HEBERT LUIZ AMARAL COSTA

ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA PARA MELHORIA DE SISTEMAS COMPUTACIONAIS CRÍTICOS USANDO SOFTWARE LIVRE: UM ESTUDO DE CASO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência da Computação, para obtenção do título de *Magister Scientiae.*

VIÇOSA
MINAS GERAIS - BRASIL
2009

HEBERT LUIZ AMARAL COSTA

ALTA DISPONIBILIDADE E BALANCEAMENTO DE CARGA PARA MELHORIA DE SISTEMAS COMPUTACIONAIS CRITICOS USANDO SOFTWARE LIVRE: UM ESTUDE DE CASO

Dissertação apresentada à Universidade Federal de Viçosa, como parte das exigências do Programa de Pós-Graduação em Ciência de Computação, para obtenção do título de *Doctor Scientiae*.

APROVADA: 13 de março de 2009.

Ricardo dos Santos Ferreira
(Co-Orientador)

Mauro Nacif Rocha
(Co-Orientador)

André Gustavo dos Santos

Marcelo Lobosco

Carlos de Castro Goulart
(Orientador)

Dedico esta dissertação à Samuel Ferreira da Costa e à Marilena Ferreira do Amaral, pais amados, companheiros, incentivadores e torcedores.

AGRADECIMENTOS

A Deus pela vida, saúde e inspiração.

A minha esposa Gisele pelo amor, dedicação, incentivo constante e por ter me presenteado dando a luz a nosso filho Henrique.

Aos meus familiares e amigos pela presença, apoio e solidariedade durante esta jornada.

A meu irmão Flávio pelo companheirismo e amor incondicional.

A Emanuela, minha cunhada querida, por seu carinho constante.

Ao professor e Orientador Carlos de Castro Goulart, pelo convívio amigo, pela experiência de vida como professor e como pessoa. Obrigado por acreditar neste trabalho!

Aos demais professores do Programa de Pós-Graduação em Ciência da Computação da UFV pelas aprendizagens e experiências propiciadas.

Ao Marcos Alessandro, aluno do Curso de Ciência da Computação das Faculdades Integras de Caratinga, pela parceria e contribuições para este estudo.

Aos profissionais e técnicos deste Programa de Pós-Graduação, em especial ao Sr. Altino pela atenção e simpatia.

Aos tantos novos amigos que encontrei nesta trajetória acadêmica. Obrigado pelas trocas e cumplicidade.

Ao Prof. Ulisses Azevedo Leitão por me incentivar na carreira acadêmica, principalmente na pesquisa científica.

E por último e em especial ao Prof. Claudio Cezar de Almeida Azevedo Leitão pela confiança e apoio de sempre.

SUMÁRIO

ÍNDICE DE TABELAS

ÍNDICES DE ILUSTRAÇÕES

RESUMO

COSTA, Hebert Luiz Amaral, M.Sc., Universidade Federal de Viçosa. Março de 2009. **Alta disponibilidade e balanceamento de carga para melhoria de sistemas computacionais críticos usando software livre: um estudo de caso.** Orientador: Carlos de Castro Goulart. Co-Orientadores: Ricardo dos Santos Ferreira e Mauro Nacif Rocha.

Os principais desafios para ampla utilização de sistemas computacionais por organizações que automatizam suas regras de negócios são qualidade e estabilidade da infra-estrutura. Este trabalho descreve uma análise experimental de uma arquitetura de *cluster* de alta disponibilidade e alto desempenho para um sistema computacional que possui uma aplicação de missão crítica. Sistemas de missão crítica são sistemas que devem ter alto grau de disponibilidade e continuar a responder às requisições mesmo em presença de falhas. A estratégia adotada neste trabalho consiste no desenvolvimento de uma solução que vise a melhor utilização dos recursos computacionais disponíveis no ambiente. Os objetivos principais são melhorar desempenho e ganhar escalabilidade através do balanceamento de carga e aumentar a confiabilidade utilizando técnicas de alta disponibilidade. Os resultados obtidos mostraram que houve uma melhor utilização da largura de banda do ambiente computacional da ordem de 32,71% para entrada de dados e 58,1% para a saída de dados. Além disso, o processador teve uma redução de uso de seus recursos em 47,65% e a memória principal em torno de 14,58%. Outros resultados complementares, para a arquitetura computacional de *clusters* proposta, foram agregados na formação da solução final, a fim de permitirem análises e conclusões quando comparados a sistemas computacionais críticos convencionais. A estratégia para o desenvolvimento deste trabalho foi estudar as técnicas e métricas de avaliação de desempenho, bem como os principais parâmetros que influenciam o desempenho de um sistema computacional crítico. A partir da monitoração e ajustes destes parâmetros, foi observada uma melhoria do tempo de resposta e uma maior disponibilidade dos recursos do sistema avaliado. Posteriormente, foram desenvolvidas novas monitorações e análises para comprovar o melhor rendimento do sistema na presença de falhas, quando as técnicas de alta disponibilidade e balanceamento de carga são utilizadas.

