores e alertas, estes verificam o estado dos serviços do sistema. Os alertas enviam mensagens ao administrador de sistemas, e o *mon* pode ficar na partição */etc/ha.d/mon/*. Depois de instalar, copiar o ficheiro *example.cf* para o directório escolhido, renomear para *mon.cf* (ver o anexo F).

As interfaces de *heartbeat* estão redundantes e são de cabos cruzados, caso todas interfaces não IP responder, significa que o servidor está isolado e pára o *heartbeat*.
O *mon* envia *fping* para os Servidores definidos, se nenhum responder, é sinal de que o servidor está isolado. Então, ele envia um *email* para o utilizador *root* e desliga o *heartbeat*[1]. Colocar o executável *mon* no directório definido em *cfbasedir*. Feito isso, iniciar o *mon* como o exemplo a seguir:

```
#/etc/ha.d/mon/mon -f -c /etc/ha.d/mon/mon.cf -b /usr/lib/mon
```

Agora, se ocorrer algum erro na rede, como queimar a placa, a porta do *switch*, cabo de rede, seja o que for, o *mon* dispara o *shutdown* do *heartbeat*, que levará a outro servidor a assumir o endereço IP e os serviços.

[1] *O script heartbeat.alert foi desenvolvido por Sandro Poppi (spoppi@gmx.de).*

8. Conclusões e Recomendações

Analisando as técnicas de alta disponibilidade estudadas e apresentadas neste trabalho, pode-se observar que para qualquer actividade que utiliza recursos computacionais nas suas transações, existe sempre um tipo de *cluster* que pode ser implementado para aumento de capacidade, disponibilidade e desempenho desse processamento.

Durante a realização deste trabalho, foi verificado que os *clusters* em ambiente *open source* podem solucionar ao mínimo falhas computacionais de sistemas nas instituições, permitindo deste modo o aumento de desempenho e confiança.

O desenho deste tipo de sistemas, requer que sejam tomados em consideração aspectos relacionados com a disponibilidade de vários componentes de infra-estrutura relacionada com o *hardware*, *software* do *cluster* em *open source* e mecanismos de monitorização, que comparativamente com aplicações de *cluster* de alternativa proprietárias são menos onerosos. Deve-se tomar em consideração todos aspectos relacionados com os únicos pontos de falha (SPOF´s) que possam surgir neste tipo de sistema.
Ha que realçar que, no trabalho em curso o *switch* e *Router* são SPOF´s, visto que não estão redundantes.

Um projecto que tome em consideração os vários parâmetros detalhados neste trabalho pode adequar-se às necessidades de alta disponibilidade nas organizações.

Recomendações:

- Em sistemas de alta disponibilidade usando ferramentas *open source*, recomenda-se que sejam implementados em sistema de ficheiros com *journalling*, visto que assegura a consistência de dados nas transações e não precisa de verificação do sistema de ficheiros após o reinicio do sistema operativo (fsck).

- Em relação a arquitectura do *cluster*, deve ser incorporados *switch* e não *hubs* para a terminação de conexões de rede, e a rede de *heartbeat* seja composta de cabos de rede RJ45 cruzados e não conexão com cabos serial, o ideal seria utilizar ligação com cabos de fibra óptica. Na rede de *heartbeat*, as terminações dos cabos cruzados devem ser do tipo ponto a ponto, e não devem ser efectuados no *switch* e muito menos no *patch painel,* para diminuir a latência nesta rede.

- Recomenda-se a continuação deste trabalho e, mais investigação e divulgação destes tipos de sistemas nas organizações e instituições de ensino.

9. Bibliografia Referenciada

MACOME, Esselina. "*Introdução a metodologia de Investigação*". Maputo: UEM, 1995;

AMARAL, Wanda do. *Guia para apresentação de tese, Dissertações, Trabalhos de Graduação*. 2ª edição. Maputo: Livraria universitária, 1999. 83p.

PITANGA, Marcos (2004). *Construindo Supercomputadores com Linux*. São Paulo: Editora *Brasport*.

BOOKMAN, Charles (2003). *Linux Clustering: Building and Maintaining Linux Clustering*. 2ª edição. USA: New Riders, 2003. 156p.

JONES, Vicent, C (2001). *High Availability Network With Cisco*, Adison Wesley.

BOVET, Daniel P, CEATI, Marcos (2000). *Understanding The Linux Kernel*, O´Reilly

WELSH, Matt, KAUFMAN, Lar, DAWSON, Terry, DALHEIMER, Mathias K (2003). *Running Linux*. 4ª Edição. Sebastopol: O´Reilly, 2003. 614p.

