1 INTRODUÇÃO

1.1 Considerações Iniciais

Sistemas de Alta Disponibilidade provêem o aumento da disponibilidade de serviços através de técnicas computacionais, tanto em recurso de software como de hardware. O uso de tais sistemas tem se expandido nos últimos anos principalmente devido ao contínuo barateamento e disponibilidade de hardware para computadores, bem como de meios físicos de comunicação. As vantagens de Sistemas de Alta Disponibilidade incluem a possibilidade de implementação de tolerância a falhas através da replicação de processos em unidades de computação distintas. O uso do Linux na formação de clusters torna esse sistema operacional ainda mais robusto.

1.2 Justificativa

As utilizações de clusters vêm crescendo rapidamente como alternativa de escolha para construção de sistemas de alto desempenho para computação paralela. O rápido avanço no desenvolvimento de microprocessadores e de redes de altíssimas velocidades com componentes de fácil disponibilidade no mercado favorece a construção deste ambiente (PITANGA, 2004).

A tecnologia de cluster de computadores pessoais passou a ser adotada por diversas universidades e empresas com o objetivo de conseguirem processar grandes volumes de dados para atingir sua demanda computacional. Provedores de internet ou sites de comércio eletrônico frequentemente requerem alta disponibilidade e balanceamento de carga de forma escalável, para se obter um sistema que forneça determinados serviços, sendo que estes devem estar sempre (ou quase sempre) disponíveis para receber solicitações.

A alta disponibilidade está ligada diretamente a crescente dependência aos computadores. Com o avanço das atividades que contam com recursos computacionais, é cada vez maior o transtorno causado pela eventual falha destes sistemas. Desde o sistema bancário até supermercados, os computadores têm papel crítico, sendo utilizados não apenas em tais tipos de serviços, mas principalmente em empresa cuja maior funcionalidade é exatamente a oferta de algum serviço computacional, como e-business, notícias, sites web, etc.

1.3 Objetivo

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é o estudo bibliográfico e a implementação da tecnologia de Servidores de Alta Disponibilidade, inserindo conceitos de clusterização, suas aplicações e a apresentação de uma rede com servidores Linux de Alta Disponibilidade, a fim de prover as informações necessárias à construção de um multicomputador de baixo custo e alto desempenho para utilização em ambientes acadêmicos e comerciais.

1.3.2 Objetivos Específicos

- Estudo sobre a tecnologia de cluster.
- Estudo sobre Alta Disponibilidade.
- Implementação de Servidores Linux de Alta Disponibilidade.

1.4 Metodologia

A metodologia seguida para o desenvolvimento deste Trabalho de Conclusão de Curso, consistiu em pesquisas bibliográficas sobre Sistemas Distribuídos, Clusters e Sistema de Alta Disponibilidade. Por fim foi desenvolvido um estudo de caso da implementação de um sistema de alta disponibilidade, sendo a solução baseada em servidores Linux, mostrando a instalação, configuração e a realização de testes aplicados em situações de operação normal, situações de falha e situações de recuperação do sistema.

Foi utilizada no estudo de caso a combinação de:

Sistema operacional Linux;

DRBD (Distributed Replicated Block Device): para o espelhamento dos dados em blocos;

Heartbeat: software responsável pela reconfiguração automatizada na ocorrência de falhas no sistema;

MON: software de monitoramento das condições da rede.

1.5 Descrição do Trabalho

Este trabalho está dividido em sete capítulos. Inicialmente, o Capítulo 1 dá uma visão geral do trabalho a ser desenvolvido, posteriormente no Capítulo 2, é mostrado um breve histórico sobre a evolução dos clusters de computador, as classificações das arquiteturas computacionais e em seguida conceitos sobre Sistema Distribuídos.

No Capítulo 3 descreveremos em detalhes o assunto Alta Disponibilidade, onde será visto que a área de tolerância à falhas contém muitas características. Ao final deste capítulo, poderemos entender as diferenças que existem em ambientes de Alta Disponibilidade.

Na seqüência, no Capítulo 4 abordaremos os tipos e os conceitos de clusters existentes, que são fundamentais na implementação de um Cluster de Alta Disponibilidade.

No Capítulo 5 mostraremos a implementação de uma solução utilizando o sistema operacional Linux para a criação de um ambiente de Alta Disponibilidade.

No capítulo 6 mostraremos a configuração do sistema utilizado e os testes realizados, junto com os resultados obtidos. E finalmente, no Capítulo 7, são relatadas as conclusões gerais do trabalho.

2 INTRODUÇÃO A COMPUTAÇÃO PARALELA E DISTRIBUÍDA

2.1 Considerações Iniciais

A constante demanda de poder computacional vem gerando a necessidade de processadores cada vez mais rápidos. Na computação de alto desempenho, utilizada para programação científica, multimídia, gerenciamento de grandes volumes de dados, entre outros, a solução passa por máquinas com múltiplos processadores ou ainda clusters proprietários fornecidos por grandes empresas. Ambas as soluções são custosas e de pouca escalabilidade.

A estrutura de agrupamentos de computadores, chamada de clusters, apresenta vantagens competitivas em relação aos ambientes multiprocessados de memória compartilhada (computadores com diversos processadores em uma placa mãe), permitindo que o acréscimo de computadores torne o sistema mais rápido, possuindo também componentes de fácil disponibilidade e manutenção.

A utilização de clusters da classe *Beowulf* vêm crescendo rapidamente como alternativa de escolha para construção de sistemas de alto desempenho para *computação paralela*. O rápido avanço no desenvolvimento de microprocessadores e de redes de altíssimas velocidades com componentes de fácil disponibilidade no mercado favorecem a construção deste ambiente.

A agregação de recursos computacionais, em configurações de clusters tem sido um tópico importante de estudo em inúmeras universidades, centros de pesquisa e empresas de Tecnologia da Informação. A grande meta da agregação de recursos computacionais, por intermédio dos clusters, é prover respostas para as limitações encontradas nas arquiteturas computacionais centralizadas. Desta forma, o desenvolvimento da computação distribuída em larga escala ganha uma especial importância. O esforço da computação distribuída em larga escala é, usualmente, denominada de computação de alto desempenho.

2.2 Cluster: Breve Histórico e Evolução

No início dos anos 70, quando pela primeira vez na história os computadores começaram a efetuar sua interligação por intermédio das redes, já surgia a idéia de agregar os recursos computacionais não utilizados. Um par de programas, conhecido como *creeper e reaper-ran*, são utilizados na ARPAnet em um conjunto de experimentos voltados para a computação distribuída.

A idéia inicial que conduz ao cluster foi desenvolvida na década de 60 pela IBM como uma forma de interligar grandes mainframes, visando obter uma solução comercialmente viável de paralelismo. Nesta época, o sistema HASP (Houston Automatic Spooling Priority) da IBM e seu sucessor, o JES (Job Entry System), proviam uma maneira de distribuir tarefas nos mainframes interligados. A IBM ainda hoje suporta o cluster de mainframes através do Parallel Sysplex System, que permite ao hardware, ao sistema operacional, ao middleware e ao software de gerenciamento do sistema prover uma melhora dramática na performance e custo ao permitir que usuários de grandes mainframes continuem utilizando suas aplicações existentes.

Entretanto, o cluster foi ganhando força até que três tendências convergiram nos anos 80: microprocessadores de alta performance, redes de alta velocidade e ferramentas padronizadas para computação distribuída de alto desempenho. Uma quarta tendência possível é a crescente necessidade de poder de processamento para aplicações científicas e comerciais unida ao alto custo e a baixa acessibilidade dos tradicionais supercomputadores.

No final de 1993, Donald Becker e Thomas Sterling, pesquisadores do Centro de Excelência em Dados do Espaço e Ciências da Informação (CESDIS), iniciaram um esboço de um sistema de processamento distribuído construído a partir de hardware convencional como uma medida de combate aos custos dos supercomputadores. No início de 1994, trabalhando no CESDIS, com o patrocínio do projeto HTPCC/ESS, criaram o primeiro cluster da classe Beowulf para a NASA, quando a agência necessitava de um equipamento que processasse na ordem de um gigaflop.

O protótipo inicial era um cluster de 16 processadores DX4 ligados por dois canais Ethernet acoplados (Ethernet bonding). A máquina foi um sucesso instantâneo e esta idéia rapidamente se espalhou pelos meios acadêmicos, pela NASA e por outras comunidades de pesquisa.

2.3 Classificação das Arquiteturas Computacionais

Devido à existência de uma grande diversidade de arquitetura de computadores, inúmeras taxonomias já foram propostas, tentando uniformizar de maneira mais coerente às características dos diferentes sistemas computacionais. A classificação dos ambientes de hardware mais aceita na área de arquitetura de computadores é a conhecida taxonomia de Flynn (FLYNN, 1972). Esta prestigiosa classificação proposta há mais de trinta anos (e válida ainda hoje) leva em consideração o número de instruções executadas em paralelo versus o conjunto de dados para os quais as instruções executadas são submetidas. Desta forma, a

taxonomia de Flynn estabeleceu as seguintes classificações de computadores descritas a seguir.

2.3.1 Arquitetura SISD (Single Instruction Single Data)

É a identificação mais simples de arquitetura, onde o equipamento é considerado seqüencial, pois só é executada uma instrução por vez para cada dado enviado. Incluem-se nesta arquitetura as máquinas Von Neumann tradicionais sem qualquer paralelismo (PITANGA, 2004).

Este modelo de arquitetura pode ser visualizado na Figura 2.1, onde a sigla MM representa o "Módulo de Memória" e a sigla EP representa o "Elemento de Processamento", ou seja, o processador.

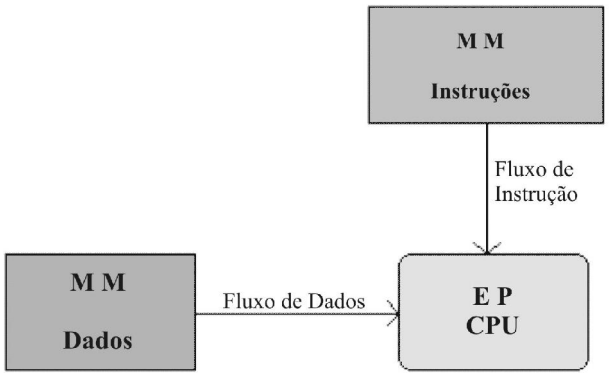

Figura 2.1: Arquitetura SISD (Single Instruction Single Data).
FONTE: DANTAS (2005).

2.3.2 Arquitetura MISD (Multiple Instruction Single Data)

São máquinas que executam várias instruções ao mesmo tempo sobre um único dado. Torna-se até difícil imaginar uma máquina nestas condições. Este tipo de classificação não possui representante e até mesmo Flynn duvidou que algum dia isso possa existir.

Mas poderia ser criada, por essa teoria, uma máquina com vários processadores tentando quebrar um código de criptografia, mas com a arquitetura MIMD este problema pode ser resolvido, sem precisar construir uma máquina para este uso específico (PITANGA, 2004).

A Figura 2.2 apresenta o modelo desta arquitetura, onde a sigla MM também representa o "Módulo de Memória" e a sigla EP também representa o "Elemento de Processamento", ou seja, o processador.