ABSTRACT

COSTA, Hebert Luiz Amaral, M.Sc., Universidade Federal de Viçosa. March, 2008. **High availability and load balancing for improvement of critical computing systems using free software: a case of study**. Orientador: Carlos de Castro Goulart. Co-Orientadores: Ricardo dos Santos Ferreira and Mauro Nacif Rocha.

The major challenges to a broader use of computer systems for organizations that automate their business rules is the quality and stability of the infrastructure. This work describes an experimental analysis of a cluster architecture of high availability and high performance to a computer system that has a mission critical application. Mission critical systems are systems that must have high availability and continue to meet the requirements even in the presence of faults. The strategy adopted in this study was to develop a solution that aims to improve the use of the available resorces. The main objectives are to improve performance and gain scalability through load balancing and improve the reliability through the use of high availability techniques. The results have shown a better bandwidth utilization of the the computing environment around 32.71% for data input and 58.1% for data output. Furthermore, the processor had a reduction of use of its resources in 47.65% and the main memory around 14.58%. Other additional results for the proposed computer architecture clusters were aggregated in the formation of the final solution, to allow analysis and conclusions when compared to the conventional critical computer systems. The strategy for the development of this work was to study the techniques and metrics for performance analysis, as well as the major parameters affecting the performance of a critical computer system. From the monitoring and adjustments of these parameters, it was observed an improvement in response time and higher availability of the evaluated system. Subsequently, it were performed new monitoring and analysis to show the better performance of the system in presence of failures, when high availability and load balancing techniques are used.

1 INTRODUÇÃO

1.1 Problema de Pesquisa e sua Importância

Devido ao avanço da indústria de computadores e ao surgimento de recursos tecnológicos modernos que possibilitam o desenvolvimento de inúmeras aplicações, bem como a sólida base científica computacional, atualmente é cada vez maior a quantidade de organizações públicas ou privadas que trabalham com grandes volumes de negócios, de todo o tipo e que dependem de sistemas computacionais para tratamento dos dados, informações e demais recursos envolvidos.

A construção de modernos parques computacionais, com processos de negócios bem definidos, tarefas automatizadas e infra-estruturas de comunicação de dados de alta velocidade, possui vários objetivos, dentre eles os principais são aumentar a lucratividade e a competitividade. Desde sistemas bancários até caixas de supermercados e padarias, os computadores há tempos desenvolvem papéis críticos no cotidiano da sociedade moderna. Como exemplo, pode-se imaginar o caos em aeroportos devido a problemas no sistema computacional que controla as aeronaves, ou até mesmo a irritação de um cliente em uma fila parada de caixa de supermercado porque o sistema computacional está inoperante.

Porém não apenas em tais tipos de serviços, mas, principalmente, em empresas cujo maior objetivo é exatamente a oferta de algum serviço computacional, como comércio eletrônico, notícias, *sites WEB*, aplicações distribuídas, etc. Para esses ambientes, onde os sistemas de computação são um fator de grande relevância e caracterizam partes críticas do sistema global de uma organização, é necessário desenvolver plano de contingência para otimizar o desempenho das atividades do processo.

A gerência de redes de computadores é um dos instrumentos utilizados para a elaboração de estratégias que visem a melhoria do nível das operações e execuções das atividades de sistemas computacionais críticos, pois incluem o oferecimento, a integração e a coordenação de elementos de hardware, software e humanos para monitorar e realizar ações, objetivando satisfazer às exigências operacionais, de desempenho e de qualidade de serviço de sistemas computacionais.