BALL, Bill (1999). *Linux in 24 hours*. 2ª Edição. Indiana: Sams, 1999. 598p.

TAYLOR, Dave, ARMSTRONG Jr, James C. (1997). *Unix in 24 hours*. 1ª edição. Indiana: Sams, 1997. 572p.

FLICKENGER, Rob (2003). *Linux Server Hacks*. 1ª edição. Sebastopol: O´Reilly, 2003. 241p.

HUNT, Craig (1997). *TCP/IP Networking Administration*. 2ª Edição. O`Reilly, 1997. 753p.

SITES Visitados:

SANDRO, Poppi (2001). *Scritpts* de Monitorização.
http://gus-br.linuxmag.com.br/documentacao/ha-slack/heartbeat.alert
[Data da última consulta 16 de Maio de 2005]

LEWIS, Phil (1999). A High-Availability Cluster for Linux
http://www.linuxjournal.com/article.php?sid=3247
[data da última consulta 16 de Maio de 2005]

VIGLIAZZI, Douglas (2002). Alta Disponibilidade *(High Availability)* em sistemas GNU/*Linux*. http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=52
[data da última consulta 17 de Maio de 2005]

GARCIA, Sulamita (2003). Montando um pequeno ambiente de Alta Disponibilidade no Slackware. http://gus-br.linuxmag.com.br/documentacao/ha-slack/
[data da última consulta 16 de Maio de 2005]

GARCIA, Sulamita (2003). Montando um pequeno ambiente de Alta Disponibilidade no Slackware http://ha.underlinux.com.br/.
[data da última consulta 16 de Maio de 2005]

VEPSTAS, Vepstas (1998). Perfomance, Tools & *General Bone-headed Questions*
http://www.linas.org/linux/Software-RAID/Software-RAID-1.html

Rudy Pawul (2003). Getting Started with Linux-HA (heartbeat)
http://linux-ha.org/download/GettingStarted.html

CHRISTENSEN, Elden (2003)
http://www.technetbrasil.com.br/Artigos/Windows2003/ConfClus/
[Data última consulta 12 de Junho de 2005]

BROOKMAN (2003)

http://www.ulbrato.br/ensino/43020/Artigos/Anais2004/Anais/gabrielaLeaoClusterEncoinfo2004.pdf

[Data da última consulta 21 de Novembro de 2005]

FESURV (2004)

http://www.ulbrato.br/ensino/43020/Artigos/Anais2004/Anais/gabrielaLeaoClusterEncoinfo2004.pdf

[Data da última consulta 21 de Novembro de 2005]

Anexos

Anexos A

Heartbeat. Alert

```
#! /usr/bin/perl
#
# Shutdown heartbeat
# derived from Jim Trocki's alert.template
#
# Jim Trocki, trockij@transmeta.com
# Sandro Poppi, spoppi@gmx.de
#
#   Copyright (C) 1998, Jim Trocki
#                 2001, Sandro Poppi
#
#   This program is free software; you can redistribute it and/or modify
#   it under the terms of the GNU General Public License as published by
#   the Free Software Foundation; either version 2 of the License, or
#   (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   You should have received a copy of the GNU General Public License
#   along with this program; if not, write to the Free Software
#   Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
use Getopt::Std;
getopts ("s:g:h:t:l:u");
```

```
# the first line is summary information, adequate to send to a pager
# or email subject line
#
#
# the following lines normally contain more detailed information,
# but this is monitor-dependent
#
# see the "Alert Programs" section in mon(1) for an explanation
# of the options that are passed to the monitor script.
#
$summary=<STDIN>;
chomp $summary;

$t = localtime($opt_t);
($wday,$mon,$day,$tm) = split (/\s+/, $t);

print <<EOF;

Alert for group $opt_g, service $opt_s
EOF

print "This alert was sent because service was restored\n"
   if ($opt_u);

print <<EOF;
This happened on $wday $mon $day $tm
Summary information: $summary
Arguments passed to this script: @ARGV
Detailed information follows:

EOF

# shutdown heartbeat
system ("/etc/init.d/heartbeat stop");
```

