THIAGO DO PRADO RAMOS

ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA PARA SERVIDORES WEB USANDO LINUX

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

LAVRAS

MINAS GERAIS – BRASIL

2009

THIAGO DO PRADO RAMOS

ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA PARA SERVIDORES WEB USANDO LINUX

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Área de Concentração:

Redes de Computadores

Orientador

Prof Dr Luiz Henrique Andrade Correia

LAVRAS

MINAS GERAIS – BRASIL

2009

THIAGO DO PRADO RAMOS

ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA PARA SERVIDORES WEB USANDO LINUX

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras, como parte das exigências do Curso de Ciência da Computação, para a obtenção do título de Bacharel em Ciência da Computação.

Aprovada em ____ de _____ de _____

_____

Prof. DSc. Remulo Maia Alves

_____

Prof. DSc. Marluce Rodrigues Pereira

_____

Prof. DSc. Luiz Henrique Andrade Correa
(orientador)

LAVRAS

MINAS GERAIS – BRASIL

Agradecimentos

Agradeço a minha esposa por sempre estar ao meu lado e me dar incentivo nos momentos difíceis,

A minha mãe pela confiança que sempre depositou em mim,

A minha família pelo incentivo,

Aos Professores da Universidade Federal de Lavras, em especial, Professor Luiz Henrique Andrade Correia e Remulo Maia Alves pela oportunidade de colocar em pratica os conhecimentos adquiridos.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA
PARA SERVIDORES WEB USANDO LINUX

Este trabalho visa à implantação de um cluster para serviços Web que possua alta disponibilidade e balanceamento de carga em caso de falhas. Clusters de alta disponibilidade são sistemas desenvolvidos para fornecer disponibilidade de serviços de forma ininterrupta, utilizando redundância de computadores. Se um computador falhar, outro assume automaticamente executando os mesmos serviços e dados. As falhas de serviços podem ser decorrentes de problemas de hardware ou sobrecarga de serviços nos servidores. O problema de sobrecarga pode ser amenizado utilizando balanceamento de carga, distribuindo-se as requisições com outros computadores do sistema, que executam os mesmos programas, reduzindo o tempo de processamento e melhorando o desempenho da aplicação. Foram avaliados os métodos de implementação de clusters, seus modelos e conceitos e quais os melhores resultados para garantir um nível de qualidade compatível com as soluções proprietárias disponíveis. É mostrado que o uso de cluster é eficiente no balanceamento de carga e na alta disponibilidade.

Palavras-chave: Cluster, alta disponibilidade, balanceamento de carga

ABSTRACT

This paper aims at implementation of a cluster for Web services that have high availability and load balancing in case of failures. Clusters of high-availability systems are designed to provide availability of services on an ongoing basis, using redundant computers. If a computer fails, another automatically takes over running the same services and data. Failures of services are due to hardware problems or overload of services on servers. The problem of overloading could be soften using load balancing, is distributing the requests / traffic with other computers in the system, running the same programs, reducing the processing time and improving the performance of the application. We evaluated the methods of implementation of clusters, their models and concepts which the best results to ensure a quality level compatible with the proprietary solutions available. It is shown that the use of cluster is effective in balancing and high availability.

keywords: Cluster, high availability, load balancing

1 - INTRODUÇÃO

1. 1 – Contextualização

Os servidores Web precisam estar operacionais 24 horas por dia, 7 dias por semana. Isso nem sempre é possível. Podem ocorrer falhas de hardware, como por exemplo, se uma placa de rede ou disco rígido queimar, o serviço que está executando naquele servidor ficará indisponível por um período de tempo. Para isso existem soluções de alta disponibilidade que possibilitam tolerância a falhas e que utilizam redundância de componentes (hardware e software). Se um servidor falhar, por qualquer motivo, o outro servidor assume imediatamente. Existem soluções desenvolvidas por grandes empresas, porém o seu custo é proibitivo, sendo inviável para médias e pequenas corporações. Com o advento do software livre é possível criar soluções de alta disponibilidade com baixo custo usando qualquer tipo de computador.

Uma característica de um servidor Web é que ele precisa suportar um grande número de conexões/requisições ao mesmo tempo. Se o servidor não suportar essas requisições começará a criar atrasos para resposta ao usuário, podendo até mesmo não conseguir processar a resposta. Para que isso não aconteça é necessário um hardware robusto, que possui um custo elevado.

Existem atualmente diversas técnicas baseadas em agrupamento de computadores (*clusters*), sendo cada qual voltada para uma área de aplicação, como: soluções para computação de alto desempenho, alta

disponibilidade e balanceamento de carga. A solução de clusters mantém de forma equilibrada o tráfego gerado por determinados serviços, além de redistribuir as requisições enviadas a um determinado nó, para os outros nós do cluster, caso este venha a falhar.

Com a solução de balanceamento de carga, usando software livre, é possível dividir o número de requisições feitas ao servidor entre os computadores que fazem parte do cluster. Assim, nenhum dos servidores será sobrecarregado, já que a carga de requisições é dividida entre os seus nós.

1.2 - Definição do problema

Os problemas com quedas de energia podem danificar componentes de um servidor, causando a interrupção dos serviços. A manutenção pode demorar horas ou até dias até que seja feita a substituição do componente e o sistema fica fora do ar. Outro problema encontrado é o aumento da demanda por serviços, que geram muitas requisições ao servidor o que pode causar sobrecarga e interrupção dos serviços. Esses problemas têm sido uma preocupação eminente dos administradores de redes.

1.3 - Objetivo

O objetivo principal desse trabalho é implantar uma solução de alta disponibilidade e balanceamento de carga para servidores Web com um baixo custo financeiro usando software livre. Esse tipo de solução tem

a vantagem de escalabilidade, que é a facilidade de adicionar novos nós para melhoria do desempenho, à medida que se cresce a carga de trabalho.

1.4 – Organização da monografia

Este texto é organizado em sete capítulos, os quais são listados a seguir:

O Capítulo 1 apresenta uma introdução ao tema discutido e o objetivo do trabalho proposto.

O Capítulo 2 apresenta a fundamentação teórica sobre cluster necessária para o desenvolvimento e compreensão deste trabalho. Uma descrição sobre sistema operacional, tipos de clusters, tipos de armazenamento e softwares usados para alta disponibilidade e balanceamento também são descritos.

No Capítulo 3 é descrita a metodologia utilizada para o desenvolvimento do trabalho, são mostradas formas e técnicas de implementação empregadas.

O Capítulo 4 detalha a configuração dos programas usados para implementação do cluster de alta disponibilidade e alto desempenho.

A descrição dos testes realizados e os resultados obtidos são apresentados no Capítulo 5.

O Capítulo 6 apresenta as conclusões sobre a análise dos resultados.

O Capítulo 7 apresenta a proposta para trabalhos futuros

2 – REFERENCIAL TEÓRICO

Clusters de alta disponibilidade e balanceamento de carga podem ser construídos usando o sistema operacional Linux, ferramentas *open source* e computadores com baixo poder de processamento. Este capítulo apresenta a fundamentação teórica sobre cluster necessária para o desenvolvimento e compreensão deste trabalho. É realizada uma descrição sobre sistema operacional, tipos de clusters, tipos de armazenamento e softwares usados para alta disponibilidade e balanceamento de carga.

2.1 – Sistema operacional

O sistema operacional serve para gerenciar todas as partes de um sistema complexo. Os computadores consistem em processadores, memórias, temporizadores, discos, mouses, interfaces de rede, impressoras e uma ampla variedade de outros dispositivos. O trabalho do sistema operacional é oferecer uma alocação ordenada e controlada dos processadores e dos dispositivos de E/S entre os vários programas que competem por eles [1].

Os sistemas operacionais podem ser proprietários ou livres (código fonte aberto). Entre os Sistemas Operacionais de concepção não proprietária encontra-se o sistema Linux.

Linux

O Linux é um sistema operacional criado a partir do Unix como sendo uma alternativa barata e funcional. Em 1983, Richard Stallman fundou a *Free Software Foundation* (Fundação de Software Livre), cujo projeto GNU, tinha por finalidade criar um clone melhorado e livre do sistema operacional Unix, mas que não utilizasse seu código-fonte.

Linus Benedict Torvalds era aluno da Universidade de Helsinque, na Finlândia, no final da década de 1980. Ele estava disposto a construir um *kernel* (núcleo do sistema) clone do Unix que possuísse memória virtual, multitarefa preemptiva e capacidade de multiusuários [2].

Depois de algum tempo em seu projeto solitário, conseguiu criar um kernel capaz de executar os utilitários de programação e os comandos-padrão do Unix clonados pelo projeto GNU. Reconhecendo que não conseguiria continuar a desenvolver sozinho o Linux, ele distribuiu o código fonte em um lista de discussão de programadores. A partir disso muitos programadores do mundo inteiro tem colaborado e ajudado a fazer do Linux o sistema operacional que é atualmente.

Diversas empresas e organizações de voluntários decidiram juntar os programas do Linux em “pacotes” próprios aos quais elas oferecem suporte. Esses “pacotes” são chamados de distribuições e, entre as mais famosas e utilizadas, destacam-se: Red Hat, Debian, Slackware, Suse, Gentoo e Fedora Core [2].

O Linux oferece diversas vantagens para quem o utiliza, entre elas, destacam-se:

- Sistema multitarefa e multiusuário de 32 ou 64 bits
- Sistema gráfico X-Window
- Suporte a diversas linguagens, como Java, C, C++, Pascal, Lisp, entre outras
- Memória Virtual
- Código Fonte do Kernel
- Estabilidade
- Permissão em arquivos, entre outros.

O sistema operacional Linux pode ser usado para implementação de clusters.

2.2 – História dos clusters

Cluster é um sistema no qual dois ou mais computadores, denominados nós, trabalham conjuntamente para executar aplicações ou realizar tarefas que exijam maior desempenho [4].

A idéia de clusters foi desenvolvida na década de 60 pela IBM como forma de interligar vários mainframes, visando uma forma viável para paralelismo. Nessa época a IBM possuía o sistema HASP (*Houston Automatic Spooling Priority*) e o JES (*Job Entry System*) que possuíam um maneira de distribuir tarefas nos mainframes interligados. [3]

Os clusters na verdade ganharam força depois da convergência de três tecnologias: microprocessadores de alta performance, redes de alta velocidade, e ferramentas padronizadas para computação distribuída de alto desempenho. Uma outra tendência é a crescente necessidade de poder de processamento para aplicações unida ao alto custo e a baixa acessibilidade dos tradicionais supercomputadores [3].

Em 1993, Donald Becker e Thomas Sterling iniciaram um esboço de um sistema de processamento distribuído construído a partir do hardware convencional, de forma a combater o alto custo dos supercomputadores. Em 1994 criaram o projeto baseado nesse tipo de cluster, o Beowulf.