A proposta deste trabalho é desenvolver uma solução híbrida que mescle as áreas funcionais de falhas e desempenho da gerência de redes para proporcionar um ambiente que possa fornecer características relacionadas à confiabilidade, estabilidade,

escalabilidade e melhores tempos de respostas das tarefas realizadas por um sistema computacional crítico. Neste contexto, a gerência de falhas coloca em evidência o processo de localizar um possível problema ou falha no sistema e propõe esquemas para resolvê-los o quanto antes. Já a gerência de desempenho procura assegurar que o sistema de computação esteja sempreacessível e com recursos disponíveis.

A solução a ser desenvolvida pretende utilizar ferramentas livres para proporcionar a construção de um ambiente capaz de realizar balanceamento de carga, remodelando a carga de processamento das tarefas de um sistema centralizado para um novo sistema computacional. Tendo este a capacidade de absorver as requisições de serviços dos clientes e distribuí-las de maneira coordenada em um conjunto de máquinas, objetivando melhorar o desempenho da prestação dos serviços envolvidos.

Aliado a esta estrutura, pretende-se também criar mecanismo de alta disponibilidade com o intuito de prover um aumento da disponibilidade de serviços, dados, informações e outros recursos do sistema computacional.

1.2 Hipótese

O desenvolvimento e a implementação de plano de contingência voltado para a melhoria da disponibilidade e da utilização dos recursos de um sistema computacional crítico, incluindo serviços, dados e informações, possibilitarão melhor desempenho e, conseqüentemente, uma maior satisfação dos clientes e usuários do sistema. A solução proposta pretende melhorar o tempo de resposta das tarefas do sistema, assim como permitir um maior tempo de disponibilização dos principais recursos envolvidos.

1.3 Objetivos

O objetivo geral deste trabalho é propor uma solução que crie um ambiente onde o desempenho de execução das atividades de um sistema computacional crítico seja melhorado e onde seja possível utilizar técnicas capazes de garantir a disponibilidade dos serviços, mesmo na presença de falhas.

Outras soluções complementares devem ser agregadas na formação da solução final, a fim de permitirem um resultado melhor que o observado em sistemas computacionais críticos convencionais. Os objetivos específicos em propor e desenvolver balanceamento de carga para obter melhor desempenho e alta disponibilidade para prover tolerância a falhas são:

- analisar os parâmetros gerais dos sistemas computacionais críticos, afim de compreender a influência que possuem no sistema global;
- implementar mecanismos de balanceamento de carga, de forma a distribuir de maneira eficiente e equilibrada as tarefas de processamento entre os processadores que o sistema computacional possui;
- utilizar softwares de gerência de sistemas computacionais críticos para realizar medidas de desempenho, tais como: tempo de resposta e disponibilidade do sistema computacional crítico.

Sistemas computacionais em redes que executam tarefas críticas precisam de ambiente favorável para execução de suas atividades, pois eventuais problemas acarretariam prejuízos materiais, financeiros, ou até mesmo perda de vidas humanas. Esta pesquisa se preocupa com duas questões básicas e vitais para estes sistemas: gerenciamento de desempenho e gerenciamento de falhas.

1.4 Organização da Dissertação

O restante desta dissertação está organizada da seguinte forma: o Capítulo 2 descreve uma breve contextualização teórica referente aos assuntos centrais de pesquisa deste trabalho; o Capítulo 3 apresenta o levantamento bibliográfico discutindo os principais trabalhos relacionados com este trabalho. Neste capítulo são discutidos principalmente os conceitos, técnicas e aplicações dos sistemas de computação de alto desempenho e alta disponibilidade; no Capítulo 4 é apresentada uma arquitetura de *cluster* como infraestrutura para sistemas computacionais críticos. Discute-se a proposta de alto desempenho através do ganho de escalabilidade com o balanceamento de carga e a proposta de alta disponibilidade através do uso da redundância física e lógica dos recursos computacionais disponíveis no ambiente; o Capítulo 5 apresenta o estudo de caso do trabalho e uma discussão da metodologia da solução proposta e adotada, os procedimentos para implementação e implantação do ambiente, bem como as definições dos testes e os resultados das execuções dos testes; finalmente, o Capítulo 6 apresenta as conclusões, contribuições e as possibilidades de trabalhos futuros.