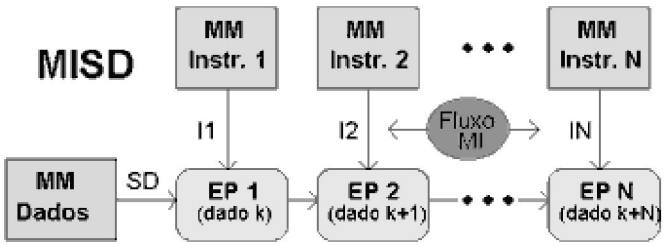

Figura 2.2: Arquitetura MISD (Multiple Instruction Single Data).
FONTE: DANTAS (2005).

2.3.3 Arquitetura SIMD (Single Instruction Multiple Data)

Esta arquitetura é o equivalente ao paralelismo de dados, onde uma simples instrução é executada paralelamente utilizando vários dados de forma síncrona, em que se executa um único programa ao mesmo tempo. Pode-se observar que, neste contexto, também se encaixa, pela característica, a tecnologia MMX (MultiMedia eXtension) existente dentro dos processadores Intel e nos computadores vetoriais como o Cray 1 e o CM-2. Nas máquinas SIMD tem-se menos hardware, menos memória e um hardware muito mais específico e o seu modelo pode ser observado na Figura 2.3 (PITANGA, 2004).

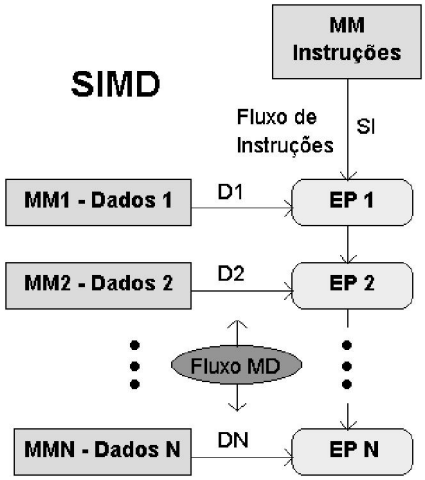

Figura 2.3.: Arquitetura SIMD (Single Instruction Multiple Data).
FONTE: DANTAS (2005).

2.3.4 Arquitetura MIMD (Multiple Instruction Multiple Data)

Esta Arquitetura refere-se ao modelo de execução paralela na qual cada processador está essencialmente agindo independentemente havendo, portanto, múltiplos fluxos de instruções e múltiplos dados como se fossem um conjunto de máquinas SISD, onde cada processador é capaz de executar um programa diferente. São exemplos nessa categoria servidores com múltiplos processadores, sistemas MPP e os clusters de computadores.

Embora a classificação de Flynn seja conveniente num primeiro momento, é bastante imprecisa para classificar as variedades de multiprocessadores existentes. A categoria MIMD engloba uma grande diversidade de máquinas, havendo várias tentativas de as classificar. Uma das formas mais usadas é a classificação que usa como critério a forma como a memória central está fisicamente organizada e o tipo de acesso que cada processador tem à totalidade da memória, classificando, assim, esta arquitetura em computadores MIMD de memória compartilhada e memória distribuída (PITANGA, 2004).

Apresentam-se a seguir, as duas classes deste tipo de arquitetura.

2.3.4.1 Arquitetura MIMD de Memória Compartilhada

Nesta classe incluem-se todas as máquinas com múltiplos processadores que compartilham um espaço de endereços de memória comum (máquinas multiprocessadas), um processador não fica parado se há trabalho a ser feito. Além de ser pouco escalável e de difícil manutenção, este tipo de arquitetura conduz a situações em que um processador, ao acessar a memória comum, pode entrar em conflito com outros processadores. A maior ou menor gravidade destes conflitos dependem, no que diz respeito ao hardware, da sofisticação do dispositivo usado para interligar os processadores e as unidades de memória.

Também o uso de "cache" associados a cada processador permite diminuir o tráfego no barramento de memória. Este esquema exige rotinas que assegurem a consistência de todas as cópias de blocos de memória principal que estejam simultaneamente em mais de um cache de diferentes processadores. É necessário também que o sistema operacional que esteja rodando nessas máquinas suportem este tipo de arquitetura e implemente mecanismos mais sofisticados de escalonamento de processos entre os processadores (PITANGA, 2004).

A arquitetura MIMD de Memória Compartilhada é esboçada graficamente através da Figura 2.4.

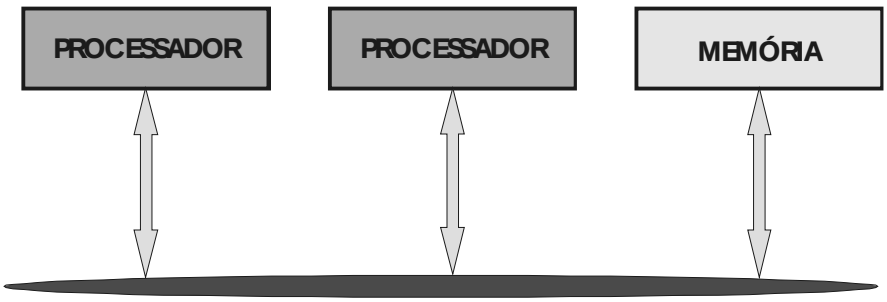

Figura 2.4: Arquitetura genérica MIMD de memória compartilhada.
FONTE: PITANGA (2004).

A seguir apresentaremos algumas das arquiteturas que utilizam memória compartilhada, com o objetivo de um melhor detalhamento de diferentes configurações que são encontradas em ambientes computacioniais.

– **Multiprocessadores Simétricos (SMP):** Os ambientes denominados como multiprocessadores simétricos (SMP) são conhecidos como arquiteturas de compartilhamento total. Estas configurações são caracterizadas por até dezenas de processadores compartilhando todos os recursos computacionais disponíveis e executando um único sistema operacional.

Os processadores são considerados simétricos, uma vez que tem os mesmos custos para acesso à memória. Todos, por exemplo, possuem acesso igual à memória e a qualquer dispositivo conectado no sistema de entrada e saída.

Um exemplo clássico de uma configuração SMP é ilustrado pela Figura 2.5, onde é possível compreender melhor o conceito de uma máquina SMP. Pode-se observar que a configuração é caracterizada por vários processadores compartilhando uma única memória e um único sistema de entrada e saída. Um fator particular da configuração é não possuir múltiplas memórias e nem múltiplos sistemas de entrada e saída, mas apenas múltiplos processadores.

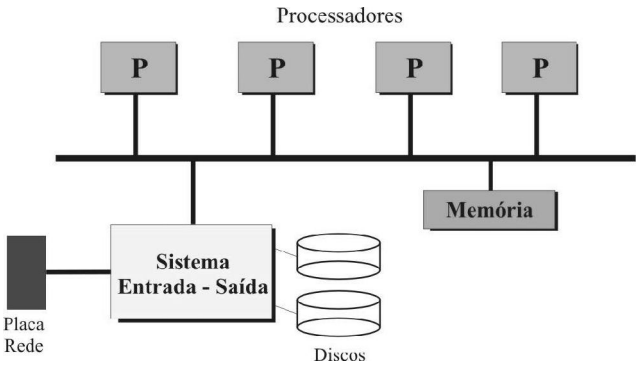

Figura 2.5: Configuração clássica de uma arquitetura SMP.

FONTE: DANTAS (2005).

As máquinas SMP são também conhecidas como multiprocessadores fortemente acoplados. Por esta razão, estes ambientes não são muito escaláveis e nem possuem, muitas vezes, uma memória local (ou cache) por processador e com o aumento no número de processadores, a taxa de colisão de acesso à memória cresce de maneira substancial.

– **Memória de Acesso não-Uniforme com Coerência de Cache (CC-NUMA):** O CC-NUMA é um sistema de multiprocessadores escaláveis que possuem uma arquitetura de coerência de cache com acesso não uniforme. Assim, como uma máquina SMP, cada processador é um sistema CC-NUMA com uma visão globalizada da memória. Nessas

arquiteturas, a memória é dividida em tantos blocos quantos forem os processadores do sistema e cada bloco de memória é conectado via barramento a um processador como memória local. O acesso aos dados que estão na memória local é muito mais rápido que o acesso aos dados em blocos de memória remotos. Apesar do uso de memória compartilhada, as arquiteturas NUMA mais modernas oferecem bibliotecas para programação utilizando troca de mensagens. Cada nó processador possui uma cache local para reduzir o tráfego na rede de interconexão. O balanceamento de carga é realizado dinamicamente pelos protocolos de coerência das caches. No caso de arquiteturas com paralelismo no nível de threads, apenas sistemas com memória compartilhada foram construídos ou propostos (PITANGA, 2004).

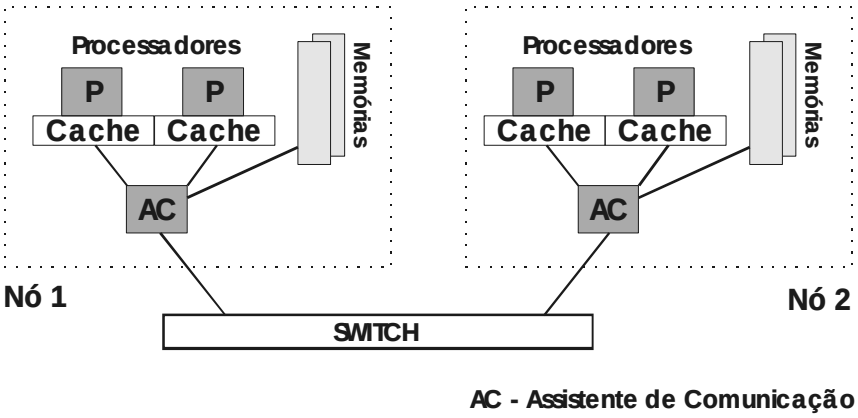

Figura 2.6: Configuração convencional de uma CCNUMA.
FONTE: PITANGA (2004).

2.3.4.2 Arquitetura MIMD com Memória Distribuída

Nesta incluem-se as máquinas formadas por várias unidades processadoras, cada uma com sua própria memória (chama-se nó uma unidade constituída por um processador, memória local e dispositivos de entrada/saída), sendo geralmente chamados de multicomputadores. Neste caso não existe qualquer tipo de memória comum e a comunicação entre os diferentes nós do sistema é feita através de dispositivos físicos de entrada/saída. Um exemplo deste tipo de arquitetura é ilustrado na Figura 2.7.

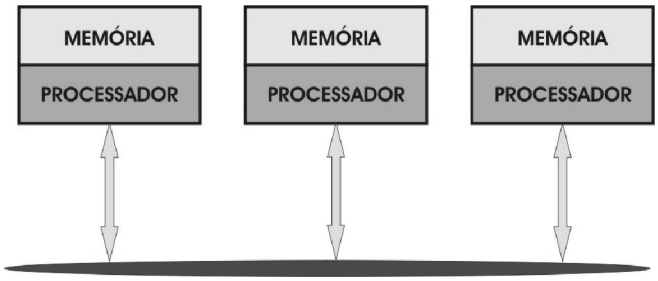

Figura 2.7: Arquitetura com Memória Distribuída.
FONTE: PITANGA (2004).