O protótipo inicial era um cluster com 16 processadores ligados por dois canais Ethernet acoplados. O protótipo foi um sucesso e rapidamente a idéia se espalhou pelos meios acadêmicos, pela NASA e por outras comunidades de pesquisa [3].

Tipos de cluster

A combinação de dois ou mais computadores trabalhando juntos para prover um determinado serviço é denominada cluster. Os clusters são utilizados para diversos propósitos, cada um deles requerendo características distintas. Existem as seguintes categorias [5]:

Alta Disponibilidade (*High Availability*): são sistemas feitos para fornecer disponibilidade de serviços de forma ininterrupta, utilizando redundância dos nós. Se um nó falhar, outro assume automaticamente rodando os mesmos serviços e oferecendo os mesmos dados. Esse tipo de cluster é essencial em servidores que executam missões críticas.

Balanceamento de Carga (*Load Balancing*): são sistemas que distribuem as requisições/tráfego com outros nós do sistema, que executam os mesmos programas. Reduz o tempo de processamento e melhora o desempenho da aplicação.

Processamento Distribuído ou Processamento Paralelo (*Parallel Processing*): são sistemas que aumentam significativamente o desempenho para as aplicações, especialmente para grandes tarefas computacionais. Fazem a distribuição das tarefas para diferentes nós do cluster, fazendo assim o processamento paralelo.

Para o usuário, o cluster é um sistema único independentemente da quantidade de nós que o compõe. Todos os aspectos relativos a distribuição de carga, dados, troca de mensagens, sincronização e organização física devem ser transparentes e abstraídos do usuário [5].

Razões para utilização de um Cluster

Os clusters são utilizados quando se tem uma missão crítica ou quando os serviços precisam estar disponíveis e/ou processados o quanto mais rápido possível. O cluster de balanceamento de carga é comumente utilizado em provedores de internet. Clusters paralelos são usados na indústria cinematográfica para renderização de gráficos de alta qualidade. Clusters de alta disponibilidade são indispensáveis para organizações financeiras e outras empresas que precisam sempre ter seus recursos acessíveis [4].

Os benefícios conseguidos com a utilização de cluster são:

Redundância: a adoção de hardware e software redundante evita que qualquer tipo de erro, cause perda das informações ou perda do processamento.

Disponibilidade: um cluster é implementando com dois ou mais nós, se um falhar a aplicação continua executando nos outros nós do sistema.

Distribuição geográfica: o ambiente computacional pode estar distribuído em espaços geográficos distintos. Podem estar localizados em cidades ou regiões diferentes, evitando assim que uma catástrofe, como por exemplo um terremoto, cause a descontinuidade do serviço.

Escalabilidade: é a capacidade de adicionar novos nós ao sistema quando necessário. É possível que sempre que houver uma perda de desempenho, adicione novos nós.

Os clusters podem ser classificados quanto ao seu tipo: alta disponibilidade, alto desempenho e balanceamento de carga.

Alta Disponibilidade

Clusters de Alta Disponibilidade ou *High Availability (HA)* é a técnica de configuração para que dois ou mais computadores trabalhem juntos. Com isso, cada computador gerência os outros e em caso de falhas, assume os serviços que ficaram indisponíveis [6].

Cluster de Alta Disponibilidade é tipicamente construído com a intenção de prover um ambiente seguro contra falhas através de redundância, ou seja, prover um ambiente computacional onde a falha de um ou mais componentes, que pode ser software, hardware ou rede, não

seja significante para afetar a disponibilidade de aplicação em uso. Na figura 2.2.1 é mostrado um cluster de alta disponibilidade.

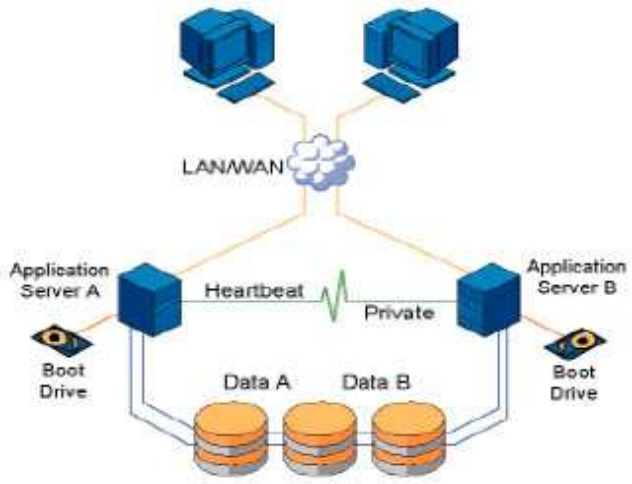

Figura 2.2.1 – Cluster de Alta Disponibilidade [7]

A figura 2.2.1 apresenta um cluster de alta disponibilidade com base de dados compartilhada.

Existem as seguintes classes de disponibilidade: [6]

- Disponibilidade convencional

  É a disponibilidade encontrada em qualquer computador pessoal, sem recursos extras de software ou hardware que esconda suas eventuais falhas. Os computadores dessa classe possuem de 99% a 99,9% de disponibilidade. Em um ano de operação o computador pode ficar indisponível por um período de 9 horas a quatro dias.

- Alta disponibilidade

Encontrada em computadores mais sofisticados com recursos de detecção, recuperação e ocultamento de falhas. Os computadores dessa classe possuem de 99,99% a 99,9996% de disponibilidade. Em um ano de operação o computador pode ficar indisponível por um período de aproximadamente 5 minutos.

- Disponibilidade Contínua

É encontrada em computadores sofisticados com recursos de detecção, recuperação e ocultamento de falhas onde se obtém disponibilidade cada vez mais próxima de 100%, reduzindo o tempo de parada do computador, de forma que venha a ser insignificante ou até mesmo nulo.

Para tolerância a falhas é preciso ter um sistema redundante, essa redundância pode ser hardware ou software. Assim, compreende-se e que a redundância de recursos é um pré-requisito para alcançar alta disponibilidade em um sistema computacional, portanto, a ausência de redundância acarretará em pontos únicos de falha – SPOF (*Single Point of Failure*) [8].

SPOF é qualquer componente de hardware e software do cluster, que no caso de falha de um dos componentes, o sistema fica indisponível, precisando de recuperação automática. Um ponto forte do cluster de alta disponibilidade é o SPOF poder recuperar no menor tempo possível, procurando fazer com que a aplicação permaneça funcional para o usuário final.

Para se entender corretamente a solução de Alta Disponibilidade, deve-se conhecer os conceitos envolvidos. É necessário saber a diferença entre falha, erro e defeito. Parecem três conceitos iguais, mas na verdade designam a ocorrência de algo anormal em três universos diferentes de um sistema computacional [9].

Falha - A falha acontece no meio físico, ou seja, a nível de *hardware*. Um problema na fonte de alimentação ou um interferência eletromagnética é uma falha. Esses são dois eventos indesejados que acontecem no universo físico e afetam a funcionamento de um computador.

Erro - A ocorrência de uma falha pode gerar um erro, que é a representação da falha no universo informal. O sistema do computador trabalha com bits, podendo conter 0 ou 1. Uma falha pode fazer que um ou mais bits troquem seu valor, o que certamente acarretará em um funcionamento anormal do sistema. Uma falha pode gerar um erro em alguma informação.

Defeito - O defeito pode ocorrer quando o erro não é percebido e tratado. O sistema simplesmente trava, exibindo uma mensagem de erro ou perde os dados dos usuários sem maiores avisos. Isso é percebido no universo do usuário.

O termo tolerância a falhas tenta tratar as falhas enquanto ainda são erros. Para que as máquinas continuem funcionando, mesmo quando exista um problema.

Para que uma máquina assuma o lugar da outra é necessário saber onde houve a falha. Isso pode ser feito por testes periódicos, onde o servidor secundário testa se a outra máquina está ativa além de testar se ela está fornecendo respostas adequadas as requisições de serviços.

Failover - O processo que uma maquina assume o lugar da outra, quando a primeira apresenta falhas, é chamado de *failover*. Além do tempo entre a falha e a sua detecção, existe também o tempo entre a detecção e o restabelecimento do serviço.

É de extrema importância que uma solução de alta disponibilidade mantenha recursos redundantes com o mesmo estado, de maneira que o serviço possa ser retomado sem perdas. Dependendo 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.

Failback - Após a falha do servidor primário, o servidor secundário irá assumir seu lugar e será necessário que seja feito a manutenção do servidor com problemas. 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 o processo de retorno de um determinado serviço de uma outra máquina para sua máquina de origem [9].

Tipos de clusters de alto disponibilidade

Existem dois tipos básicos de clusters de alto disponibilidade, os ativo-passivo (simétrico) e o ativo-ativo (assimétrico).

Cluster de alta disponibilidade ativo-passivo se baseiam no conceito de servidor primário e secundário. Servidores que estão provendo algum tipo de serviço são considerados como primários. Servidores que aguardam a ocorrência de alguma falha no servidor primário são denominados servidores secundários. A detecção da falha ocorre através do monitoramento dos servidores primários por parte dos secundários.

Nesse tipo de configuração, a memória e o processador do cluster passivo tendem a ser aproveitados para melhorar a performance das transações no cluster ativo [10]. A figura 2.2.2 apresenta o modelo de cluster ativo-passivo.

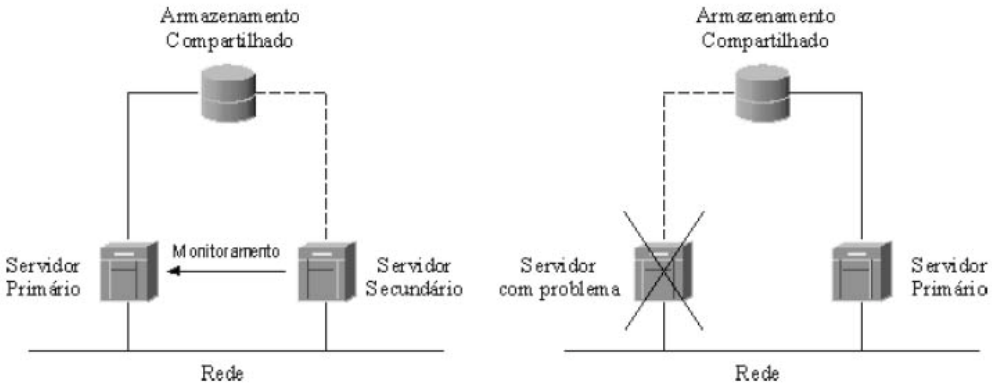

Figura 2.2.2 - Cluster de alta disponibilidade ativo-passivo [10].