2 CONTEXTUALIZAÇÃO TEÓRICA

Dificilmente nos dias de hoje o ambiente corporativo, tanto as organizações públicas quanto as privadas, executa suas atividades sem a implementação de um ambiente computacional estruturado. No início da utilização de sistemas computacionais em ambientes corporativos, os sistemas eram altamente centralizados, ou seja, existia uma central de processamento de dados onde os usuários deveriam levar seus trabalhos para o devido processamento.

Com o advento das redes de computadores, integrando computadores e meios de comunicação, este cenário mudou drasticamente. Os usuários podiam utilizar suas próprias estações de trabalho para processar, via infra-estrutura de conectividade, seus trabalhos nas centrais de processamentos de dados, que geralmente utilizavam computadores de grande porte, chamados de *mainframes,* com alta capacidade de processamento e armazenamento de dados. Esta estrutura foi usada por um bom tempo para várias finalidades, dentre elas destacavam-se sistemas de arquivos, softwares específicos e seus bancos de dados.

O cenário dos sistemas computacionais passou por consideráveis modificações devido aos avanços da indústria de computadores pessoais, proporcionando estações de trabalhos com alta capacidade de armazenamento e processamento de dados e com custo menores. Dessa forma, o modelo de um único computador de grande porte para atender a demanda de processamento, passa a ser substituído por um conjunto de computadores de menor porte interconectados. Este novo modelo apresenta vantagens como menor custo, escalabilidade e desempenho igual ou melhor que os computadores de grande porte, dependendo de sua configuração. A principal desvantagem desse modelo é o aumento da complexidade de construção de softwares aplicativos. Ou seja, preocupações que não existiam ou não eram tão relevantes como, aspectos relacionados a segurança, sincronização e dependência de infra-estrutura de comunicação fazem parte, de acordo com este novo modelo, do cotidiano dos desenvolvedores de soluções para ambientes onde os sistemas podem estar distribuídos.

Segundo Tanenbaum [TANENBAUM 2003], “as redes de computadores são um conjunto de computadores autônomos interconectados por uma única tecnologia”. Acontece, às vezes, ligeira confusão entre redes de computadores e sistemas distribuídos. A diferença básica é que para os sistemas distribuídos a existência de vários processadores é transparente para o usuário, ou seja, o sistema oferece uma interface como se houvesse um único processador. Já para as redes de computadores a existência

de vários computadores é visível para o usuário, que deve indicar explicitamente quais computadores devem participar de que tarefa.

Com relação ao uso das redes de computadores, podemos destacar ainda segundo Tanenbaum [TANENBAUM 2003] dois aspectos distintos. O primeiro está relacionado a aplicações comerciais, que se concentram no compartilhamento de recursos, e o objetivo é tornar todos os programas, equipamentos e dados ao alcance de todos os usuários a partir de políticas previamente estabelecidas, independente da distância física entre os recursos e o usuário. O segundo aspecto está relacionado ao uso das redes para aplicações domésticas, com o objetivo de acesso a informações remotas, comunicação entre pessoas, entretenimento interativo e comércio eletrônico.

2.1 Sistemas de Missão Crítica

Para muitas empresas e setores de todo tipo de atividade, onde o uso de sistemas computacionais é imprescindível para a manutenção do negócio, uma falha que interrompa seu funcionamento ou que cause a perda de dados importantes pode provocar até mesmo a falência dessas empresas. Nestes ambientes, onde os sistemas computacionais e recursos tecnológicos são importantíssimos para atividade fim da organização, os sistemas podem ser chamados de críticos, pois possuem características particulares e com elevadas exigências em termos de confiança e segurança [NEIL 1996]. Para evitar qualquer tipo de transtorno, a maioria das empresas "monta" seus sistemas como sendo de missão crítica. Ou seja, utilizam ferramentas e recursos tecnológicos para evitar a paralisação de serviços computacionais por algum tipo de falha, sobrecarga, e inclusive a perda de dados e informações importantes.