A seguir comentamos um tipo de arquitetura que utiliza memória distribuída, que é encontrada em ambientes computacioniais.

– **Processadores Paralelos Massivos (MPP):** Um sistema MPP é um grande sistema de processamento paralelo com arquitetura de memória compartilhada (DSM) e centralizada. Ele é composto por algumas centenas de elementos processadores (nodes), interconectados por uma rede/switch de alta velocidade. Cada nó pode possuir uma memória principal e um ou mais processadores e alguns nós podem possuir periféricos como discos ou um sistema de backup dedicado (DANTAS, 2005).

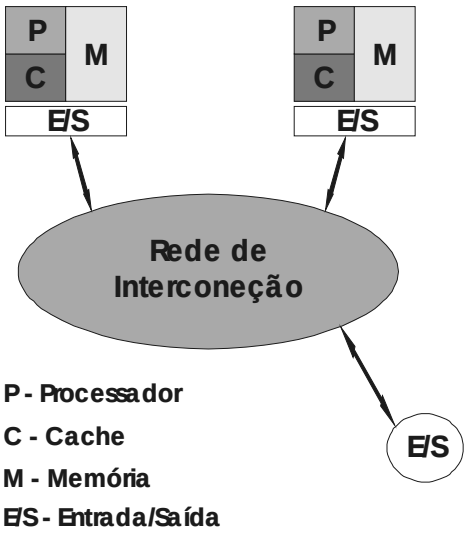

Figura 2.8: Configuração genérica de um MPP.
FONTE: DANTAS (2005).

2.4 Sistemas Distribuídos

Um sistema distribuído é um conjunto de elementos que se comunicam através de uma rede de interconexão e que utilizam software de sistema distribuído. Cada elemento é composto por um ou mais processadores e uma ou mais memórias. O uso de tais sistemas tem se expandido nos últimos anos principalmente devido ao contínuo barateamento e disponibilidade de hardware para computadores, bem como a disseminação das redes de computadores. As vantagens dos sistemas distribuídos incluem a possibilidade de seu crescimento, ou seja, grande escalabilidade, a possibilidade de implementação de aplicação distribuída e a possibilidade de implementação de sistemas tolerante a falhas por meio de replicação de serviços ou de máquinas distintas (PITANGA, 2004).

Sob o aspecto de arquitetura de máquinas para a execução de aplicativos, os sistemas distribuídos devem ser vistos como configurações com grande poder de escala pela agregação dos computadores existentes nas redes convencionais. Nos ambientes distribuídos, a homogeneidade ou heterogeneidade de um conjunto de máquinas, onde cada qual possui sua arquitetura de software-hardware executando sua própria cópia de sistema operacional, permite a formação de interessantes configurações de SMPs, de MPPs, de clusters e grids computacionais (DANTAS, 2005).

Sistemas distribuídos não são sinônimos de redes de computadores. Uma rede de computadores pode fornecer a infra-estrutura computacional para um sistema distribuído, mas nem toda aplicação de uma rede é necessariamente distribuída (PEIXOTO, 2002).

Um sistema distribuído é o resultado da integração de sistemas computacionais autônomos combinados de forma a atingir um objetivo comum. Baseado neste conceito pode-se citar algumas características importantes, como:

- Compartilhamento de recursos: hardware, software e dados;
- Sistemas abertos: independência;
- Extensões de hardware: periféricos adicionais, interfaces de comunicação ou memória;
- Extensões de software: adição de características aos Sistemas Operacionais e serviços de compartilhamento de recursos;
- Concorrência de recursos;
- Acesso concorrente e utilização de recursos compartilhados sincronizados.

Embora a utilização de ambientes distribuídos seja interessante sob o aspecto de utilização de recursos abundantes e, na maioria das vezes, ociosos nas redes, alguns cuidados devem ser verificados nas fases de projeto e implementação de aplicativos candidatos ao processamento nestas configurações, tais como:

- Segurança: onde um aplicativo executando de forma remota em máquinas desconhecidas poderá estar exposto a processos ou indivíduos não autorizados;
- Retardos de comunicação: nas configurações distribuídas esses retardos são usualmente grandes, posto que componentes do ambiente podem estar geograficamente distribuídos. Desta forma, somente aplicativos com uma granulidade grossa devem ser candidatos à execução distribuída;
- Confiabilidade e disponibilidade: em configurações distribuídas esses aspectos são complexos, devido principalmente aos inúmeros conjuntos de

componentes do ambiente. De uma outra forma, deve-se entender que em sistemas com mais computadores, compostos ainda por vários pacotes de software, a probabilidade de uma falha ocorrer aumenta consideravelmente;

- Compatibilidade: juntamente com os demais já apresentados, é um problema constante em grandes sistemas distribuídos, uma vez que diferentes sistemas operacionais, arquitetura de processadores, compiladores e interfaces de redes podem não interagir conforme solicitado.

2.5 Considerações Finais

Tratamos neste capítulo alguns fundamentos das arquiteturas computacionais, visando mostrar os caminhos para os próximos capítulos sobre cluster e alta disponibilidade computacional. Sendo assim, foram introduzidos os conceitos de arquiteturas de memória distribuída e compartilhada, SMP, MPP e CCNUMA. Abordamos ainda conceitos sobre Sistemas Distribuídos para que se possa compreender como funciona a idéia de compartilhamento de recursos em rede.

3 ALTA DISPONIBILIDADE

3.1 Considerações Iniciais

Atualmente, os computadores crescem em todos os ambientes empresariais, comerciais e até mesmo domésticos e nenhum usuário quer que seu equipamento pare de funcionar. A alta disponibilidade vem a tópico para resolver este tipo de situação, garantindo a continuidade de operação do sistema em serviços de rede, armazenamento de dados ou processamento, mesmo se houver falhas em um ou mais dispositivos, sejam eles hardware ou software (PITANGA, 2003).

Com isso, a alta disponibilidade tem como função essencial deixar um sistema no ar vinte e quatro horas por dia, sete dias por semana ou que não suportem paradas de meia hora ou alguns minutos. São estas paradas não planejadas que influenciam diretamente a qualidade no serviço e os prejuízos financeiros que as mesmas possam acarretar (FILHO, 2002).

A disponibilidade dos sistemas torna-se então uma questão vital de sobrevivência empresarial, ou seja, se o sistema parar, a empresa também pára. Um sistema de comércio eletrônico, como venda de livros, por exemplo, não admite indisponibilidade. De maneira geral, um servidor de boa qualidade apresenta uma disponibilidade de 99,5%, enquanto que uma solução através de cluster de computadores apresenta uma disponibilidade de 99,99% (PITANGA, 2003).

3.2 Definição

O conceito de Alta Disponibilidade (HA - High Availability) produz a ilusão de serviços ininterruptos. Para que se entenda a Alta Disponibilidade, faz-se necessário inicialmente, perceber que a Alta Disponibilidade não é apenas um produto ou uma aplicação que se instale e sim uma característica de um sistema computacional. Existem mecanismos, técnicas e blocos básicos que podem ser utilizados para aumentar a disponibilidade de um sistema. A simples utilização destes blocos, entretanto, não garante este aumento se não for acompanhado de um completo estudo e projeto de configuração.

3.3 Classes de Disponibilidade

3.3.1 Disponibilidade Básica

A Disponibilidade Básica é aquela encontrada em máquinas comuns, sem nenhum mecanismo especial, em software ou hardware, que vise de alguma forma mascarar as eventuais falhas destas máquinas. Costuma-se dizer que máquinas nesta classe apresentam uma disponibilidade de 99% a 99,9%. Isto equivale a dizer que em um ano de operação a máquina pode ficar indisponível por um período de 9 horas a quatro dias. Estes dados são empíricos e os tempos não levam em consideração a possibilidade de paradas planejadas (que serão abordadas mais adiante), porém são aceitas como o senso comum na literatura da área (MORIMOTO, 2002).

3.3.2 Alta Disponibilidade

Adicionando-se mecanismos especializados de detecção, recuperação e mascaramento de falhas, pode-se aumentar a disponibilidade do sistema, de forma que este venha a se enquadrar na classe de Alta Disponibilidade. Nesta classe, as máquinas 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. Nesta classe se insere grande parte das aplicações comerciais de Alta Disponibilidade, como as centrais telefônicas (MORIMOTO, 2002).

Por sua definição, percebe-se que, o principal objetivo da Alta Disponibilidade é buscar uma forma de manter os serviços prestados por um sistema a outros elementos, mesmo que o sistema em si venha a se modificar internamente por causa de uma falha. Nesta definição está implícito o conceito de mascaramento de falhas, através de redundância ou replicação (termos que serão conceituados mais tarde). Um determinado serviço, que se quer altamente disponível, é colocado por trás de uma camada de abstração, que permita mudanças em seus mecanismos internos mantendo intacta a interação com elementos externos.

Este é o principal fundamento da Alta Disponibilidade tornando-a, dessa forma, uma sub-área da Tolerância a Falhas, onde esta visa manter a disponibilidade dos serviços prestados por um sistema computacional, através da redundância de hardware e reconfiguração de software. A Alta Disponibilidade pode ser representada por vários computadores juntos agindo como um só, cada um monitorando os outros e assumindo seus serviços, caso perceba que algum deles falhou.

Menos complexo de desenvolver que o hardware, o software de Alta Disponibilidade é quem se preocupa em monitorar outras máquinas de uma rede, saber que serviços estão sendo prestados, quem os está prestando e o que fazer quando uma falha é percebida.

3.3.3 Disponibilidade Contínua

Com a adição de noves se obtêm 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 paradas planejadas e não planejadas são mascaradas e o sistema está sempre disponível.

3.4 Cálculo da Disponibilidade

Em um sistema real, se um componente falha, ele é reparado ou substituído por um novo componente. Se este novo componente falha, ele é substituído por outro e assim por diante. O componente reparado é tido como no mesmo estado que um componente novo. Durante sua vida útil, um componente pode ser considerado como estando em um destes dois estados: *funcionando* ou *em reparo*. O estado *funcionando* indica que o componente está operacional e o estado *em reparo* significa que ele falhou e ainda não foi substituído por um novo componente.

Em caso de defeitos, o sistema vai do estado *funcionando* para o estado *em reparo* e, quando a substituição é feita, ele volta para o estado *funcionando*. Sendo assim, pode-se dizer que o sistema apresenta, ao longo de sua vida, um tempo médio até apresentar falha (Mean Time to Failure - MTTF) e um tempo médio de reparo (Mean Time to Repair - MTTR). Seu tempo de vida é uma sucessão de MTTFs e MTTRs, à medida que vai falhando e sendo reparado. O tempo de vida útil do sistema é a soma dos MTTFs nos ciclos MTTF+MTTR já vividos (JALOTE, 1994).

De forma simplificada, diz-se que a disponibilidade de um sistema é a relação entre o tempo de vida útil deste sistema e seu tempo total de vida. Isto pode ser representado pela fórmula abaixo:

$$\text{Disponibilidade} = \text{MTTF} / (\text{MTTF} + \text{MTTR})$$

Ao avaliar uma solução de Alta Disponibilidade, é importante levar em consideração se na medição do MTTF são observadas como falhas as possíveis paradas planejadas.