No cluster de alta disponibilidade ativo-ativo não há o papel de um servidor inativo aguardando a falha de outro, para assumir seus serviços. Neste modelo, ambos servidores servem alguma aplicação. Assim, os termos primário e secundário deixam de ser usados para designar o servidor, passando a ser usado para a aplicação.

Na ocorrência de falha em um dos servidores, dá-se início ao processo de *failover*. Neste caso, o servidor ainda ativo assume o controle da aplicação do servidor problemático. O monitoramento deverá ocorrer entre todos os nós. A figura 2.2.3 apresenta o modelo de cluster ativo-ativo.

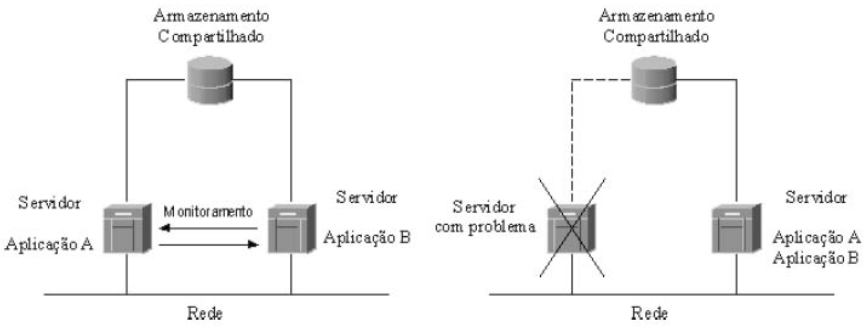

Figura 2.2.3 - Cluster de alta disponibilidade ativo-ativo [10].

Alto desempenho

Esse tipo de cluster é usado quando se precisa de um grande poder de processamento na execução de atividades complexas. Uma grande

tarefa computacional pode ser dividida em tarefas menores e então distribuída através dos nós de um cluster. Esse método é denominado processamento distribuído ou processamento paralelo, reduzindo dessa forma o tempo gasto na execução total da tarefa. O processamento paralelo caracteriza-se pela distribuição dos processos para mais de uma CPU e, o processamento distribuído, caracteriza-se pelo compartilhamento dos recursos de computadores otimizando dessa forma uma tarefa computacional [11].

Uma tarefa inicia a execução em um nó do cluster, enquanto ocorre o processamento, os nós ficam comunicando-se entre si. Esta tarefa é dividida em pequeno pedaços com os outros nós. Por exemplo, a tarefa do nó A está realizando um processamento, cujo resultado será utilizado por uma outra tarefa do nó B. Por sua vez, a tarefa do nó B está aguardando o resultado do término da execução da tarefa do nó A para que possa prosseguir com sua execução, ou seja, os resultados intermediários de um nó afetam resultados de tarefas futuras de outro nó. Quando os nós finalizam a execução de sua tarefa, os resultados da tarefa do processo dividido são remontados e a informação é disponibilizada como um resultado único de processamento [7].

Um cluster de alto desempenho implementa a comunicação entre os nós de duas maneiras: *Parallel Virtual Machine* (PVM) e *Messaging Program Interface* (MPI). Essas são as bibliotecas de programação para sistemas paralelos que compartilham recursos de memória. Estas bibliotecas possuem rotinas para iniciar e configurar o ambiente de mensagem, assim como, o recebimento e envio de pacotes de dados.

O PVM pode ser considerado tanto como um ambiente quanto uma biblioteca de passagem de mensagem, que pode ser usado para executar aplicações paralelas em sistemas de supercomputadores através de clusters de estações de trabalho. O MPI é uma especificação para passagem de mensagem designada a ser um padrão para computação paralela distribuída usando explicitamente passagem de mensagem. Esta interface é uma tentativa para estabelecer uma prática portável, eficiente e flexível de padronização para passagem de mensagem [13].

Este tipo de cluster está relacionado com aplicações industriais, científicas, de engenharia e com aplicações voltadas para negócios com banco de dados que precisam recuperar e analisar grandes quantidades de dados. Entre as aplicações que fazem uso dessa tecnologia estão: previsão do tempo; estudos geológicos para a indústria do petróleo; processamento e renderização de imagens; estudos científicos em geral [7].

O cluster Beowulf, desenvolvido pela NASA, é um dos mais populares projetos de clusters de alto desempenho. É um sistema que normalmente consiste em um nó servidor e um ou mais nós clientes conectados via rede E*thernet* o outro dispositivo de conexão como o *switch*. Foi construído utilizando o sistema operacional linux, as bibliotecas MPI e PVM, ferramentas de gerenciamento e *hardware* simples. São usados computadores compatíveis com arquitetura Intel x86, AMD, Compaq Alpha e IBM, com pouca capacidade de processamento, memória e disco rígido. Com isso tem um baixo custo por nó comparado ao custo de soluções com supercomputadores [12].

As tarefas do cluster Beowulf são iniciadas no nó servidor e sua execução é distribuída em paralelo entre os demais membros do cluster

para os nós clientes, reduzindo o tempo de processamento da tarefa. Os nós clientes são controlados pelo nó servidor e, ao final da execução da tarefa, é retornado o resultado acumulado do processamento, similar aos clusters de alto desempenho.

É possível adicionar novos computadores sempre que necessário e, normalmente, os nós do cluster são configurados de forma idêntica, com mesma quantidade de memória, rede e discos. Não é pré-requisito para o seu funcionamento que os nós sejam iguais, pelo contrário, a adição de novos nós com diferentes configurações pode ser feitas, permitindo um melhor aproveitamento dos recursos computacionais [14].

O projeto Beowulf além das características comuns a um cluster, deve possuir as seguintes características [12]:

· Nenhum componente feito sob encomenda;

· Independência de fornecedores de hardware e software;

· Periféricos escaláveis;

· Software livre de código aberto;

· Uso de ferramentas de computação distribuída disponível livremente com alterações mínimas;

· Retorno à comunidade do projeto e melhorias.

A figura 2.2.4 representa um cluster de alto desempenho com vinte e quatro máquinas.

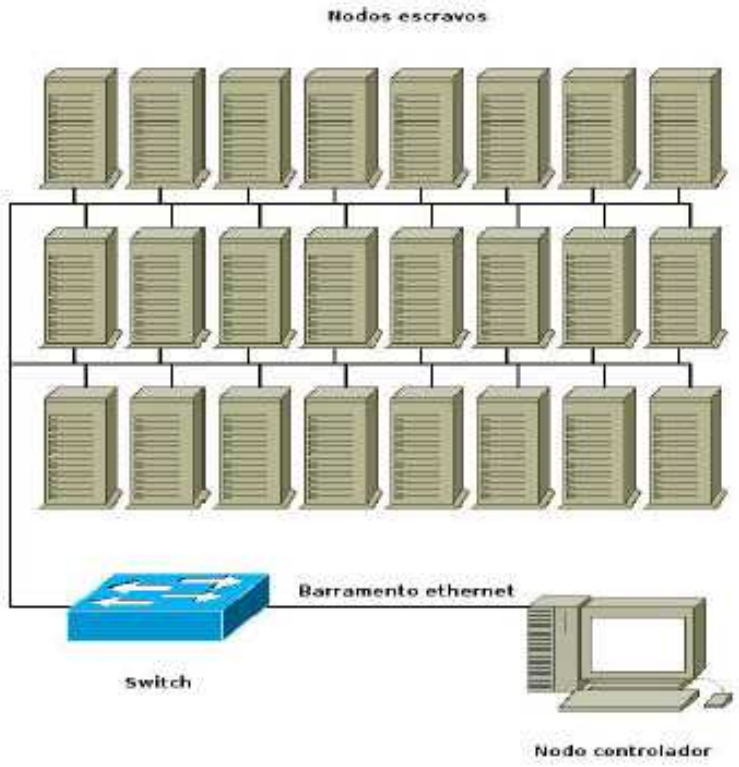

Figura 2.2.4 – Cluster de Alto Desempenho [7]

Balanceamento de Carga

Cluster de balanceamento de carga é, na maioria das vezes, uma solução híbrida porque possui características do cluster de alta disponibilidade e as características próprias do cluster de balanceamento de carga, que melhora o desempenho e a escalabilidade do serviço [15].

Esse cluster consiste em dividir as requisições ou processos que chegam entre os computadores pertencentes ao sistema, para que não haja

sobrecarga de informações, ou seja, fazer com que o computador não fique muito atarefado prejudicando assim o desempenho do sistema.

O cluster de balanceamento de cargas também segue o paradigma que, para o usuário, este balanceamento será totalmente transparente. Essa transparência pode ser feita através de um software específico ou por um simples redirecionamento de DNS (*Domain Name Service*). Um servidor de DNS pode ter configurado diversos endereços respondendo por um determinado nome de domínio, assim, os requisitantes do endereço acabam resolvendo o nome com endereços IP diferentes e realizando conexões com diversos nós do cluster.

Quando se colocam dois computadores com a mesma capacidade de resposta para atender uma mesma requisição, sem fazer o balanceamento de cargas, pode acontecer de os dois computadores responderem a mesma requisição, prejudicando assim a conexão. Para evitar isso, coloca-se um software de balanceamento de carga que fará o gerenciamento entre os servidores e o cliente [16].

O cluster de balanceamento de carga, apesar de ter como objetivo principal o aumento do desempenho, difere do modelo de cluster de alto desempenho, pois as requisições dos clientes são atendidas completamente pelo nó ao qual a requisição foi direcionada. O objetivo neste caso é dividir as requisições e não as sub-tarefas nelas envolvidas

Os servidores não precisam possuir a mesma configuração, isso os torna mais flexíveis. Devido a essa flexibilidade, os serviços destinados aos nós podem ser diferentes. Enquanto um ou mais nós podem ser destinados a requisições de processos pesados, os outros recebem requisições somente de consulta e de curta duração. Se necessário pode-se

fazer o processamento distribuído uniformente entre os nós do cluster usando o sistema *Linux Virtual Server* (LVS).

A figura 2.2.5 apresenta um modelo de cluster de balanceamento de carga.

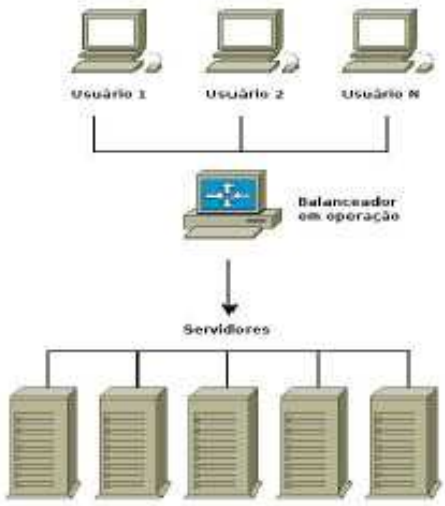

Figura 2.2.5 - Cluster de balanceamento de carga [9]

A figura 2.2.5 apresenta três usuários fazendo requisições a um computador, chamado balanceador em operação, que redireciona as requisições para os servidores de acordo com o algoritmo pré-definido.