Anexos B

[root@localhost /root]# **debugfs**

debugfs 1.18, 11-Nov-2004 for EXT2 FS 0.5b, 04/08/09

debugfs: open -f -w /dev/sdb1

Agora, o uso "features" para ver que bocados da característica são ajustados no sistema de ficheiros:

debugfs: features

Filesystem features: has_journal filetype sparse_super

Em seguida cancela-se os bocados do jornal:

debugfs: features -has_journal -needs_recovery

Filesystem features: filetype sparse_super

debugfs: quit

[root@ localhost /root]# debugfs

Deste modo terá o ext2 habilitado

Anexo C

Drbd.conf

Seguindo a configuração deste o ficheiro ficaria assim:

```
resource drbd0 {                    #primeiro espelhamento
  protocol=B                        #protocolo usado
  fsck-cmd=fsck.ext2 -p -y          # comando para fsck caso necessário
 disk {                             # Dados relativos ao disco
   do-panic                         # Em caso de erro, um kernel panic vai obrigar o
servidor a parar e a outra assumir
   disk-size=4096543                # É uma boa ideia indicar o tamanho do disco,
principalmente quando os discos não são exactamente iguais
 }
  on nó1 {                          # Primeiro nodo
    device=/dev/nb0                 # Dispositivo de drbd
    disk=/dev/hda6                  # Dispositivo de bloco
    address=10.10.10.1              # Endereço IP
    port=7789
  }
  on nó2   {
                    device=/dev/nb0
                    disk=/dev/hda6
                    address=10.10.10.3
                    port=7789
                  }
                }
```

Feito isto nos dois nós, que devem ter o *drbd.setup* idêntico, inserir o módulo do *drbd* e inicie o espalhamento nos dois nó:

```
# modprobe drbd
# /etc/rc.d/drbd start
```

Se estiver usando uma das últimas versões do *drbd*, o comando *'cat /proc/drbd'*, permiti verificar o progresso da sincronização. Se os nós estiverem em estado de *SyncAll*, é sinal que estão sincronizados. Quando a mensagem mudar para *Connect*, a sincronização está feita.

Neste ponto, a partição /dev/nb0 pode ser usada como qualquer outro dispositivo. Mesmo antes da sincronização terminada. Caso seja uma partição nova, crie a partição com o comando *mkfs* (tamanho de bloco igual a 4096 é recomendado):

```
# mkfs -j -b 4096 /dev/nb0          - para ext3
# mkreiserfs /dev/nb0               - para reiserfs
```

Se já existirem dados na partição ext2, passar para ext3 é simples:

```
# tune2fs -j /dev/nb0
```

Se for transformar em reiserfs, não tem jeito. Backup, e criar a partição novamente, depois passar os dados para a partição nova. E é só montar onde preferir:

```
# mount /dev/nb0 /mnt/mountpoint
```

Se ao carregar o módulo receber mensagens do tipo:

```
drbd.o: unresolved symbol sock_alloc
drbd.o: unresolved symbol proc_register
drbd.o: unresolved symbol schedule_timeout
```

É porque o *kernel* foi construído com CONFIG_MODVERSIONS e o DRBD foi construído sem MODVERSIONS, ou vice versa. Para resolver isso, existem duas maneiras:
Usando o sistema sem MODVERSIONS: Mude sua configuração do *kernel* e desabilitar a opção CONFIG_MODVERSIONS. Recompile o *kernel*.
Usando o sistema com MODVERSIONS: Edite o ficheiro *drbd/Makefile.vars* e adicione -DMODVERSIONS -DCONFIG_MODVERSIONS na opção KERNFLAGS. Reconstrua o *drbd*.

Atenção: Deve-se montar a partição apenas em um servidor, não faça isso nas duas.

Isto pode e irá corromper o sistema de ficheiros.
Acrescente no /etc/fstab onde quer que a partição espelhada seja activada.
"/dev/nb0 /ponto_de_montagem ext2 noauto 0 0"

Neste ponto, agora tem um espelhamento dos dados usando *drbd*.