3.5 Aspectos Considerados na Alta Disponibilidade

Para se entender corretamente dos fundamentos que levam a utilizar uma solução de Alta Disponibilidade, deve-se conhecer os aspectos mais importantes envolvidos nessa solução. Não são muitos, porém estes termos são muitas vezes utilizados de forma errônea em literatura não especializada. Inicialmente, são mostradas as diferenças básicas entre falha, erro e defeito. Estas palavras, que parecem tão próximas mas, na verdade, designam a ocorrência de algo anormal em três universos diferentes de um sistema computacional.

3.5.1 Falha, erro e defeito

Uma falha acontece no universo físico, ou seja, no nível mais baixo do hardware. Uma flutuação da fonte de alimentação, por exemplo, é uma falha. Uma interferência eletromagnética também. Estes são dois eventos indesejados, que acontecem no universo físico e afetam o funcionamento de um computador ou de partes dele (FILHO, 2002).

A ocorrência de uma falha pode acarretar um erro, que é a representação da falha no universo informacional. Um computador trabalha com bits, cada um podendo conter 0 ou 1. Uma falha pode fazer com que um (ou mais de um) bit troque de valor inesperadamente, o que certamente afetará o funcionamento normal do computador. Uma falha, portanto, pode gerar um erro em alguma informação (RUIZ, 2000).

Já esta informação errônea, se não for percebida e tratada, poderá gerar o que se conhece por defeito. Quando ele acontece, o sistema simplesmente trava, mostra uma mensagem de erro ou, ainda, perde os dados do usuário sem maiores avisos. Isto é percebido no universo do usuário.

Portanto, de acordo com o exposto acima, uma falha no universo físico pode causar um erro no universo informacional, que por sua vez pode causar um defeito percebido no universo do usuário. A Tolerância a Falhas visa exatamente acabar com as falhas, ou tratá-las, enquanto ainda são erros. Já a Alta Disponibilidade permite que máquinas travem ou errem, contanto que exista outra máquina para assumir seu lugar.

Para que uma máquina assuma o lugar de outra, é necessário que se descubra, de alguma forma, que a outra falhou. Isso é feito através de testes periódicos, cujo período deve ser configurável, nos quais a máquina secundária testa não apenas se a outra está ativa, mas também se está fornecendo respostas adequadas a requisições de serviço. Um mecanismo de detecção equivocado pode causar instabilidade no sistema. Por serem periódicos, nota-se que existe um intervalo de tempo durante o qual o sistema pode estar indisponível sem que a outra máquina o perceba.

3.5.2 Failover

O processo no qual uma máquina assume os serviços de outra, quando esta última apresenta falha, é chamado *failover*. O *failover* pode ser do tipo automático ou manual, sendo o primeiro tipo o que normalmente se espera de uma solução de Alta Disponibilidade. Ainda assim, algumas aplicações não críticas podem suportar um tempo maior até a recuperação do serviço e, portanto, podem utilizar *failover* manual. Além do tempo entre a falha e a sua detecção, existe também o tempo entre a detecção e o reestabelecimento do serviço. Grandes bancos de dados, por exemplo, podem exigir um considerável período de tempo até que atualizem os índices de suas tabelas e, durante este tempo, o serviço ainda estará indisponível.

Para se executar o *failover* de um serviço, é necessário que as duas máquinas envolvidas possuam recursos equivalentes. Um recurso pode ser uma placa de rede, um disco rígido, os dados neste disco e todo e qualquer elemento necessário à prestação de um determinado serviço. É vital que uma solução de Alta Disponibilidade mantenha recursos redundantes com o mesmo estado, de forma que o serviço possa ser retomado sem perdas.

Dependendo da natureza do serviço, executar um *failover* significa interromper as transações em andamento, perdendo-as, sendo necessário reiniciá-las após o *failover*. Em outros casos, significa apenas um retardo até que o serviço esteja novamente disponível. Nota-se que o *failover* pode ou não ser um processo transparente, dependendo da aplicação envolvida.

3.5.3 Failback

Ao ser percebida a falha de um servidor, além do *failover*, é obviamente necessário que se faça manutenção no servidor falho. Ao ser recuperado de uma falha, este servidor será recolocado em serviço, e então se tem a opção de realizar o processo inverso do *failover*, que se chama *failback*. O *failback* é, portanto, o processo de retorno de um determinado serviço de uma outra máquina para sua máquina de origem. Também pode ser do tipo automático, manual ou até mesmo indesejado. Em alguns casos, em função da possível nova interrupção na prestação de serviços, o *failback* pode não ser atraente.

3.5.4 Missão

A missão de um sistema pode ser definida como o período de tempo no qual ele deve desempenhar suas funções sem interrupção. Supor, por exemplo, uma farmácia, que funcione no horário das 8h às 20h. Esta farmácia não pode ter seu sistema fora do ar durante este período de tempo. Se este sistema vier a apresentar defeitos fora deste período, ainda que

indesejados, estes defeitos não atrapalham em nada o andamento correto do sistema quando ele é necessário. Uma farmácia 24h, por outro lado, tem uma missão contínua, de forma que qualquer tipo de parada deve ser mascarada.

A Alta Disponibilidade visa eliminar as paradas não planejadas. Porém, no exemplo mostrado acima, da farmácia que funciona das 8h às 20h, as paradas planejadas não devem acontecer dentro do período de missão. Paradas não planejadas decorrem de defeitos, já paradas planejadas são aquelas que se devem a atualizações, manutenção preventiva e atividades correlatas. Desta forma, toda parada dentro do período de missão pode ser considerada uma falha no cálculo da disponibilidade.

Uma aplicação de Alta Disponibilidade pode ser projetada inclusive para suportar paradas planejadas, o que pode ser importante, por exemplo, para permitir a atualização de programas por problemas de segurança, sem que o serviço deixe de ser prestado.

3.6 Considerações Finais

Neste capítulo foram abordados os conceitos e os componentes que envolvem a Alta Disponibilidade.

A alta disponibilidade e tolerância à falhas consistem, basicamente, em ter hardware redundante que entra em funcionamento automaticamente após a detecção de falha do hardware principal. Independentemente da solução adotada, existe sempre um tempo médio de recuperação (MTTRe - Mean Time To Recover), que é o espaço de tempo (médio) que decorre entre a ocorrência da falha e a total recuperação do sistema ao seu estado operacional.

Finalizando, foram vistos os principais aspectos e conceitos sobre falha, erro, defeito, Failover (que é uma operação onde um computador assume a função de outro que apresentou falha), Failback (que é o retorno do computador que se recupera de uma falha retornando as atividades normais de operação) e missão (que é o tempo de operação sem interrupção).

4 CLUSTER COMPUTACIONAL

4.1 Considerações Iniciais

Na sua forma mais básica, um cluster é um sistema que compreende dois ou mais computadores ou sistemas (denominados nodos), no qual trabalham em conjunto para executar aplicações ou realizar outras tarefas, de tal forma que os usuários tenham a impressão que somente um único sistema responde para eles, criando assim uma ilusão de um recurso único (computador virtual). Este conceito é denominado transparência do sistema. Como características fundamentais para a construção destas plataformas incluem-se elevação da confiança, distribuição de carga e performance.

4.2 Tipos de Clusters

4.2.1 Alta Disponibilidade (High Availability (HA) 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 idéia geral é que se um nó do cluster vier a falhar (*failover*), aplicações ou serviços possam estar disponíveis em outro nó. Estes tipos de cluster são utilizados para base de dados de missões críticas, correio, servidores de arquivos e aplicações.

Nos clusters de alta disponibilidade os equipamentos são usados em conjunto para manter um serviço ou equipamento sempre ativo, replicando serviços e servidores, evitando com isso a ocorrência de máquinas paradas, ociosas, que estão esperando apenas o outro equipamento ou serviço paralisar, passando as demais a responder por ela normalmente. É claro que com isso pode-se ter perda de performance e poder de processamento, mas o principal objetivo será alcançado, ou seja, não paralisar o serviço (TORRES, 2001).

O cluster HA pode ser observado na Figura 4.1, onde temos terminais clientes ligados por uma rede pública a dois servidores com bases de dados de armazenamento espelhadas.

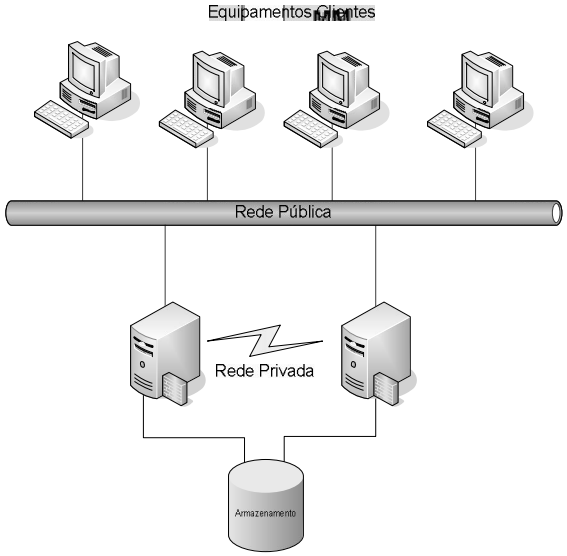

Figura 4.1: Um simples cluster de Alta Disponibilidade.
FONTE: DANTAS (2005).

4.2.2 Balanceamento de Carga (Load Balancing)

Este modelo distribui o tráfego entrante ou requisições de recursos provenientes dos nodos que executam os mesmos programas entre as máquinas que compõem o cluster. Todos os nodos estão responsáveis em controlar os pedidos. Se um nó falhar, as requisições são redistribuídas entre os nós disponíveis no momento (PITANGA, 2003).

Quando não for feito um balanceamento de carga entre servidores que possuem a mesma capacidade de resposta a um cliente, começa-se a ter problemas, pois um ou mais servidores podem responder a requisições feitas e a comunicação fica prejudicada. Por isso, deve-se colocar um elemento que fará o balanceamento entre os servidores e os usuários e configurá-lo para isso. Entretanto podem-se colocar múltiplos servidores onde, para o usuário que estiver utilizando o serviço, aparentará somente uma única máquina (PITANGA, 2003).

O elemento responsável pelo balanceamento entre os servidores e usuários acima é capaz de evitar o desequilíbrio do balanceamento e poderá ser um software dedicado a esse gerenciamento ou um equipamento de rede que combine performance de hardware e software para fazer a passagem dos pacotes e o balanceamento de carga em um só equipamento. Baseado nas afirmações para o sucesso do balanceamento pode-se salientar alguns pontos principais:

- Um algoritmo deve ser utilizado para o balanceamento, onde deverá levar em consideração como é feito o balanceamento entre os servidores e como fazer com que todo processo de escolha e resposta do servidor de serviços feita pelo

cliente torne-se transparente para este cliente, através de um endereço virtual (VS);

- Deve ser criado um método que verifique constantemente se os servidores estão ativos no decorrer do tempo para que a comunicação não seja interrompida ou que a comunicação não seja redirecionada para um nó que acabou de falhar;
- Deve-se criar um método que será utilizado para ter a certeza que um cliente acessa o mesmo servidor quando quiser.

Na Figura 4.2 tem-se um esquema de visualização de um cluster de balanceamento de carga.