Algoritmos de Balanceamento de carga

Existem oito principais algoritmos para balanceamento de carga: *Round Robin, Weighted Round-Robin, Least Connection, Weighted Least Connection, Locality Based Least Connection, Locality Based Least Connection with replication, Destination Hashing 1, Destination Hashing2.*

Round Robin - As requisições são distribuídas uniformemente entre os membros do cluster, eles recebem um tempo para cada processo, em parte iguais, e possuem a mesma prioridade.

Os processos são armazenados em fila circular. O escalonador da CPU percorre a fila alocando para a CPU para cada processo durante um *quantum* (unidade de tempo), e retira o primeiro processo da fila procedendo sua execução. Se o processo não termina no quantum determinado ele é inserido no final da fila e se termina antes o quantum é liberado para outro processo [1].

Weighted Round-Robin - É uma variante do algoritmo round robin. Ele determina um peso para cada servidor. O servidor que possuir maiores recursos computacionais recebe um peso maior e assim mais requisições serão redirecionadas para ele [1].

Least Connection - Esse algoritmo analisa qual servidor está com menos conexões ativas e distribui as novas requisições tomando isso como parâmetro. É indicado quando as maquinas tem configuração similar [17].

Weighted Least Connection - Similar ao Least Connection, com a diferença que adiciona pesos para os servidores. As requisições são distribuídas para o servidor com maior peso e o menor numero de requisições [17].

Locality Based Least Connection - Este algoritmo atribui as requisições com o mesmo endereço de destino sempre ao mesmo servidor, a não ser que ele esteja sobrecarregado ou indisponível no momento. Se ele estiver sobrecarregado ou indisponível o trabalho é atribuído a outro servidor com menos conexões [17].

Locality Based Least Connection with replication - Atribui as requisições ao servidor que tem o menor número de conexões dentre o conjunto que está lidando com certo endereço de destino. Caso todos estejam superlotados, um nó com menos requisições é escolhido e adicionado ao conjunto. Se o servidor não foi modificado em um determinado período de tempo o nó mais ocupado é removido do conjunto para evitar um alto grau de replicação [17].

Destination Hashing 1 - O servidor de destino é escolhido mediante uma tabela de hashing baseada no endereço de destino [18].

Destination Hashing 2 - O servidor de destino é escolhido mediante uma tabela de hashing baseada no endereço de origem [18].

Os servidores dos clusters acessam a base de dados que podem ter diferentes implementações de armazenamento.

2.3 – Tipos de armazenamento

Dependendo da escala, performance, funcionalidade e flexibilidade do cluster, são necessárias diferentes implementações de armazenamento. Os principais métodos de armazenamento são o compartilhamento total, sem compartilhamento e espelhamento de disco.

No compartilhamento total, todos os servidores acessam o mesmo meio de armazenamento. Este modelo apresenta potenciais problemas de corrupção de dados quando mais de um servidor está ativo [19].

A figura 2.3.1 apresenta o modelo de compartilhamento total da unidade de armazenamento.

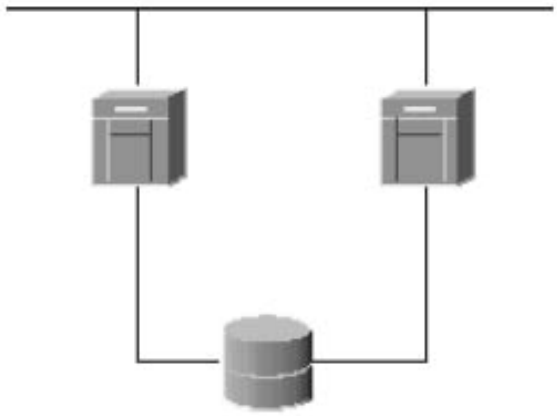

Figura 2.3.1 - Compartilhamento total da unidade de armazenamento [10].

No armazenamento sem compartilhamento, dois ou mais servidores possuem seu próprio meio de armazenamento. No evento de uma falha em um dos servidores, o servidor responsável pela substituição passa a ter acesso total ao meio de armazenamento original do nó defeituoso [19].

A figura 2.3.2 apresenta o modelo da unidade de armazenamento sem compartilhamento.

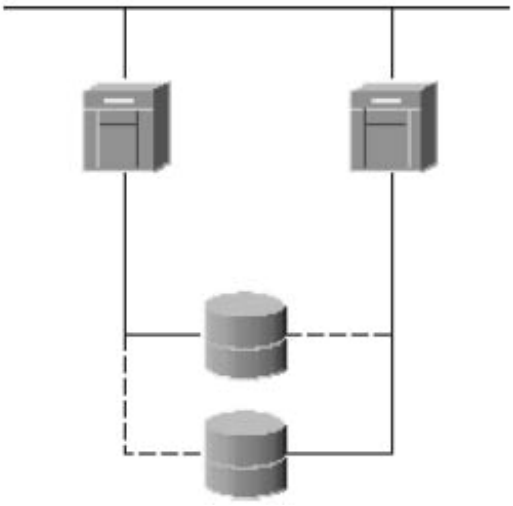

Figura 2.3.2 – Unidade de armazenamento sem compartilhamento [10]

Em um ambiente de espelhamento de discos, não há nenhum tipo de compartilhamento entre os nós servidores. Através do uso de software, os dados são espelhados ou replicados de um servidor para o outro através da rede. O princípio deste modelo é que todos servidores secundários devem possuir seu próprio meio de armazenamento e, ao mesmo tempo, uma réplica do armazenamento do servidor a ser substituído. Neste caso, será necessário dois ou mais dispositivos, caracterizando-se um array de discos [19].

A figura 2.3.3 apresenta o modelo espelhamento de discos.

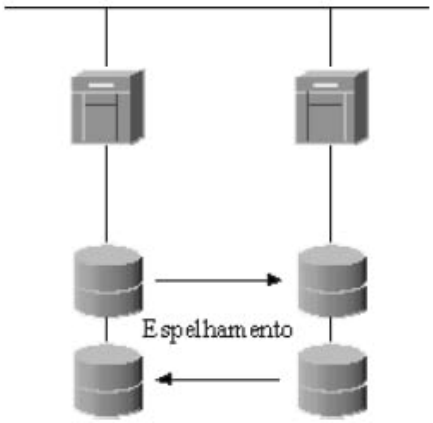

Figura 2.3.3 – Espelhamento de disco [10]

Os arrays de discos são conhecidos como RAID (*Redundant Array os Independent Disks*) e são usados, para melhorar o desempenho e a confiabilidade de um sistema de armazenamento, armazenando dados redundantes [20].

Existem diferentes tipos de RAID, cada um com seu método para manipular os dados que são armazenados no disco: RAID 0, RAID 1, RAID 2, RAID 3, RAID 4, RAID 5, RAID 6, RAID 0 + 1, RAID 1 + 0, RAID 50 e RAID 100. Os métodos relacionados a cluster são: RAID 0, RAID 1 e RAID 5.

RAID-0: Os níveis de RAID envolvem uma técnica de armazenamento chamada de segmentação de dados (data stripping). Os benefícios do RAID-0 estão no aumento do número máximo de operações de leitura e escrita por segundo e aumento na taxa de transferência de dados, sendo recomendável para aplicações que necessitam alta performance, mas não tem necessidade de alta disponibilidade nas informações armazenadas [21].

A figura 2.3.4 apresenta o modelo de RAID 0.

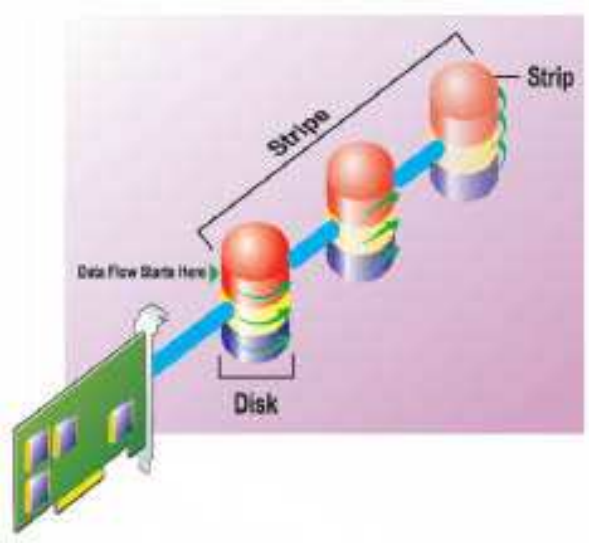

Figura 2.3.4 - RAID 0 [22]

RAID-1: É a forma mais simples de arranjo tolerante a falhas. Este nível é baseado no conceito de espelhamento. Este arranjo consiste de vários grupos de dados armazenados em 2 ou mais dispositivos. Entretanto, somente 50% da capacidade do drive neste caso estará disponível para armazenamento. Se ocorrer uma falha em um disco de arranjo RAID-1 leituras e gravações serão direcionados para o(s) disco(s) ainda em operação. Os dados então são reconstruídos em um disco de reposição, usando dados do(s) disco(s) sobreviventes. O processo de reconstrução do espelho apresenta um impacto sobre a performance de E/S(entrada e saída) do arranjo, pois todos os dados terão de ser lidos e copiados do(s) disco(s) intactos(s) para o disco de reposição [21].

A figura 2.3.5 apresenta o modelo RAID 1.

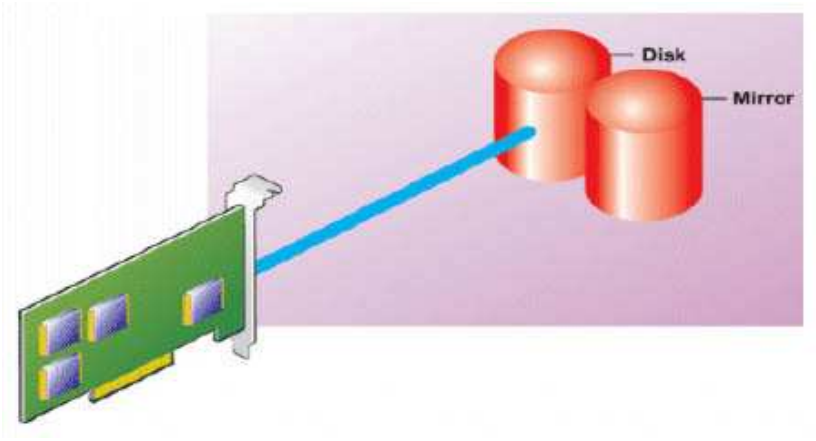

Figura 2.3.5 - RAID 1 [22]

RAID-5: Os dados do arranjo são distribuídos ao longo de todos os discos, ao invés de serem armazenados em um disco dedicado. Essa idéia reduz o gargalo de escrita que era feito em um único disco, isto acontece pelo fato das escritas concorrentes nem sempre requisitarem um acesso a informações em um disco dedicado. Contudo, a performance de escrita em geral ainda sofre por causa do processamento adicional causado pela leitura das informações [21].