Anexo D

Ficheiro /etc/ha.d/ha.cf

```
#!/bin/bash
# Ficheiro para mensagens de depuração (debug)
# debugfile /var/log/ha-debug
#
# Ficheiro para armazenar mensagens gerais do cluster
# logfile       /var/log/ha-log
# Enviar mensagens de log e mensagens gerais para o syslog
# (pode ser utilizado em conjunto com as opções acima).
logfacility     local0
# Os tempos podem ser especificados em segundos ou milissegundos.
# A unidade padrão é segundos.
# Exemplos:
#       10          significa dez segundos
#       1500ms      significa 1,5 segundos
# keepalive: tempo entre heartbeat (sinais de vida)
keepalive 3
# deadtime: tempo para declarar o nodo como inoperante
deadtime 30
# warntime: quanto tempo esperar antes de emitir o aviso de "late
#       heartbeat" (heartbeat atrasado). Verifique na FAQ como
#       utilizar este parâmetro para melhor ajustar o deadtime.
warntime 10
# initdead: deadtime na inicialização do heartbeat.
#
# Em alguns casos a rede leva algum tempo para tornar-se operante
# após uma reinicialização do sistema. Para estes casos, existe um
# parâmetro (initdead) para controlar o deadtime na inicialização
# do heartbeat. Sugere-se que ele seja no mínimo 2*deadtime.
# initdead 120
```

```
# nice_failback: determina o comportamento do cluster com relação
#               aos recursos.
#
# O comportamento padrão, equivalente a "nice_failback off", é que
# enquanto o servidor declarado como primário para um grupo de recursos
# estiver operante, os recursos estão com ele. Caso ele retorne de uma
# falha, os serviços retornam para ele (failback).
#
# Com "nice_failback on", caso o servidor secundário esteja executando
# os serviços e o servidor primário (segundo a configuração) retorne,
# os serviços continuam sendo prestados pelo secundário até que este
# torne-se inoperante.
# nice_failback on
# Comunicação
#
# udpport: porta udp utilizada para comunicação (694 é o padrão)
udpport 694
# Interfaces para o envio de heartbeats
#
# bcast: envia heartbeats por broadcast (comportamento padrão das
#       versões anteriores).
bcast eth0
# ucast: envia heartbeats para um endereço IP específico
#       Uso: ucast interface IP-do-outro-nodo
# ucast eth0 192.168.1.2
# mcast: envia heartbeats para um grupo de multicast
# Uso: mcast [dev] [mcast group] [port] [ttl] [loop]
#
# [dev]         dispositivo para envio e recepção de
#               heartbeats
# [mcast group]   grupo de multicast a usar (end. multicast
#               classe D: 224.0.0.0 - 239.255.255.255)
```

```
# [port]        porta udp para envio e recepção de heartbeats.
#               Pode ser a mesma usada nos exemplos de bcast
#               e ucast.
# [ttl]         propagação dos pacotes (heartbeats) enviados.
#               (0-255)
# [loop]        recebe (1) ou não (0) os próprios heartbeats.
# serial: porta serial utilizada para envio de heartbeats
#
# Consulte a documentação do pacote para maiores informações
# sobre a confecção de cabos e outros detalhes.
# Uso: serial porta
#serial /dev/ttyS0
# baud: velocidade (baud rate) da porta serial...
# baud   19200
# node: lista os nodos do cluster
#      (contém o nome retornado por uname -n)
node    cluster1
node    cluster2
```

Anexo E

Ficheiro /etc/ha.d/ha.conf

```
#!/bin/bash
debugfile /var/log/ha-debug
logfile  /var/log/ha-log          #Muito úteis para debug de configuração
serial /dev/ttyS0                 #Coloque a interface serial apropriada, se for utilizar a serial
keepalive 2                       #Indicar o tempo entre os heartbeats
deadtime 10                       #O nó será declarado como indisponível depois de 10 segundos
baud 19200                        #Velocidade da serial (bps)
udpport 694                       #Porta a ser usada (lembre-se de habilitar caso esteja usando firewall)
udp eth0                          #Interface a ser usada pelo heartbeat (broadcast)
mcast eth0 225.0.0.1 694 1 1 #Caso prefira usar multicast, coloque as configurações
nice_failback on                  #Opcional. Em caso de falha, quando o principal retornar a rede ele não tentará
retornar os recursos.
node linuxha1.linux-ha.org        #Obrigatório, hostname do servidor conforme descrito em `uname -a`
node linuxha2.linux-ha.org        S#Obrigatório, hostname do servidor conforme descrito em `uname -a`
```

Anexo F

Ficheiro mon.cf

```
#!/bin/bash
cfbasedir   = /etc/ha.d/mon/etc                    #Directório de configuração
alertdir    = /usr/lib/mon/alert.d                 #Directório dos alertas
mondir      = /usr/lib/mon/mon.d                   #Directório dos monitores
maxprocs    = 20                                   #Numero máximo de processos filhos
histlength = 100                                   #Tamanho do histórico
historicfile = /var/log/mon.log                    #Ficheiro de log
randstart = 60s
hostgroup servers 1.1.1.254 1.1.1.253 1.1.1.252    #Aqui criamos um grupo de hosts chamado servers
com os endereços IPs dos Servidores
watch servers                                      #Aqui começa a configuração para monitorar o grupo servers
service ping                                       #Nomeia o serviço, pois podem ser especificados mais de um para o mesmo grupo
    interval 1m                                    #Tempo entre as consultas
    monitor fping.monitor -a                       #Monitor a ser utilizado
    period wd {Mon-Frid}                           #Período
      alert mail.alert root@localhost              #Alertas a serem disparados
      alert heartbeat.alert
      alertevery 1h
    period wd {Sat-Sun}                            #Período
      alert heartbeat.alert                        #Alertas a serem disparados
      alert mail.alert root@localhost
```