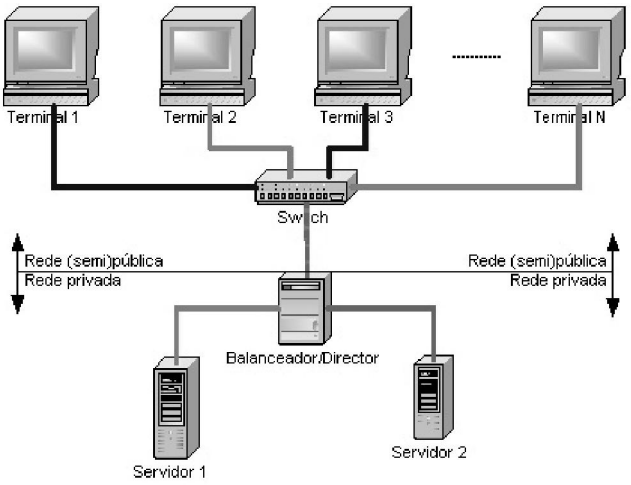

Figura 4.2: Cluster de Balanceamento de Carga.

FONTE: PITANGA (2004).

4.2.3 Combinação HA e Load Balancing

Trata-se de uma combinação envolvendo características de aplicações de Alto Desempenho (HA) e Balanceamento de Carga, proporcionando ao sistema segurança e bom desempenho nas aplicações. Este tipo de configuração de cluster é bastante utilizado em serviços como os de web, mail, ftp, entre outros.

Este tipo de cluster apresenta características como:

- Redirecionamento de solicitações dos nós que apresentam falhas para os nós ativos, chamados nós reservas;
- Melhoria nas atividades e serviços para aplicações típicas de rede;
- O sistema fica disponível tanto para alto desempenho quanto para replicações de informações em outros locais (servidor);

- Disponibiliza uma arquitetura Framework altamente escalável.

4.2.4 Processamento Distribuído ou Processamento Paralelo

Este modelo de cluster aumenta a disponibilidade e performance para as aplicações, particularmente para as grandes tarefas computacionais. Uma grande tarefa computacional pode ser dividida em pequenas tarefas que são distribuídas ao redor das estações (nodos), como se fosse um supercomputador massivamente paralelo. É comum associar este tipo de cluster ao projeto Beowulf da NASA. Estes clusters são usados para computação científica ou análises financeiras, tarefas típicas para exigência de alto poder de processamento.

4.2.5 Cluster de Alta Performance de Computação (HPC)

O cluster HPC é um tipo de sistema para processamento paralelo ou distribuído que consiste de uma coleção de computadores interconectados, trabalhando juntos como um recurso de computação simples e integrado. Um nó do cluster pode ser um simples sistema multiprocessador (PCs, estações de trabalho ou SMPs) com memória ou dispositivos de entrada/saída de dados de um sistema operacional. No entanto, esse sistema pode fornecer características e benefícios (serviços rápidos e confiáveis), encontrados somente em sistemas com memória compartilhada (Multiprocessadores Simétricos – SMP), como os supercomputadores. O esquema do cluster de alta performance de computação pode ser observado na Figura 4.3, mostrada a seguir.

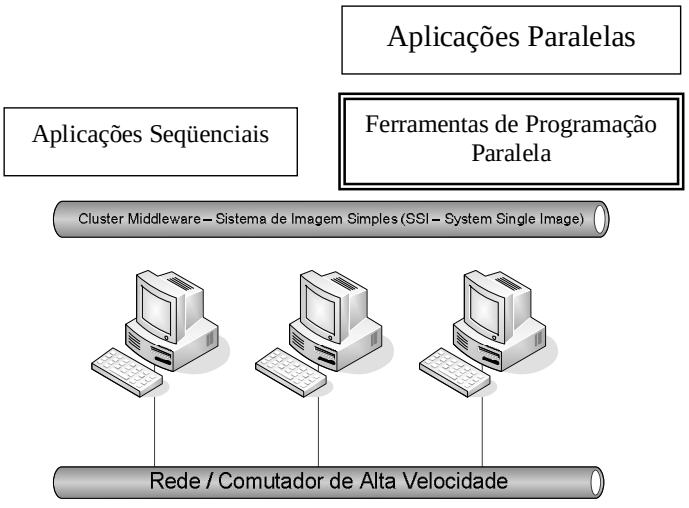

Figura 4.3: Arquitetura de um cluster de PCs.

FONTE: DANTAS (2005).

4.3 Vantagens na Utilização de Clusters de Computadores

Grandes corporações precisam de um enorme poder computacional a baixo custo, como, por exemplo, para renderizar gráficos em alta velocidade, prever o clima, fazer simulações de explosões atômicas e outras tarefas que exigem alto desempenho dos computadores. Empresas que trabalham com serviços de missão crítica precisam de alta disponibilidade para garantir que equipamentos hospitalares e sistemas públicos de emergência estejam sempre acessíveis e funcionando. Manter apenas um computador realizando uma tarefa importante não é uma garantia segura de que o serviço vai estar sempre disponível, pois problemas de hardware ou software podem causar a interrupção do serviço.

Os clusters fornecem desempenho e tolerância à falhas, não encontrados em qualquer sistema com multiprocessamento simétrico (SMP). O desempenho é escalável através do simples acréscimo de computadores ao sistema. Os clusters também oferecem maior disponibilidade, pelo fato de que se um nó falhar, os outros podem continuar fornecendo serviços de dados e a carga de trabalho dos nós defeituosos pode ser redistribuída para os nós restantes.

Dentre as inúmeras vantagens em usar cluster podem se destacar algumas mais importantes:

- Alto Desempenho: possibilidade de resolver problemas muitos complexos através do processamento paralelo, o que diminui o tempo de resolução do problema;
- Escalabilidade: possibilidade de que novos componentes sejam adicionados à medida que cresce a carga de trabalho;
- Tolerância a Falhas: o aumento de confiabilidade do sistema como um todo, caso alguma parte falhe;
- Baixo Custo: a redução do custo para se obter processamento de alta desempenho utilizando-se simples PCs;
- Independência de Fornecedores: utilização de hardware aberto, software de uso livre e independência de fabricantes e licenças de uso.

4.4 Considerações Finais

Neste capítulo foram abordados os conceitos e tipos de clusters existentes atualmente e as vantagens de sua utilização. As tecnologias de Clustering possibilitam a solução de diversos problemas que envolvem grande volume de processamento. As aplicações que um cluster pode ter são diversas, indo desde a simples melhora no desempenho de um

determinado sistema ou a hospedagem de um site, até o processo de pesquisas científicas complexas. O que realmente chama a atenção, é que todo o processamento pode ser feito de maneira que pareça ser um único computador dotado de alta capacidade. Assim, é possível que determinadas aplicações sejam implementadas em cluster, mas sem interferir no funcionamento de outras aplicações que estejam relacionadas.

5 MONTANDO UM AMBIENTE DE HA

5.1 Considerações Iniciais

A alta disponibilidade objetiva manter em operação serviços prestados por um sistema computacional, através da redundância de recursos. Esta idéia se torna mais atrativa quando é utilizado software gratuito operando em computadores de uso comum e de custo mais baixo do que os utilizados em sistemas construídos com esta finalidade. Este capítulo realiza uma análise de algumas das ferramentas disponíveis para o sistema operacional gratuito Linux, constatando, de maneira empírica a cobertura abrangida por essas ferramentas.

5.2 Elementos Necessários

Conforme descrito anteriormente, um ambiente de HA é implementado através de redundância de hardware, tendo uma máquina chamada de nodo principal e outra de secundária, que será como espelho do nodo principal.

Para que uma máquina espelho assuma o lugar da máquina principal quando esta vier a falhar, ambas devem possuir os mesmos dados e/ou recursos envolvidos no tipo de serviço oferecido. A máquina principal e a secundária devem estar configuradas de forma a espelharem os dados, para que em caso de substituição de um sistema por outro, os dados estejam atualizados e consistentes. Apenas tais dados seriam espelhados, pois não faria sentido espelhar uma máquina inteira. O mínimo que aconteceria seria as duas máquinas entrarem em conflito eternamente pelo controle do ambiente.

Os principais fatores envolvidos na montagem de um ambiente de HA, vistos mais detalhadamente, são:

- Hardware: redundância de máquinas (requisito), de links, conexão dedicada e de alta velocidade (recomendada);
- Instalação do sistema: os softwares devem ser independentes de distribuição. A identidade das máquinas fica a cargo do administrador do sistema, devendo-se definir qual será o nodo principal e qual será o nodo espelho;
- Consistência dos dados: sistemas de arquivo *journaled*. Estes sistemas armazenam em um *journal (log)* as ações antes de serem efetuadas. Desta forma, em uma parada no sistema, seja por pane no sistema ou queda de energia, a recuperação é muito mais rápida e sem maiores perdas de informações;

- Espelhamento de dados: os dados devem ser espelhados em tempo real para a completa disponibilidade do sistema em caso de defeito;
- Controle de serviços: com os dados espelhados, os serviços podem ser passados de uma máquina para outra. Porém o sistema deve ser autônomo nesta transferência e ser capaz de se reconfigurar para continuar atendendo aos usuários;
- Monitoração: o sistema deve monitorar seus serviços para detectar um defeito, disparando assim a reconfiguração.

5.3 Exemplo de um ambiente de alta disponibilidade

Atualmente já existem muitas ferramentas para fornecer sistemas com tolerância a falhas e alta disponibilidade, porém algumas delas ainda são muito limitadas e passam por aperfeiçoamentos e testes. Constantemente são lançadas novas versões dessas ferramentas. No entanto, existem algumas outras que apresentam alto grau de confiança e podem ser utilizadas de maneira segura comercialmente.

As ferramentas para alta disponibilidade podem ser classificadas de acordo com sua funcionalidade:

- Replicação de dados: utilizadas quando se deseja realizar espelhamento;
- Detecção de falhas: fazem verificações no sistema, de modo que falhas sejam detectadas e ações possam ser tomadas (desde que a falha esteja estipulada para o sistema de detecção).

Dentre as ferramentas existentes, foi escolhida uma solução composta de representantes das categorias citadas acima:

- Sistema de Arquivos *Journaled* - ext3: o ext3 possui uma vantagem significativa sobre todos os outros sistemas atuais, que é a sua total compatibilidade com o sistema de arquivos ext2. O ext3 é praticamente um ext2 acrescido das propriedades de "*journal*". Conseqüentemente, pode empregar todas as aplicações existentes desenvolvidas para manipular o sistema de arquivos ext2, bem como permitir uma migração para ext3 sem grande esforço;
- Espelhamento de dados - DRBD: o DRBD (Distributed Replicated Block Device) espelha os dados em blocos, repassando-os através da rede. Funcionando em pares, ele estabelece em cada nodo uma partição para espelhar e ajusta um dos nodos como primário. Toda alteração neste nodo irá

ser refletida no nodo secundário. O nodo primário possui a partição montada com DRBD montada, enquanto o secundário não possui;

- Sinal de Vida - Heartbeat: nele reside a configuração dos serviços e do "ip de serviço", sendo o responsável pela reconfiguração automatizada na ocorrência de problemas. Cada máquina possui seu ip próprio, válido ou não, mas o conjunto é identificado por um ip de serviço, que é mantido pela máquina primária no par. Ao acontecer uma falha no sistema, o Heartbeat de um lado libera os serviços e ips e, do outro, assume;
- Monitoração - Mon: bastante leve, possuindo diversos monitores e sistemas de alerta para as mais variadas funções e serviços. Possui também envio de mensagens via página, correio, sms, bem como pode ser configurado para disparar programas em quaisquer linguagens.