A figura 2.3.6 apresenta o modelo RAID 5.

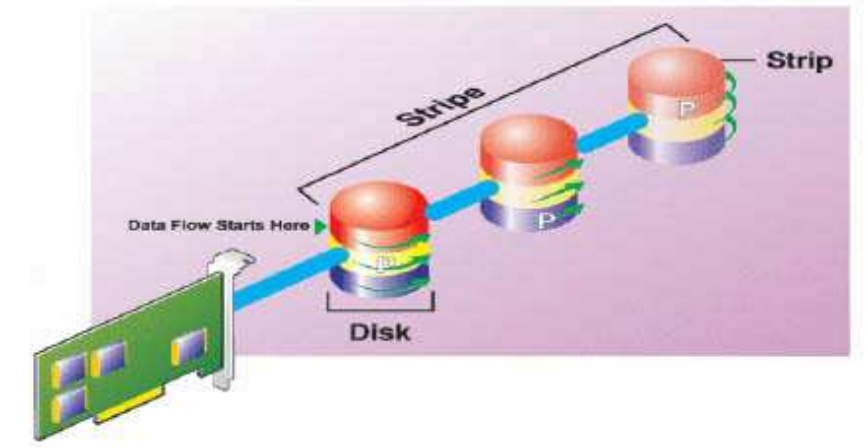

Figura 2.3.6 - RAID 5 [22]

Devido às características de desempenho e redundância encontradas em cada nível de RAID, o método mais adequado dependerá da aplicação envolvida. A tabela 3.1 apresenta as principais características e mostra a comparação entre eles:

Tabela 2.3.1 – Principais característica e comparação entre RAID

Nível de RAID	Disponibilidade do dado	Desempenho de leitura	Desempenho de gravação	Desempenho na reconstrução dos dados	Quantidade mínima de discos	Recomendação de uso
0	Baixa	Muito bom	Muito bom	-	N	Dados não críticos
1	Excelente	Muito bom	Bom	Bom	2 (N=1)	Pequenas bases de dados, *logs*, dados críticos
5	Boa	Leitura seqüencial: Boa/Leitura transacional: Muito boa	Razoável	Ruim	N+1 (N>=2)	Bases de dados e aplicações com alta intensidade de leitura

N = Número de discos no *array*
Y = Número de conjuntos *RAID*

Definido o modelo de implementação da unidade de armazenamento é preciso conhecer os softwares disponíveis para implementação do cluster de alta disponibilidade e balanceamento de carga.

2.4 – Softwares para implementação do cluster

Existem várias soluções para criação de um cluster. Uma solução possível para um cluster de alta disponibilidade e balanceamento de carga é baseada em quatro sistemas básicos: replicação/sincronização de discos (executados pelos softwares DRDB ou Rsync), monitoramento de nós (executado pelo software Heartbeat), balanceamento de carga (executado pelo software Linux Virtual Server – LVS) e Front-end para acesso ao LVS (executado pelo software Piranha).

DRDB (*Data Replicator Block Device*)

O DRDB é um sistema de replicação desenvolvido para Cluster de Alta Disponibilidade. A sua função é criar um espelho (imagem) de um dispositivo a outro utilizando a rede como meio de transmissão.

Os dispositivos DRDB funcionam em dois estados: primário e secundário. Ele toma conta do dados escritos no disco primário e os envia para o secundário. Esse processo recebe o nome de sincronização. Ë possível criar vários espelhamentos, sendo limitado somente pela largura de banda da rede [23].

Heartbeat

A principal finalidade do Heartbeat é de ser um monitor do estado do sistema e dependendo do resultado encontrado, tomar uma decisão. Heartbeat significa batimento cardíaco. Esse termo é usado para definir os pacotes enviados entre dois computadores que indicam que estão vivos, ou seja, estão funcionando e disponíveis para executar tarefas. Esses pacotes podem ser enviados por uma placa de rede ou uma porta serial. Se o pacote falhar, o computador secundário irá assumir que o computador primário falhou e tomar os serviços que estavam rodando no computador primário [23].

Linux Virtual Server (LVS)

O projeto Linux Virtual Server (LVS) fornece os softwares necessários para montar um servidor virtual e facilmente escalável em um cluster. O LVS assume o IP principal do cluster e passa a responder por ele perante as requisições dos clientes. O usuário faz a requisição através da internet ao cluster de balanceamento de carga. Quando chega a requisição ele se encarregara de encaminhar a requisição ao nó que terá melhor condição de atendimento (dependendo do algoritmo escolhido) [13].

Piranha

Piranha é um produto para clustering da RedHat, e inclui o código do LVS para o kernel, uma ferramenta gráfica para configuração do cluster e uma ferramenta de monitoração do cluster. É possível configurar

os servidores que fazem parte do cluster, qual tipo de roteamento será usado, qual serviço é monitorado entre outras configurações [4].

2.5 – Considerações finais

Neste capítulo foi apresentado a fundamentação teórica sobre cluster. No próximo capítulo será apresentada a metodologia.

3 - METODOLOGIA

Este capítulo detalha a implantação de um cluster de alta disponibilidade e de balanceamento de carga. Serão apresentados os programas utilizados e o ambiente de teste.

A classificação deste trabalho quanto à natureza é considerado uma pesquisa tecnológica. Quanto ao objetivo é uma pesquisa descritiva, que tem a finalidade de observar, registrar e analisar sistemas técnicos. Em relação aos procedimentos é uma pesquisa experimental e será uma pesquisa em laboratório, onde é possível controlar as variáveis que compõem o ambiente, ou minimizar sua interferência no ambiente [24].

O desenvolvimento desse projeto foi realizado no laboratório do Centro de Informática da Universidade Federal de Lavras, denominado CIN-UFLA.

Com a implantação do cluster pretende-se proporcionar maior confiabilidade e desempenho ao serviço Web, já que dessa forma, temos um sistema tolerante a falhas e que distribui o tráfego.

Devido ao servidor de páginas da Universidade (UFLA) possuir um grande volume de requisições diariamente, ele pode não suportar o tráfego e ficar muito lento ou até mesmo travar. Pode acontecer também de um componente queimar e ser necessário parar o servidor durante dias para a troca do componente.

3.1 - Solução Proposta

Todo o tráfego relativo ao servidor Web passará antes por uma máquina definida como Diretor. O diretor tem por objetivo receber as requisições de entrada e reencaminhá-las ao servidor disponível. A figura 3.1 mostra a estrutura do cluster e o papel do diretor.

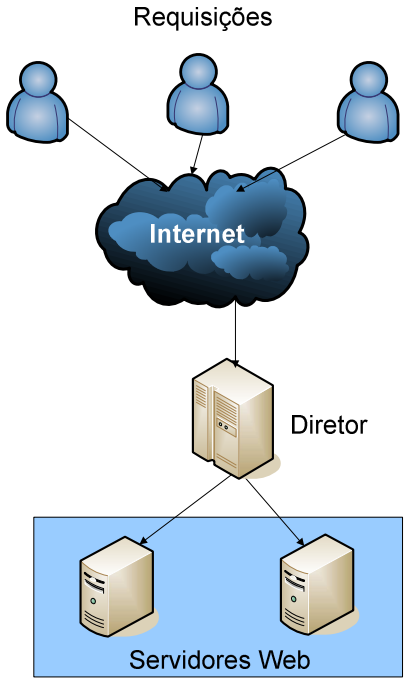

Figura 3.1: Estrutura de um cluster

O projeto inicial de proporcionar balanceamento de carga e alta disponibilidade era de utilizar a configuração em três camadas, ou seja, o diretor na primeira camada, os servidores reais na segunda camada e a base de dados na terceira camada. Nesse sistema os servidores reais acessam a fonte de dados compartilhados conforme mostra figura 3.2:

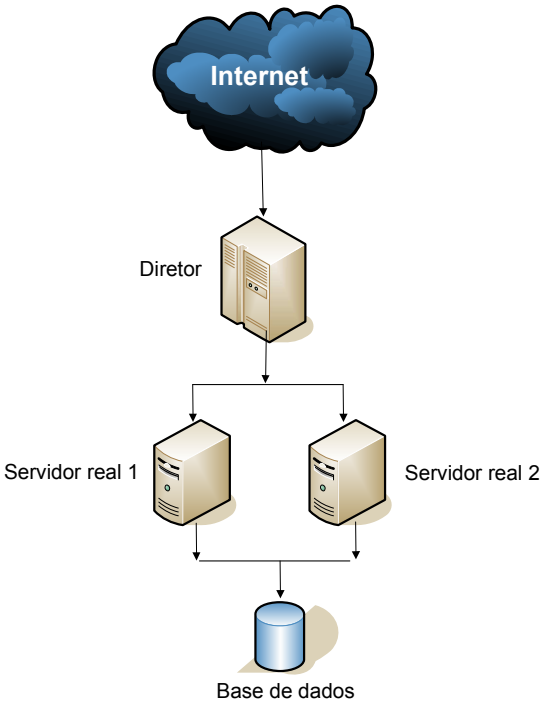

Figura 3.2 – Cluster em três camadas

Nessa configuração é necessária uma máquina exclusiva para alocar a base de dados. Para otimizar recursos foi decidido que cada servidor real possuiria sua própria base de dados. Dessa forma foi usada a configuração em duas camadas, ou seja, o diretor na primeira camada e os servidores reais e a base de dados na segunda camada. Conforme mostra a figura 3.3 abaixo:

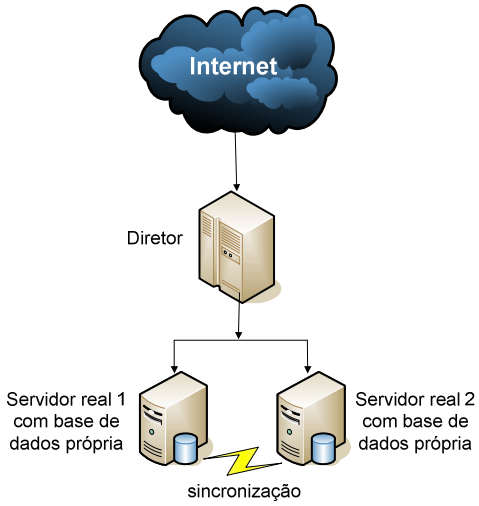

Figura 3.3 – Cluster em duas camadas

É necessário que os servidores reais sincronizem de forma confiável e otimizada os arquivos para que não possuam informações diferentes em sua base de dados. Por exemplo, o usuário faz alterações em seus arquivos no servidor 2 e termina a conexão. Logo após faz uma nova requisição e o diretor o direciona para o servidor real 1. Caso não tenha sido feita a sincronização ele estará acessando os arquivos desatualizados.

As requisições que chegam ao diretor são enviadas a um IP virtual (VIP). Quando chegar uma requisição ao servidor Web, essa é distribuída aos nós do cluster. Um IP Virtual pode estar configurado no mesmo dispositivo que conecta o diretor à Internet. Por exemplo, o servidor se conecta à Internet pela interface real eth0 e cria-se o ip virtual como eth0:1.

O diretor irá redirecionar as requisições dos endereços de IP Virtual para os servidores verdadeiros de acordo com o algoritmo de balanceamento de carga. Se o serviço de um dos servidores reais parar, o

diretor deixa de enviar as requisições para ele até que o serviço se restabeleça.

Para fazer o balanceamento o diretor dispõe de oito algoritmos principais. Esses algoritmos podem escolher o servidor que irá receber a requisição aleatoriamente, escolher o servidor por capacidade de processamento ou escolher o servidor por número de requisições que estão sendo processadas no momento.

Neste trabalho, foi escolhido o algoritmo *Weighted Least-Connections* que possibilita adicionar pesos aos servidores de acordo com sua capacidade de processamento. As requisições são distribuídas para o servidor com maior peso e o menor número de requisições. Por exemplo: servidor X tem peso 3 e o servidor Y tem peso 1, o servidor X recebe 3 conexões a cada 1 conexão do Y. Usando esse método pode ocorrer desequilíbrios momentâneos, como por exemplo, os servidores X e Y têm peso 1 e o terceiro, servidor Z, tem peso 3. Se o servidor Z parar de responder, os servidores X e Y distribuem uniformemente a carga abandonada. Quando o servidor Z estiver online, o diretor analisa que ele tem zero conexões e o inunda com todos os pedidos recebidos até que ele esteja igual aos servidores X e Y.

É necessário escolher o método de roteamento com que as requisições são devolvidas aos solicitantes. Existem dois métodos: o roteamento com NAT e o roteamento direto.

Roteamento com NAT

Usando o roteamento com NAT, o diretor recebe a requisição e envia para o servidor apropriado. O servidor real processa o pedido e devolve os pacotes para o diretor que utiliza o NAT para substituir o endereço do servidor real nos pacotes para o IP do diretor. O endereço de IP dos servidores reais é escondido dos clientes solicitantes. A figura 3.4 mostra o roteamento com NAT.

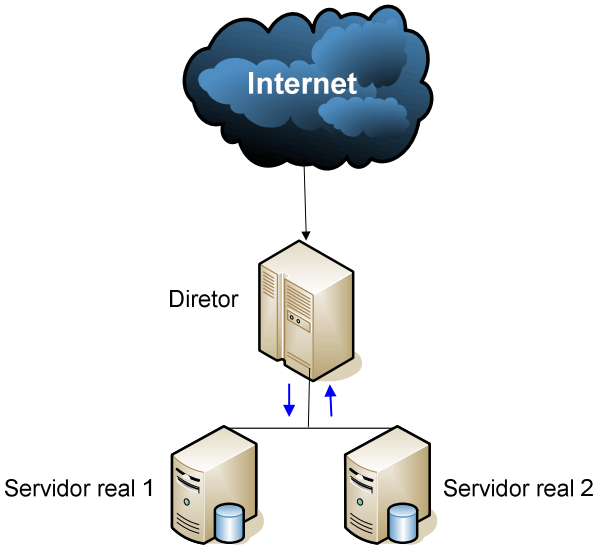

Figura 3.4 – Roteamento com NAT

Os servidores reais podem ser qualquer tipo de máquina executando qualquer sistema operacional. A principal desvantagem é que o diretor pode se tornar um gargalo quando tem um grande número de respostas dos servidores reais.

Roteamento Direto

Esse método permite que os servidores reais processarem e responderem as requisições diretamente para um usuário, ao invés de passar pelo diretor. Assim evita o gargalo das respostas das requisições no diretor. A figura 3.5 mostra o roteamento direto.

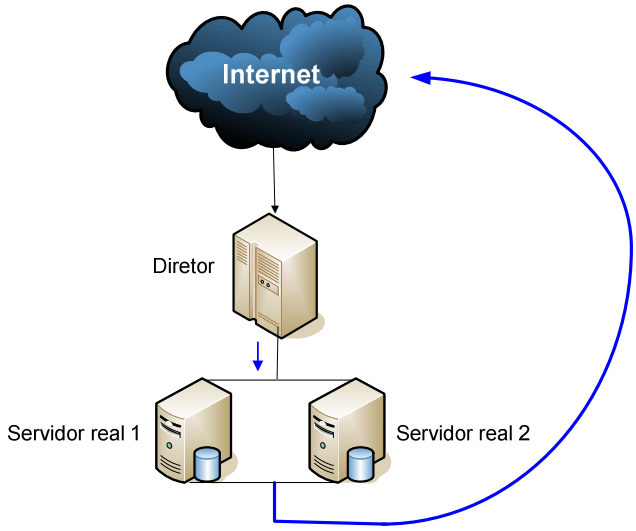

Figura 3.5 – Roteamento direto

Embora haja vantagens de utilizar o roteamento direto, pode também acontecer problemas, como o IP Virtual ser associado diretamente a um dos verdadeiros servidores e as requisições serem tratadas diretamente pelo servidor real, ignorando completamente o serviço do Diretor. Isso pode ignorar o balanceamento de carga.

Para resolver esse problema e garantir que os pedidos recebidos são sempre enviados para o Diretor ao invés de um dos servidores reais, pode se usar os aplicativos `arptables` ou o `iptables` na filtragem de pacotes. Como temos um grande número de requisições e consequentemente um grande número de respostas foi escolhido neste trabalho o método de Roteamento direto para um melhor desempenho.

Após escolhido o método de roteamento é necessário escolher como será feita a sincronização dos dados dos servidores reais. O software `DRBD` funciona muito bem para cluster de alta disponibilidade,

pois ele cria um dispositivo e o associa a uma partição do sistema em cada nó do cluster. O nó marcado como primário é responsável por propagar os dados para os outros nós definidos como secundário. O problema é que os dados ficam inacessíveis em nós secundários, até que esse nó se torne primário. No balanceamento de carga a cada momento os dados são acessados em um local diferente, quando acessar os dados em um nó secundário teria problemas.

Para usar a configuração de todos os nós como primário, seria necessário mudar o sistema de arquivos do servidor para GFS (*Global FileSystem*). O GFS é um sistema de arquivos criado para manter a consistência dos dados. Usa um esquema de trava controlado pelo dispositivo de armazenamento, que implementa o conjunto de operações *read-modify-write*.

O Rsync é um programa simples para cópia de arquivos, porém seu algoritmo realiza o particionamento do arquivo de origem em vários pedaços e gera uma pequena assinatura para cada um. Depois transmite pela rede as assinaturas e, fazendo o mesmo do outro lado, descobre quais pedaços faltam no arquivo de destino para torná-lo igual ao de origem.

O Rsync deve ser usado juntamente com um agendador de tarefas que permita executar tarefas repetidamente, ou seja, toda semana, todo dia, toda hora, ou em qualquer horário pré estabelecido. Para atender todos os requisitos considerados acima foi escolhido o Cron que é um agendador de tarefas.

Resumidamente, o cluster de alta disponibilidade e balanceamento de carga definido neste trabalho consiste de:

Cluster em duas camadas

Algoritmo de balanceamento Weighted Least – Connection

Método de Roteamento direto

Sincronização de dados dos servidores reais, utilizando Rsync e agendador de tarefas Cron.

3.2 – Considerações finais

Neste capítulo foi apresentado o modelo de camadas, as técnicas de roteamento e os programas de sincronização. Foi definido a utilização do modelo de duas camadas, com método de roteamento direto e o aplicativo Rsync para sincronização dos dados juntamente com o Cron

No próximo capítulo serão apresentados detalhes da implementação do cluster de alta disponibilidade e balanceamento de carga.

4 - IMPLEMENTAÇÃO

Este capítulo detalha a configuração dos programas usados para implementação do cluster de alta disponibilidade e alto desempenho.

4.1 – Ambiente configurado

O cluster será formado por três computadores, o Diretor e dois servidores Web.

A descrição dos serviços implantados nos computadores e suas características técnicas são:

*** Diretor (Servidor de Monitoramento de Atividades)** – Responsável por receber as requisições e distribuí-las aos servidores e monitorar se algum dos nós não esta respondendo. Essa maquina tem a seguinte configuração:

Processador: 1 dual core

HD: 260 GB Memória: 2 GB

IP: 200.131.250.20

VIP (IP Virtual): 200.131.250.19

*** Servidor 1** – Contém a instalação do servidor Web.

Processador: 2 quad core

HD: 1,5 TB Memória: 8 GB

IP: 200.131.250.28

VIP (IP Virtual): 200.131.250.19

*** Servidor 2** – Contém a instalação do servidor Web.

Processador: 2 quad core

HD: 900 GB Memória: 4 GB

IP: 200.131.250.29

VIP (IP Virtual): 200.131.250.19

A figura 4.1 apresenta o cenário do ambiente de testes.

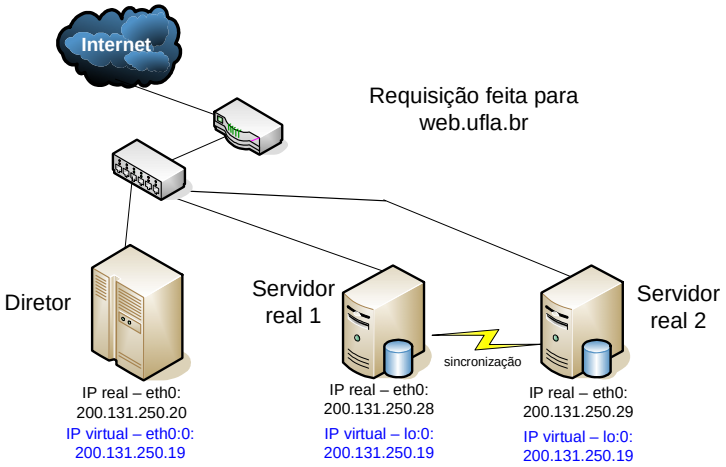

Figura 4.1 – Cenário do ambiente de testes

A solução foi implantada usando software livre, sistema operacional Linux, distribuição Fedora Core 10, Linux Virtual Server (LVS), Heartbeat, Piranha (front end) e o Rsync.