A justificativa para se usar vários aplicativos e não um sistema geral deve-se ao fato de que, além de serem muito mais portáveis e leves, estes softwares podem ter várias outras aplicações. Pode ser, por exemplo, desejar apenas monitorar um servidor ou pode ser que se possua um *storage* externo compartilhado por duas máquinas e queira-se coordenar o controle deste.

Estas ferramentas operando em conjunto pretendem fornecer um ambiente com alto grau de disponibilidade, funcionando de maneira automática, embora em determinadas situações haja a necessidade de intervenção manual para recuperação do sistema.

5.4 O Software DRBD

DRBD é um módulo de *kernel* e *scripts* associados que oferecem um dispositivo de bloco projetado para construir clusters de alta disponibilidade. Isto é feito espelhando conjunto de blocos via rede (dedicada). Ele pode ser visto como um raid via rede.

O DRBD toma conta dos dados, escritos no disco rígido local e envia-os ao outro host. No outro host, ele escreve os dados no disco.

Cada dispositivo (o DRBD providencia mais de um destes dispositivos) tem um estado, que pode ser primário ou secundário. No nodo com o dispositivo primário, a aplicação está executando e tem acesso ao dispositivo (/dev/nbX). Toda escrita é enviada para o "dispositivo de bloco do nível mais baixo" e para o nodo com o dispositivo secundário. O secundário simplesmente escreve o dado no "dispositivo de bloco do nível mais baixo". Leituras são sempre realizadas localmente.

Se o nodo primário falhar, o *Heartbeat* permutará o dispositivo secundário em primário e iniciará a aplicação no nodo secundário.

Se o nodo que falhou retornar, ele se torna o secundário e tem que sincronizar seus dados com o primário. Isto é feito sem nenhuma interrupção do serviço em *background*.

A maioria dos clusters HA atuais como (HP, Compaq,...) estão usando dispositivos de armazenamento compartilhados, desta forma estes dispositivos são conectados a mais de um nodo. Isto pode ser feito com barramento SCSI compartilhado ou *Fibre Channel*.

O DRBD usa a mesma semântica de um dispositivo compartilhado, mas ele não necessita de nenhum hardware incomum. Ele trabalha o topo de redes IP, as quais são mais baratas do que redes com sistemas de armazenamento especial.

Atualmente, o DRBD garante acesso de leitura-escrita apenas em um nodo por vez, o que é suficiente para o *failover* usual de um cluster HA.

5.4.1 Instalando o DRBD

Atualmente o DRBD utiliza uma área para os meta-dados (informações sobre os blocos a serem ou que estão sendo espelhados) em uma partição definida pelo administrador. Esta partição pode ser a mesma que está sendo espelhada ou uma outra, e a regra é que cada partição espelhada gasta 128Mb com estas informações. Então, se pode optar por incluir este espaço de armazenamento no espelhamento e deve-se considerar que este espaço na partição está reservado. Também pode-se criar uma partição especialmente para a utilização dos meta-dados e indicar na configuração.

A primeira etapa na instalação do DRBD é descompactar o programa, utilizando o comando:

```
root@toskinha:~/# tar zxvf  drbd-0.7.13.tar.gz
```

É preciso ter o programa fonte do *kernel*, que deve ser da mesma versão que estiver em uso pelo sistema, apontado pelo link /usr/src/linux. Por exemplo, na máquina teste, tem-se:

```
toska@toskinha:~$ uname -r
2.6.12.3
toska@toskinha:~$ ls -l /usr/src/
lrwxr-xr-x   1 root root   15 2005-07-30 22:16 linux -> linux-2.6.12.3/
```

No diretório do DRBD, é compilado e instalado o programa, através das instruções:

root@toskinha:~/# cd drbd-0.7.13/drdb

root@toskinha:~drbd-0.7.13/drbd# make clean all

Se forem usados os programas fontes do Linux em outro diretório, deve-se informar este caminho com a variável KDIR da seguinte maneira:

root@toskinha:~drbd-0.7.13/drbd# make clean ; make

KDIR=/path/to/kernel/source

Neste momento, são criadas as ferramentas drbdsetup e modprobe, através das seguintes linhas de comando:

root@toskinha:~drbd-0.7.13/drbd# cd ..

root@toskinha:~drbd-0.7.13# make tools

E finalmente, instalar o programa com a instrução:

root@toskinha:~drdb-0.7.13# make install

5.4.2 Configurando o DRBD

O drbdsetup é uma ferramenta de configuração do ambiente do DRBD e pode ser usado para associar dispositivos DRBD com dispositivos de bloco de nível mais baixo, configurar pares de dispositivos DRBD para espelharem seus dispositivos de bloco e também verificar as configurações dos dispositivos DRBD que estão rodando. É conveniente primeiro levantar um ambiente via drbdsetup e, depois, partir para o arquivo de configuração, pois esta tarefa auxilia na depuração de erros que possam vir a acontecer, além de maior controle ao administrador da configuração.

É assumido que as duas máquinas, chamadas de nodo1 (10.0.0.10) e nodo2 (10.0.0.20), usarão os dispositivos de armazenamento /dev/hda6 como dispositivo de bloco de nível mais baixo em ambas as máquinas. No nodo2, serão executados os seguintes comandos:

modprobe drbd

drbdsetup /dev/drbd0 disk /dev/hda6

drbdsetup /dev/drbd0 net 10.0.0.20 10.0.0.10 B -r 10M

Na primeira linha é carregado o módulo do DRBD. A segunda linha indica que o DRBD vai utilizar a partição hda6 como espelho, já na terceira linha é indicado que o espelhamento ocorre através da interface associada ao ip 10.0.0.20 na máquina e remotamente com o ip 10.0.0.10, através do protocolo B, a ser visto posteriormente. O parâmetro -r 10M indica que a taxa de transmissão pode chegar a 10Mbs. A vantagem desta configuração é proporcional ao tamanho da sua partição. O DRBD infelizmente leva bastante tempo para

sincronizar dados. E por padrão, ele vem ajustado com uma taxa de transmissão de 256 bits/s. Desta forma, economiza-se bastante tempo com o sincronismo dos nodos.

No nodo1 são executados os mesmos comandos, apenas mudando as informações relativas ao nodo:

```
# modprobe drbd
# drbdsetup /dev/drbd0 disk /dev/hda6
# drbdsetup net 10.0.0.10 10.0.0.20 B -r 10M
# drbdsetup /dev/drbd0 primary
```

Neste último comando, é indicado que este nodo será o primário. A verificação do conteúdo do diretório "/proc/drbd", através do comando "cat /proc/drbd", por exemplo, pode verificar o progresso da sincronização. Se os nodos estiverem em estado de SyncAll, é sinal que estão sincronizando. Quando a mensagem mudar para Connect, a sincronização está feita.

No exemplo acima, o protocolo B é usado. O DRBD permite selecionar o protocolo com a qual irá controlar como os dados serão escritos no dispositivo secundário. Estes protocolos podem ser vistos na Tabela 5.1.

Tabela 5.1. Protocolos DRBD

Protocolo	Descrição
A	Uma operação de escrita é considerada completa tão logo o dado é escrito no disco e enviado para a rede.
B	Uma operação de escrita é considerada completa tão logo a confirmação de recepção pelo outro nodo chegue.
C	Uma operação de escrita é considerada completa tão logo a confirmação de escrita venha do outro nodo.

FONTE: Elaboração própria.

O protocolo A é mais leve e mais rápido, porém os protocolos B e C são mais rígidos com os dados. Para o espelhamento em uma rede confiável e isolada, os protocolos B e C são mais recomendáveis para maior segurança. Se os utilizar numa rede comum a outras máquinas, o consumo vai ser bastante elevado, sendo melhor o protocolo A.

5.5 O Software *Heartbeat*

Heartbeat é uma ferramenta de monitoramento de computadores (nodos) pertencentes a um cluster de alta disponibilidade. *Heartbeat* (do inglês, batimento cardíaco) faz a verificação destes nodos do cluster da seguinte maneira: cada nodo do cluster envia, a cada intervalo de tempo (configurável), um "*ping*" ao outro nodo avisando que está em atividade. Do outro lado, cada nodo sempre espera receber esse aviso, caso contrário, o nodo é considerado parado ou desligado; então se o nodo falho é o ativo, ou seja, o que atualmente está prestando os serviços, tem esses serviços "tomados" por outro nodo, que agora passa a responder pelos serviços e atender as requisições dos usuários (GARCIA, 2001).

É recomendável utilizar-se de um meio de comunicação dedicado para o Heartbeat, pois isso elimina questões relacionadas ao tráfego de rede ou mesmo partição da rede. O mais comum é utilizar a interface serial para a comunicação entre eles, embora também seja possível utilizar a comunicação via rede (intranet ou Internet) ou mesmo através de uma rede dedicada. Essa possibilidade garante uma maior tolerância a falhas do sistema, pois em caso de falha do hardware (a porta serial, por exemplo) o *Heartbeat* ainda tenha outros caminhos alternativos. Com isso, a probabilidade de ocorrerem falhas em todos os meios de comunicação ao mesmo tempo entre os nodos é muito menor.

Através do *Heartbeat* é possível controlar uma infinidade de processos e serviços nos nodos ou criar *scripts* que possam realizar algum tipo de ação ou tarefa, tudo de maneira automática.

5.5.1 Instalando o *Heartbeat*

O passo inicial para a instalação do *Heartbeat* é descompactar e compilar o programa, através das seguintes linhas de comando:

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

O *Heartbeat* traz um *script* para integrar o DRBD, que é o datadisk, instalado no diretório /etc/ha.d/resource.d/datadisk. O datadisk negocia a troca do dispositivo DRBD de secundário para primário, monta as partições DRBD especificadas no fstab e pode ser chamado do arquivo haresources. Ele trata das partições DRBD especificadas no fstab. Então, se é desejado que o *Heartbeat* tome a tarefa de montar as partições DRBD, deve-se tomar conhecimento da inclusão da sua configuração no fstab.