No Diretor está instalado o sistema operacional Linux, distribuição Fedora Core 10 com servidor Apache rodando na porta 80.

Primeiramente foi instalado o ipvsadm e o front-end Piranha.

Depois de instalado é preciso definir uma senha para acesso via navegador para o piranha. Se a senha for alterada durante uma sessão ativa do piranha, o administrador deverá logar novamente.

A senha foi alterada com o comando `piranha-passwd`. Depois de iniciado o serviço é preciso alterar no arquivo de configuração do Apache quais IPs podem acessar a interface do Piranha via navegador.

Para que o Diretor possa transmitir os pacotes adequadamente pela rede para os servidores reais, deve-se ativar o encaminhamento de pacotes. Foi alterado a linha net.ipv4.ip_forward = 0 no arquivo /etc/sysctl.conf para net.ipv4.ip_forward = 1, sendo preciso reiniciar o diretor.

Para assegurar que os servidores reais não respondam diretamente as requisições dos clientes foi instalado no servidor real 1 e no servidor real 2, o pacote arptables_jf usando o comando *yum*. Após instalado o arptables foram executados os seguintes comandos:

Servidor 1

```
arptables -A IN -d 200.131.250.19 -j DROP
arptables -A OUT -d 200.131.250.19 -j mangle --mangle-ip-s 200.131.250.28
```

Servidor 2

```
arptables -A IN -d 200.131.250.19 -j DROP
arptables -A OUT -d 200.131.250.19 -j mangle --mangle-ip-s 200.131.250.29
```

Após os comandos, a tabela ARP foi salva e o comando `chkconfig` foi usado para que o sistema carreguasse a configuração arptables na inicialização.

No arquivo /etc/sysctl.conf é necessário inserir as linhas abaixo:

```
net.ipv4.conf.lo.arp_ignore = 1
```

```
net.ipv4.conf.lo.arp_announce = 2
net.ipv4.conf.all.arp_ignore = 1
net.ipv4.conf.all.arp_announce = 2
```

Devemos configurar o IP virtual na interface de rede. Usamos o comando `ifconfig eth0:0 200.131.250.19` e máscara 255.255.255.255. Essa máscara de 32 bits significa que o endereço de destino do pacote a ser encaminhado deve coincidir exatamente com o endereço de rede para que esta rota seja utilizada.

Para configurar o servidor do Diretor é usado o software Piranha. No navegador pode-se acessar digitando o IP do diretor e a porta 3636.

Na figura 4.2 é mostrado o menu da interface do software Piranha.

Figura 4.2 – Menu da interface do front-end piranha

Na tela CONTROL/MONITORING foi marcada a opção de Auto update no intervalo de tempo de 10 segundos. Essa opção mostra como está o balanceamento de carga.

Na tela GLOBAL SETTINGS é preciso configurar o IP real do Diretor e decidir o método de roteamento. Nosso verdadeiro IP é 200.131.250.20.

A tela REDUNDANCY define as configurações do backup do diretor.

Na tela VIRTUAL SERVERS é preciso clicar no botão adicionar onde é possível configurar as opções do Servidor Virtual e do Servidor Real.

Na figura 4.3 é mostrado o menu da virtual servers.

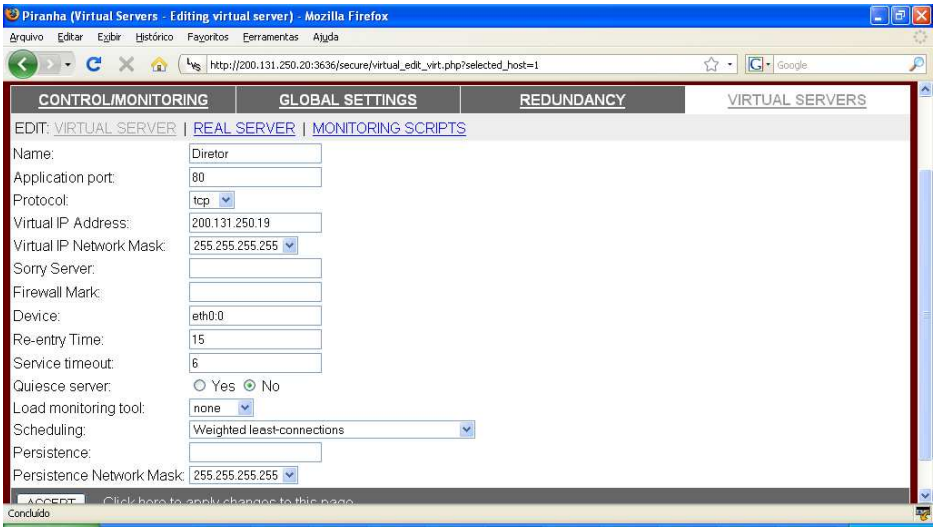

Figura 4.3 – Menu virtual servers

Name é o nome que identifica a aplicação desse servidor virtual. *Aplication port* foi definido como 80 por se tratar da porta que o servidor web está respondendo. O *virtual IP Address* e o *Virtual IP Network mask* é o IP e a máscara que definimos na interface de rede virtual. *Device* é a interface virtual que foi criada. *Re-entry time* é o período de tempo, em segundos, antes que o Diretor tente trazer um servidor real de volta ao balanceamento após uma falha. *Service Timeout* é o período de tempo, antes de um servidor real ser considerado morto e retirado do balanceamento. O campo *scheduling* possui os algoritmos de balanceamento de carga.

A figura 4.4 mostra o menu da virtual servers/real server.

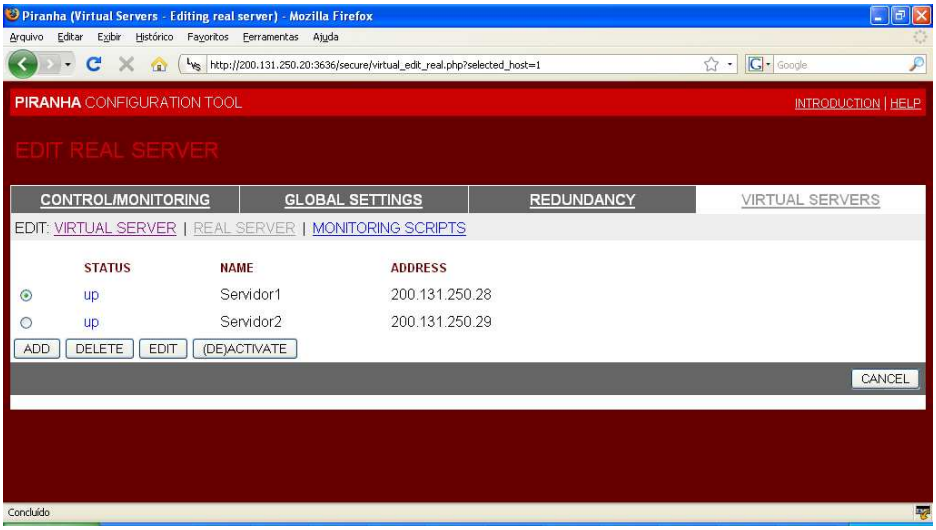

Figura 4.4 – Menu virtual servers/real server

Na opção Real Server é preciso definir o peso de cada servidor e qual seu IP real.

As configurações feitas através da interface do Piranha se encontram no arquivo /etc/sysconfig/ha/lvs.cf.

Depois dessas configurações é necessário configurar a interface virtual no servidor real 1 e no servidor real 2. Essa interface deve ser configurada na interface virtual de *loopback.*

Ifconfig lo:0 200.131.250.19 máscara 255.255.255.255.

Para fazer a sincronização dos dados entre os servidores é preciso criar um script para executar o rsync de forma automática. Abaixo o script que precisa ser agendado no cron do servidor 1.

```
#!/bin/sh
   rsync -Cravz --progress --partial --delete-excluded
root@200.131.250.29:/var/www/html/ /var/www/html/
```

É necessário implementar autenticação automática entre os servidores para que no momento da sincronização não seja necessário a autenticação manual. Para isso é preciso criar chaves de segurança usando o programa `ssh-keygen`. Depois de gerada a chave pública no servidor 1 é preciso copiá-la para o servidor 2 e coloca dentro do arquivo /root/.ssh/authorized_keys. Assim o servidor 1 pode acessar o servidor 2 sem autenticação manual.

O `cron` foi usado para rodar o script do Rsync automaticamente de 5 em 5 minutos. No arquivo /etc/crontab foi adicionado a seguinte linha:

*/5 * * * * root sh /root/sincroniza_apache.sh

A sintaxe de configuração para um agendamento do cron é a seguinte:

(minutos) (horas) (dias do mês) (mês) (dias da semana) (usuário) (comando).

4.2 – Considerações finais

Neste capítulo foi apresentado os detalhes de configuração e os comandos utilizados para implementação do projeto.

A seguir serão apresentados os resultados dos testes simulados para avaliar o funcionamento do cluster.

5 - RESULTADOS

Neste capítulo são mostrados os teste realizados para avaliar o funcionamento do ambiente e análise dos resultados obtidos.

5.1 – Testes

Para conferir o perfeito funcionamento do sistema cinco testes foram aplicados.

Teste 1 – Balanceamento com pesos iguais

Foi simulado noventa e cinco requisições ao IP 200.131.250.19 (Diretor) e observado pelo ipvsadm se o balanceamento de carga estava funcionando da forma correta. O balanceamento ocorreu como previsto conforme mostra a figura 5.1.

```
[root@salsa ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  200.131.250.19:http wlc
  -> 200.131.250.29:http          Route   1      0          48
  -> 200.131.250.28:http          Route   1      0          47
[root@salsa ~]#
```

Figura 5.1 – Balanceamento com pesos iguais

A figura 5.1 mostra que o Servidor Real 1 (200.131.250.28) e o Servidor 2 (200.131.250.29) estão ativos. No campo *Weight* é possível observar que os dois servidores têm pesos iguais. No campo *InActConn* é possível observar que Servidor real 1 recebeu 48 requisições e o Servidor real 2 recebeu 47 requisições encaminhadas pelo Diretor.

Teste 2 – Balanceamento com pesos diferentes

Foi alterado o valor do peso para 3 no Servidor 1. Isso significa que a cada três requisições que o Servidor 1 responde, o Servidor 2 responde por uma. A figura 5.2 mostra o balanceamento com pesos diferentes.

```
[root@salsa ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  200.131.250.19:http wlc
  -> 200.131.250.28:http          Route   3      0          7
  -> 200.131.250.29:http          Route   1      0          2
[root@salsa ~]#
```

Figura 5.2 – Balanceamento com pesos diferentes

A figura 5.2 mostra que o Servidor Real 1 (200.131.250.28) e o Servidor 2 (200.131.250.29) estão ativos. No campo *Weight* é possível observar que os servidores têm pesos diferentes. No campo *InActConn* é possível observar que das nove requisições que foram feitas, o servidor 1 recebeu 7 requisições e o servidor 2 recebeu 2 requisições encaminhadas pelo Diretor.