5.5.2 Configurando o *Heartbeat*

Posteriormente, deve ser criado um diretório /etc/ha.d, onde ficam os arquivos de configuração. Do diretório de instalação, deve-se copiar os arquivos ha.cf, haresources e authkeys do diretório doc para /etc/ha.d. Primeiro será configurado o ha.conf. conforme parâmetros da tabela abaixo:

Tabela 5.2: Parâmetros de configuração do arquivo /etc/ha.d/ha.cf

Parâmetro	Descrição	Exemplo
debugfile caminho	Muito útil para debug de configuração	debugfile /var/log/ha-debug
logfile caminho	Armazena as mensagens do processo	logfile /var/log/ha-log
Serial dispositivo	Especifica a interface serial a ser utilizada	serial /dev/ttSy0
keepalive tempo	Configura o tempo entre as verificações	keepalive 2
deadtime tempo	O nodo será declarado como indisponível depois de tempo segundos	deadtime 2
Baud velocidade	Velocidade da porta serial (bps)	baud 19200
udpport porta	Porta a ser usada caso seja utilizada interface de rede	udpport 694
udp interface	Interface a ser usada pelo heartbeat para broadcast	udp eth0
node nodo 1	Hostname da maquina nodo conforme descrito em ‘uname –a’	node linuxha1
node nodo 2	Hostname da maquina nodo conforme descrito em ‘uname –a’	node linuxha2

FONTE: Elaboração própria

Uma vez configurado o ha.cf é preciso ajustar o arquivo haresources. Este arquivo especifica os serviços para o cluster e quem é o proprietário padrão. Para o referido exemplo, é assumido que os serviços são o DRBD e o apache. O endereço IP para o cluster é obrigatório e não pode ser configurado o IP do cluster fora do arquivo haresources. O arquivo irá precisar de uma linha de comando:

linuxha1 192.168.85.3 datadisk httpd

Esta linha diz que ao iniciar, linuxha1 serve o IP 192.168.85.3 e inicia o apache e o DRBD. Ao desligar, o Heartbeat irá primeiro parar o apache, o DRBD e depois liberar o IP. Isto supõe que o comando ‘uname –n’ tem como saída linuxha1.

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 diretórios:

/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 utiliza-se qualquer serviço tão logo se faça um script conforme estas especificações.

É preciso passar argumentos para um script customizado e o formato deve ser: Nome_script::argument. Então, adiciona-se um serviço “serv” que precisa de um argumento “arg”, o arquivo haresources deverá ser assim configurado:

linuxha1 192.168.85.3 datadisk httpd maid::vacuum

Isto traz mais flexibilidade com o endereço IP de serviço, sendo usada em uma notação abreviada. Esta linha pode assim ser lida:

linuxha1 IPaddr::192.168.85.3 datadisk httpd

onde IPaddr é o nome do script do serviço, tendo como argumento 192.168.85.3. Observando o diretório /etc/ha.d/resource.d, é encontrado um script chamado IPaddr. Este script irá também permitir a manipulação do endereço de broadcast e a máscara de rede do seu serviço IP. Para especificar uma subrede com 32 endereços, pode-se definir o serviço como:

linuxha1 192.168.85.3/27 httpd smb

Este comando configura o endereço do serviço IP para 192.168.85.3, a máscara de rede para 255.255.255.224 e o endereço de broadcast para 192.168.85.31 (o qual é o endereço mais alto da subrede). O último parâmetro que pode ser configurado é o endereço de broadcast. Para sobrescrever o padrão e ajustá-lo para 192.168.85.16, deve-se fazer:

linuxha1 192.168.85.3/5/192.168.85.16 httpd smb

Qualquer uma das alternativas acima é necessária para configuração. Isto depende, pois se houver uma rota estabelecida (independente do *Heartbeat*) para os endereços IP do serviço, com a máscara e broadcast corretos, então isto não é necessário. Entretanto, este caso não é utilizado de maneira geral e este é o motivo destas opções. Em adição, pode-se ter mais de uma interface possível que pode ser usada para o serviço IP.

Com os arquivos haresources para o ha.cf e haresources para /etc/ha.d configurados estes estão prontos para iniciar.

O *Heartbeat* fornece também a possibilidade de autenticação e segurança de transmissão das informações necessárias para o monitoramento. Existem três tipos de métodos, conforme a Tabela 5.3:

Tabela 5.3: Métodos de autenticação para o Heartbeat.

Método	**Caso de Uso**
crc	Rede segura (porta serial, ou rede com canal dedicado). Método mais barato computacionalmente.
md5	Rede insegura, mas com uma preocupação com segurança e menor sobrecarga da CPU.
sha1	Rede insegura. Método mais seguro que o md5, porém exige maior sobrecarga da CPU.

FONTE: Elaboração própria.

Finalmente, para uma melhor autenticação sem se preocupar com recursos de CPU, deve-se usar o sha1, por ser o mais robusto.

O formato do arquivo é o seguinte:

auth <número>

<número> <método de autenticação> [<chave de autenticação>]

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

auth 1

1 - sha1 - cluster

Para md5, pode-se usar o mesmo, apenas trocando ‘sha1’ por ‘md5’.

Finalmente para crc, pode ser:

auth 2

1 crc

Um aspecto importante da configuração do arquivo haresources para uma máquina que tenha múltiplas interfaces ethernet é saber como o *Heartbeat* seleciona qual interface irá servir o endereço de serviço que foi configurado. Afinal, nenhuma interface foi especificada no haresources.

O *Heartbeat* decide qual interface usar observando a tabela de roteamento. Ele tenta selecionar a rota de menor custo para o endereço IP que irá ser assumido. No caso de um empate, ele escolhe a primeira rota encontrada. Para a maioria das configurações isto significa que a rota *default* terá preferência.

Se não for especificada uma máscara de rede para o IP no haresources, será usada a que estiver associada com a rota escolhida.

5.5.3 Iniciando o *Heartbeat*

O *Heartbeat* deve ser iniciado, cujo script deve estar no diretório /etc/init.d/:

#/etc/init.d/heartbeat start

Pode-se acompanhar o que está acontecendo pelos logs, que estão no diretório /var/log/ha-log. Caso ocorra algum erro, é verificado a causa nesse arquivo.

O *Heartbeat* agora irá tratar da inicialização dos serviços que foram configurados no haresources. Caso a máquina principal pare de responder, a secundária irá assumir e iniciar os mesmos recursos.

5.6 O Software Mon

O software escolhido para monitoração foi o Mon - Service Monitoring Daemon. Seguindo a configuração, deve-se ter a linguagem de programação Perl instalada e alguns de seus módulos, tais como:

-Time::Period

-Time::HiRes

-Convert::BER

-Mon::*

A instalação dos módulos é muito simples e realizada de acordo com as seqüências de linhas de comandos a seguir:

#bzip2 –cd modulo.bz2 | tar xvf –

cd modulo

perl Makefile.pl

make

make test (para verificar se o processo está correto)

make install

Esta ferramenta trabalha baseada em *scripts*. O *script* de verificação mais utilizado é o fping, que a cada intervalo de tempo envia um comando semelhante à um ping para algum host (podem ser definidos quantos hosts se achar conveniente) e se para todos estes hosts a operação retornar falha, então MON considera que o nodo está isolado. A partir daí, de acordo com os scripts definidos, um conjunto de ações são tomadas para tentar contornar essa situação indesejável.

Além do script fping, outros estão disponíveis e são responsáveis pelas ações pós-detecção. Estão disponíveis junto com a ferramenta, scripts que enviam um e-mail para algum endereço (do administrador, por exemplo), que enviam uma mensagem a um aparelho de bip ou telefone celular, gravam registros no log, entre outros; e também podem ser definidos pelo administrador outros scripts. Em princípio pode parecer estranho um nodo isolado enviar um

e-mail ou mensagem em um aparelho de bip; no entanto, é importante ressaltar que o MON pode fazer monitoramento remoto de nodos, de modo que se este não responde, está isolado (ou pelo menos assim é considerado). Esse monitoramento pode ser feito da seguinte maneira: o administrador pode utilizar o MON para realizar verificações dos nodos do cluster (enviando "fpings") de um computador fora do cluster; quando estes não respondem, a ferramenta então executa as tarefas predefinidas (e-mail, mensagem de bip, etc.). A partir desse monitoramento remoto é possível enviar um e-mail ou mensagem para o aparelho de bip do administrador. Uma característica peculiar e interessante desta ferramenta é que, dependendo dos dias da semana (dia útil ou final de semana) e de acordo com o horário (horário comercial ou não comercial) diferentes ações podem ser tomadas. Por exemplo, para horário comercial em dias úteis, se uma falha ocorre, o MON pode ficar emitindo um sinal sonoro, mas se for em horário fora do comercial (madrugada, por exemplo) ele pode enviar uma mensagem para o aparelho de bip do administrador ou para seu celular.

A configuração do MON é realizada através do arquivo MON.cf que está localizado dentro do diretório MON (definido pelo usuário). Nele são definidos os caminhos para os arquivos necessários para o funcionamento da ferramenta, os nodos e os scripts que serão executados em caso de problemas (alertas). Para cada tipo de serviço (http, telnet, etc.) podem ser construídas diferentes ações, todas no mesmo arquivo.

Como pode acontecer de a máquina estar ela mesma com problemas, além do IP do outro nodo, colocamos mais ips para que isto seja detectado. Caso nenhum IP responda, a máquina está isolada e pára o Heartbeat.

Ao executar um fping (comando que envia vários pacotes simultaneamente) para os servidores definidos e nenhum responder, é sinal de que a máquina está isolada. Então ele envia um mail para o root e desliga o *Heartbeat*. Coloque os monitores e alertas nos diretórios definidos (vem junto com o mon). Insira o executável mon no diretório definido em cfbasedir. Feito isso, inicie o Mon, através da linha de comando:

```
#/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 hub, o cabo, o que for, o Mon dispara o *shutdown* do *Heartbeat*, que levará a outra máquina a assumir o endereço IP e os serviços.

5.7 Considerações Finais

Este capítulo abordou os elementos necessários para implementação de um ambiente de alta disponibilidade utilizando hardware de baixo custo e softwares gratuitos, oferecendo assim uma visão geral do que está sendo envolvido na alta disponibilidade.

Foram analisadas as ferramentas envolvidas no processo de implementação da alta disponibilidade descrevendo cada uma delas e demonstrando passo-a-passo a instalação e configuração das respectivas ferramentas.

6 TESTES REALIZADOS NO AMBIENTE DE HA

6.1 Considerações Iniciais

Através deste estudo de caso, tentou-se verificar o quão satisfatório se encontram atualmente as ferramentas existentes. Dado a existência de considerável quantidade, algumas delas foram escolhidas (escolha feita baseada em relatos de sucesso na experimentação dessas ferramentas encontrados na Internet) e alguns testes realizados para uma avaliação das mesmas.

A avaliação se deu pela injeção de falhas que podem ocorrer com maior freqüência em sistemas computacionais. Os tipos de falhas são:

- **Queda de energia**: um sistema computacional, assim como qualquer outro que dependa de fontes externas de alimentação, estar sujeito a uma interrupção no fornecimento de energia e, embora possam haver alternativas de proteção contra esse tipo de problema, em alguns casos não é possível manter o funcionamento do sistema. Este tipo de problema termina quando a fonte de alimentação volta ao seu funcionamento normal;
- **Falhas de hardware:** quando alguma parte física do sistema computacional apresenta mau funcionamento. Componentes como discos, interfaces de comunicação (rede, porta serial), processador, entre outros, podem apresentar desgaste com o passar do tempo, vindo a falhar. Este tipo de falha se resolve substituindo o componente defeituoso;
- **Indisponibilidade de comunicação:** devido à alguma anomalia, o sistema não consegue receber requisições vindas dos clientes. Isso pode acontecer por tráfego excessivo no canal de comunicação, causando lentidão na rede ou por uma falha de hardware na interface de comunicação. No caso do primeiro problema, a resolução é mais difícil, pois depende de agentes externos ao controle do sistema.

6.2 Configuração do Sistema

Os testes foram realizados em um cluster de duas máquinas, com computadores pessoais e contavam com a configuração descrita na Tabela 6.1. Cada teste foi repetido três vezes, e assim pode-se avaliar de maneira empírica a sua eficiência.

Tabela 6.1: Máquinas e softwares utilizados nos testes

Quantidade de máquinas	02
Processador Nodo 1	Atlhon XP 2.7
Processador Nodo 2	Pentium 4 3.0
Memória Principal Nodo 1	512 Mb
Memória Principal Nodo 2	2 Ghz
Disco Nodo 1	10 Gb
Disco Nodo 2	10 Gb
Interface eth0 Nodo1	Placa de rede 10/100 Mbps – 3Com
Interface eth0 Nodo2	Placa de rede 10/100 Mbps – Via
Cabo de Rede	Par Trançado, conector RJ 45
HUB	10/100 Mbps
Distribuição do sistema operacional	Slackware 9.0
Versão do kernel do sistema operaciona	2.4.26
DRBD	0.6.1-pre18
Heartbeat	0.4.9.2-1
MON	0.99.2

FONTE: Elaboração Própria

Neste estudo de caso, o DRBD foi configurado para servir dados de um disco.

A ferramenta *Heartbeat* é responsável pelo controle do servidor HTTP (Apache Web Server) e também controla a montagem/desmontagem automática da partição de dados, através da utilização de scripts fornecidos pelo DRBD (datadisk).

A ferramenta MON foi configurada para, em caso de falha, gravar os registros no log e reiniciar o sistema. Se em três tentativas de reinicialização ainda permanecer com problemas, pára o processo Heartbeat e o MON fica emitindo um sinal sonoro. Ao desligar o *Heartbeat*, o outro nodo considera que o primeiro está falho (pois este não está mais enviando sinais via *Heartbeat*) e então assume os serviços, passando a responder por eles.

A Figura 6.1 mostra a configuração lógica do cluster proposto.

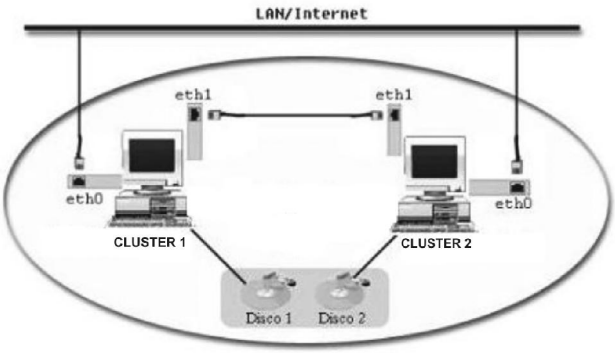

Figura 6.1: Visão do cluster de alta disponibilidade

FONTE: DANTAS (2005).

6.3 Teste 1: Queda e Retorno do Nodo Primário

O primeiro teste que foi realizado é o mais básico, onde o nodo 1 (o nodo 1, nos testes, foi considerado como sendo o primário) sofre uma falha de comunicação e após um tempo retorna (automaticamente ou não). Neste caso, o nodo 2 (secundário) passa a ser o primário. Ao retornar, o nodo 1 faz a sincronização dos dados, já que durante a sua inoperabilidade podem ter sido realizadas modificações. Este tipo de procedimento é comum quando ocorre interrupção da alimentação de energia em somente um dos nodos. No caso de um retorno rápido ou demorado, o estado deverá ser o mesmo e, dessa forma, o nodo falho (nodo 1) deverá permanecer em estado secundário.

6.4 Teste 2: Queda e Retorno Básico do Nodo Secundário

Bem semelhante ao caso do teste 1. O nodo 2 (secundário) falha e retorna (independentemente do tempo decorrido entre a falha e seu retorno à atividade) e após o retorno deste, o nodo 1 (primário) falha. Neste caso a expectativa é que o sistema funcione de maneira normal, com o nodo 2 assumindo o posto de primário após a falha do nodo 1.

6.5 Teste 3: Falha na Interface de Rede Dedicada

Como dito anteriormente, o DRBD utiliza uma rede dedicada para transmissão de informações. Assim, uma falha nesse canal (problemas no cabo ou defeito na interface de rede) faz com que não seja possível realizar o espelhamento. A sincronização é feita quando o canal for restabelecido. O DRBD não sabe se o que ocorreu foi a queda do canal ou do nodo, ele considera que o nodo não está disponível e vai armazenando informações sobre as modificações realizadas. A ferramenta verifica periodicamente se o outro já retornou, quando este retornar a sincronização é realizada.

6.6 Teste 4: Falha no Disco de Espelhamento do Nodo 1

Um equipamento que com o passar do tempo também pode apresentar falhas é o disco rígido. No caso deste trabalho, foi utilizado um disco exclusivo para dados, de maneira que a falha neste preservasse o sistema operacional e a falha no disco principal (onde está instalado o sistema operacional) garantisse a consistência dos dados. Neste tipo de problema, a solução somente se dá pela troca do disco (intervenção manual).

6.7 Considerações Finais

Pela análise dos resultados pode-se perceber que as ferramentas escolhidas (DRBD, Heartbeat e MON) conseguiram lidar com a maioria dos problemas de maneira satisfatória, obedecendo em quase todas as situações as características de sistemas de alta disponibilidade.

A ferramenta DRBD apresentou demora para a sincronização dos discos. Salvo quando é executada a sincronização rápida, o processo é extremamente lento, mesmo que nenhuma alteração tenha sido efetuada e também mesmo que a taxa de transmissão fosse alta. Aliado a isso está à impossibilidade de poder realizar ações corretivas em caso de falha dos discos. A ferramenta *Heartbeat* ainda é muito limitada, pois para a utilização em mais de dois nodos é exigida a existência de mais canais dedicados de comunicação. Por exemplo, se temos três nodos N1, N2 e N3, e desejamos utilizar o Heartbeat, então precisamos conectar N1 e N2 por uma interface serial (ou outra qualquer) e N2 e N3 por uma outra interface (que pode ser uma segunda interface serial em N2 ou uma interface de rede), já que cada ligação é única. O mesmo vale para a ligação entre N1 e N3. Isso se torna caro e conseqüentemente não atrativo em termos de hardware à medida que aumenta a quantidade de nodos no cluster.

Embora apresentem esses problemas, as ferramentas operando em conjunto permaneceram estáveis enquanto estão funcionando em um sistema como o que foi proposto.

7 CONCLUSÃO

Tolerâncias a falhas e alta disponibilidade são áreas de grande interesse, principalmente do mercado corporativo, onde há uma grande preocupação de pleno e completo fornecimento de serviços/recursos aos usuários. A possibilidade de se construir sistemas tolerantes a falhas utilizando equipamentos de baixo custo e de propósito genérico (como é o caso dos computadores pessoais) tem se tornado o principal atrativo desse tipo de solução, juntamente com a possibilidade destes equipamentos baratos serem gerenciados por um sistema operacional de baixo custo, onde se encaixa o sistema Linux.

Distribuições Linux já estão disponibilizando ferramentas que fornecem alta disponibilidade; não só as que foram apresentadas neste trabalho, mas também outras soluções. Empresas distribuidoras de versões do sistema operacional também têm dado grandes contribuições no desenvolvimento e aprimoramento dessas ferramentas, a fim de deixá-las mais robustas e eficientes.

Embora haja um grande número de pessoas e entidades envolvidas com alta disponibilidade em Linux, a documentação ainda apresenta pouco nível de detalhamento, principalmente sobre problemas que ocorrem durante a instalação, configuração e utilização do sistema. A documentação dessas ferramentas não reporta a abrangência de falhas que as mesmas suportam, de tal maneira que muitas dificuldades surgiram no decorrer dos estudos, onde não era possível definir o comportamento do sistema em determinadas situações. A todo o momento era necessário recorrer às listas de discussão para obter informações sobre problemas e suas soluções. Este trabalho serve, então, para que pessoas que desejem construir um sistema com essas características possam identificar e saber pelo menos alguns dos comportamentos que o sistema pode apresentar em situações atípicas.

Este trabalho abrangeu tanto aspectos teóricos quanto práticos de alta disponibilidade. Através dele foi possível analisar uma das soluções existentes atualmente no sistema Linux e verificar que já apresentam resultados satisfatórios. As ferramentas escolhidas (DRBD, Heartbeat e MON) foram instaladas e configuradas para suportar a falha de um dos nodos do cluster. Foram realizados testes em que essa limitação era quebrada, de maneira que situações de falha ocorressem em ambos os nodos, e foi possível avaliar que mesmo não sendo abrangido pela solução proposta, a consistência dos dados era preservada.

Através da análise dos testes pôde-se avaliar quais situações o sistema comporta-se de maneira estável e em quais ainda necessitam ser resolvidos problemas que causam

indisponibilidade do sistema. De maneira geral as ferramentas demonstraram serem de grande auxílio na construção de sistemas tolerantes a falhas.

Este trabalho teve como objetivo a avaliação de maneira empírica a eficiência das ferramentas de alta disponibilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

DANTAS, Mário. **Computação Distribuída de Alto Desempenho Redes, Clusters e Grids Computacionais**, ed. Axcel Books, 2005.

FLYNN, M. **Some Computer Organizations and Their Effectiveness**, IEEE Transaction on Computer, Vol. C-21, pp.94, 1972.

PEIXOTO, L., Escalonamento Adaptativo em Sistemas Paralelos. Proposta de Dissertação de Mestrado. Universidade de Minho Campus de Gualtar – Portugal 2002.

PITANGA, Marcos. **Construindo Supercomputadores com Linux** ed. Brasport, 2004.

PITANGA, Marcos. **Computação em Cluster** ed. Brasport, 2003.

JALOTE, Pankaj. **Fault Tolerance in Distributed Systems**. Prentice-Hall, 1994.

TORRES, Gabriel. **Redes de Computadores: Curso Completo** Axcel Books, 2001.

Alta Disponibilidade documentação, sem autoria, Disponível em: http://www.linux-ha.org. Acessado em 05 de janeiro de 2007.

DRBD documentação, sem autoria, Disponível em: http://www.drbd.org. Acessado em: 05 de janeiro de 2007.

FILHO, N., Linux, Cluster e Alta Disponibilidade. Universidade de São Paulo, 2002. Disponivel em: http://ime.usp.br/~nelio/publications/linuxha/html/>. Acessado em 15 de novembro de 2006.

MORIMOTO, Carlos. Solução Conectiva para Alta Disponibilidade 2002, Disponível em: http://www.conectiva.com.br. Acesso 10 de novembro de 2006.

PITANGA, Marcos. Computação em Cluster, Disponível em: http://www.clubedohardware.com.br/artigos/153. Acesso em 21 de outubro de 2006.

RUIZ, A. Alta Disponibilidade em servidores Linux. Revista do Linux, 2000. Disponível em: http://www.revistadolinux.com.br/ed/006/alta.php3. Acesso em 21 de novembro de 2006.

VIGLIAZZI, Douglas, Alta Disponibilidade (High Availability) em sistemas GNU/Linux, Disponível em: http://www.vivaolinux.com.br/artigos/verArtigo.php?codigo=52&pagina=3. Acessado em 10 de janeiro de 2007.

MACHADO, Carlos. Cluster com Linux na Mão!. Revista Info Exame, p.104-106, setembro, 2004.