Teste 3 – Alta disponibilidade – Servidor 2

O servidor Apache do servidor 1 foi desligado. Somente o Servidor 2 pode responder as requisições. A alta disponibilidade funcionou corretamente conforme a figura 5.3.

```
[root@salsa ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  200.131.250.19:http wlc
  -> 200.131.250.29:http          Route   1      0          4
[root@salsa ~]#
```

Figura 5.3 – Alta disponibilidade – Servidor 2

A figura 5.3 mostra que somente o Servidor 2 (200.131.250.29) está ativo. No campo *InActConn* é possível observar que as quatro requisições que foram repassadas pelo diretor, foram respondidas pelo servidor 2.

Quando o Servidor Apache foi reiniciado no servidor 1, o sistema do Diretor voltou a reconhece-lo e o colocou na lista para realizar o balanceamento. Como mostra a figura 5.4 o servidor 1 ainda não respondeu a nenhuma requisição.

```
[root@salsa ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  200.131.250.19:http wlc
  -> 200.131.250.28:http          Route   1      0          0
  -> 200.131.250.29:http          Route   1      0          4
[root@salsa ~]#
```

Figura 5.4 – Alta disponibilidade – Servidor 1 restaurado

Teste 4 – Alta disponibilidade – Servidor 1

O servidor Apache do Servidor 2 foi desligado. Somente o Servidor 1 pode responder às requisições. A alta disponibilidade funcionou corretamente conforme a figura 5.5

```
[root@salsa ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  200.131.250.19:http wlc
  -> 200.131.250.28:http          Route   1      0          12
[root@salsa ~]#
```

Figura 5.5 – Alta disponibilidade – Servidor 1

A figura 5.5 mostra que somente o Servidor 1 (200.131.250.28) está ativo. No campo *InActConn* é possível observar que as doze requisições que foram repassadas pelo Diretor, o Servidor 1 respondeu por todas.

Apache reiniciado no Servidor 2 e foram feitas oito requisições ao IP 200.131.250.19. O balanceamento volta a funcionar conforme figura 5.6.

```
[root@salsa ~]# ipvsadm
IP Virtual Server version 1.2.1 (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  200.131.250.19:http wlc
  -> 200.131.250.29:http          Route   1      0          8
  -> 200.131.250.28:http          Route   1      0          12
[root@salsa ~]#
```

Figura 5.6 – Alta disponibilidade – Servidor 2 restaurado

Teste 5 - Sincronização

Foi realizado a sincronização de dados através da colocação dois arquivos na pasta /var/www/html do Servidor 2, um com 27 Mb e outro com 148 Mb para testar o tempo de sincronização. Foi colocado um arquivo chamado teste 4 no Servidor 1 na pasta /var/www/html para testar se o mesmo seria apagado automaticamente. A figura 5.7 mostra o resultado da sincronização.

```
[root@cluster01 www]# sh /root/sincroniza_apache.sh
receiving incremental file list
deleting teste4
./
index.html
          24 100%   23.44kB/s    0:00:00 (xfer#1, to-check=2/4)
teste1
    27992891 100%    6.37MB/s    0:00:04 (xfer#2, to-check=1/4)
teste2
   154953029 100%   26.61MB/s    0:00:05 (xfer#3, to-check=0/4)

sent 77 bytes  received 36349753 bytes  3461888.57 bytes/sec
```

Figura 5.7 – Resultado da sincronização

A figura 5.7 mostra que somente o arquivo chamado teste4 colocado no Servidor 1 foi apagado e que a sincronização dos arquivos do Servidor 2 para o Servidor 1 demorou 9 segundos.

5.2 – Considerações finais

Este capítulo mostrou os testes realizados para medir a eficiência do projeto e os resultados obtidos com a implantação do cluster.

6 - CONCLUSÃO

O cluster, experimento desse trabalho, se mostrou eficaz no balanceamento de carga e na alta disponibilidade do sistema. O servidor Web recebe muitas requisições ao mesmo tempo, com o balanceamento e a escolha do algoritmo *Weighted Least Connection* foi possível dividir as requisições de acordo com o poder de processamento de cada máquina. Além disso, o sistema é escalável, ou seja, com o crescimento do número de requisições ao Diretor é possível adicionar novos nós ao cluster, melhorando o balanceamento.

Com a alta disponibilidade garantiu que mesmo após a falha de um dos servidores, o sistema não saiu do ar e logo que o servidor se recuperou da falha, automaticamente voltou a fazer parte do balanceamento.

A sincronização se mostrou eficiente em relação ao tempo de sincronização/cópia, mas ineficiente na alta disponibilidade. Com o uso do sistema de arquivos EXT, utilizado no experimento, os servidores não conseguiriam gravar ou escrever simultaneamente. As atualizações só podem ser feitas no Servidor 2 e replicadas para os outros. Quando o Servidor 2 fica indisponível, não é possível atualizar o conteúdo do site.

Um ponto essencial é ter a alta disponibilidade do Diretor. No ambiente de testes se o Diretor apresentar alguma falha, o serviço fica totalmente inoperante, até sua recuperação.

Como o servidor de páginas Web é um serviço essencial da universidade, ficou comprovada a necessidade de um cluster desse tipo instalado na instituição.

7 - TRABALHOS FUTUROS

Durante o desenvolvimento do projeto, foram analisados algumas necessidades para melhoria do projeto.

Uma proposta é de implementar a sincronização dos dados usando um sistema de arquivos GFS para que a sincronização não dependa de um único servidor. Assim, os dados podem ser lidos e escritos em todos os nós do cluster simultaneamente.

Utilizar a alta disponibilidade no Diretor, já que se esse ficar inoperante, todo o sistema não funcionará.

8 – REFERÊNCIAS BIBLIOGRÁFICAS

[1] TANENBAUM, A. S; WOODHULL, A, S.. Sistemas Operacionais; projetos e implentação. Porto Alegre - 2 edição. Editora Bookman, 2000

[2] MENETH, E.; SYNDER, G.; HEIN, T. R. Manual Completo do Linux: Guia do Administrador. Editora Person, 2 edição, 1007

[3] MARTINS, E; Ferreira, G; Mendes, H. M. Cluster. Artigo Unicamp. Campinas São Paulo. 2005.

[4] PITANGA, M. Computação em Cluster, 344 páginas - 1ª edição – 2003 - Editora Brasfort

[5] HOCHSTETLER, S; BERINGER, B. Linux Clustering With CSM and GPFS, IBM Redbooks, 2004.

[6] FERREIRA, R. E. Linux: Guia do Administrador de Sistemas. São Paulo: Novatec, 2003. 510p.

[7] ZAMINHANI, D. Cluster de Alta Disponibilidade Através de Espelhamento de Dados em Máquinas Remotas. 85f. Monografia (Especialização em Administração em Redes Linux) – Departamento de Ciência da Computação, Universidade Federal de Lavras.

[8] GARTNER, F. C. *Fundamentals of fault-tolerant distributed computing in asynchronous environments* [on-line]. Disponível na Internet via www. url: http://portal.acm.org. Arquivo capturado em 05 de junho de 2008

[9] FRÓES, A. R.; SOUSA, I. C. M. Integração de Soluções em Software Livre na Configuração de um Cluster. 2004. 51f. Monografia (Engenharia de Redes de Comunicação) - Departamento de Engenharia de Redes de Comunicação, Universidade de Brasília, Brasília.

[10] SIMÕES, I. E.; TORRES, G. M. Alta Disponibilidade em Servidores. 2003. 51f. Monografia (Engenharia de Computação) - Departamento de Engenharia da Computação, Universidade Federal de Goiás, Goiânia.

[11] ALVARENGA, F. V. Uma Proposta de Aplicação para a Solução do Problema da Árvore Geradora de Custo Mínimo com Grupamentos Utilizando Cluster em Linux. 2007. 63f. Monografia (Especialização em Administração em Redes Linux) – Departamento de Ciência da Computação, Universidade Federal de Lavras.

[12] STERLING, T. L. Cluster Computing with Linux, 536 pg - Editora: MIT Press, 2001

[13] BORTOLIN, E. L. P. *Alta disponibilidade usando CODA e LVS*. 2005. 68f. Monografia (Especialização em Administração em Redes

Linux) – Departamento de Ciência da Computação, Universidade Federal de Lavras.

[14] ROCHA, J. M. G. Clusters Beowulf: Aspectos de projeto e implementação. Belém: 2003. Dissertação. (Mestrado em Engenharia Elétrica). – Departamento de Pós Graduação em Engenharia Elétrica. UFPA-PA, 2003.

[15] BOOKMAN, Charles. Agrupamento de Computadores em Linux. Editora Ciência Moderna, 2003

[16] ABREU, Harley Oliveira; GORAYEB Inácio Leite. Construindo Cluster Beowulf com Software Livre.2004. 66f. (Graduação em Ciência da Computação) – Universidade da Amazônia – Centro de Ciências Exatas e Tecnologia. Belém, 2004

[17] BATISTA, C. A. *Estudo teórico sobre cluster linux*. 2007. 65f. Monografia (Especialização em Administração em Redes Linux) – Departamento de Ciência da Computação, Universidade Federal de Lavras.

[18] BOLINA, M. Algoritmos de balanceamento de carga. Disponível em: <http://www.cafelug.org.ar/eventos/sept-03/slides/linux-ha-html/linux-ha.pdf>. Acessado em 10/06/2008.

[19] TUCKER, A. B. Computer Science Handbook, 2752 pg CRC Press 2004

[20] SILBERSCHATZ, A.; GALVIN, P. B.; CAGNE, G. Sistemas Operacionais com Java. Tradução da Sexta Edição. Editora Campus. 2004.

[21] FERREIRA, A. F. Alta disponibilidade com estudo de caso em sistema de faturamento da Telefonia fixa. 46f. (Graduação em Ciência da Computação) – Centro Universitário do Triângulo Mineiro. Uberlândia, 2002

[22] DELL. Understanding Storage Concepts [on-line]. Disponível na Internet via www. url: http://support.dell.com. Arquivo capturado em 15 de junho de 2008.

[23] PEREIRA, R. B. O. Alta Disponibilidade em Sistemas GNU/LINUX. 109f. Monografia (Especialização em Administração em Redes Linux) – Departamento de Ciência da Computação, Universidade Federal de Lavras, 2005.

[24] ZAMBALDE, A. L.; PÁDUA, C. I. P. S.; ALVES, R. M. O documento científico em Ciência da computação e Sistemas de Informação. Lavras/MG: DCC/UFLA, 2008.