A identificação de um sistema para um ambiente organizacional qualquer é parte fundamental do processo de informatização e automatização das tarefas. É exatamente neste momento que deve-se identificar se o ambiente caracteriza ou não a necessidade de técnicas e ferramentas para propor um sistema de missão crítica. A grande maioria das aplicações desenvolvidas ultimamente considera a conectividade como peça chave em seus projetos, ou seja, o aplicativo deve estar disponível em rede para acesso remoto dos usuários. De acordo com esta perspectiva um dos primeiros parâmetros que devem ser analisados é exatamente a infra-estrutura de rede existente ou que será desenvolvida para o devido funcionamento da aplicação. A gerência de redes é um dos principais instrumentos utilizados para este procedimento.

2.2 Gerência de Redes

As redes de computadores foram concebidas objetivando inicialmente um meio de compartilhar dispositivos periféricos caros, tais como impressoras, *drivers* de alta velocidade, discos, entre outros, existindo apenas em ambientes acadêmicos, governamentais e algumas empresas de grande porte. Entretanto, a rápida evolução das tecnologias de redes, aliada à grande redução de custos dos recursos computacionais devido a evolução e transformação da indústria de computadores, motivou a proliferação das redes de computadores por todos os segmentos da sociedade, possibilitando uma grande variedade de aplicações funcionando em ambientes distribuídos.

À medida que essas redes foram crescendo e tornando-se integradas às organizações, o compartilhamento de equipamentos tornou-se aspecto secundário em comparação às outras vantagens oferecidas. Então, as redes passaram a fazer parte do cotidiano das pessoas como uma ferramenta que oferece recursos e serviços e que permite uma maior interação entre os usuários e um conseqüente aumento de produtividade. Novos serviços, como correio eletrônico, transferência de arquivos, aplicações multimídia, dentre outras, foram acrescentadas, aumentando ainda mais a complexidade das redes. Um outro fato importante na evolução do mundo da interconexão de sistemas foi a grande heterogeneidade de padrões, sistemas operacionais e equipamentos de vários fabricantes diferentes.

Devido à evolução e ao crescimento das redes, torna-se cada vez mais necessário o controle do ambiente de redes de computadores para mantê-lo funcionando corretamente. Surge então a necessidade de buscar uma maneira consistente de realizar o gerenciamento de redes para, com isso, manter toda a estrutura funcionando de forma a atender as necessidades dos usuários e conseqüentemente maior produtividade nas tarefas desenvolvidas.

Segundo Kurose [KUROSE 2005], “Gerenciamento de rede inclui o oferecimento, a integração e a coordenação de elementos de hardware, software e humanos, para monitorar, testar, consultar, configurar, analisar, avaliar e controlar os recursos da rede, e de elementos, para satisfazer as exigências operacionais, de desempenho e de qualidade de serviço em tempo real a um custo razoável”.

Dificilmente pode ser destacada uma desvantagem na utilização de gerência de redes, talvez a mais plausível possa ser o custo de instalação, configuração e administração do ambiente. Entretanto, este custo é desprezível tendo em vista os prejuízos que possam acontecer devido a problemas de estabilidade ou paradas repentinas

de um sistema de computação, principalmente para organizações que utilizam computação e infra-estrutura física de conectividade como atividade fim para seus negócios. Os principais benefícios da utilização de gerência de redes, ainda segundo Kurose [KUROSE 2005], são:

- detecção de falha em Interface: indica que uma das interfaces de rede não está funcionando;
- monitoração de Hospedeiro: verifica periodicamente se todos os hospedeiros da rede estão ativos e operacionais;
- monitoração de Tráfego: monitora padrões de tráfego entre fontes e destinos;
- monitoração de Aplicação: monitoração e manutenção de aplicações que precisam se adaptar e funcionar em redes diversificadas;
- detecção de Mudança em Tabelas de Roteamento: detecta e corrige problemas com tabelas de rotas;
- monitoração de SLA (*Service Level Agreement*): definem parâmentros específicos de medida e níveis aceitáveis de desempenho do provedor de rede em relação a essas medidas;
- detecção de Intrusos: detecta tráfego de fonte suspeita ou quando o tráfego se destina a essa fonte.

A ISO (*International Organization for Standardization*) produziu um modelo de gerenciamento de rede definindo cinco áreas apresentadas a seguir [KUROSE 2005]:

- Gerenciamento de Desempenho: objetivo é quantificar, medir, informar, analisar e controlar desempenho;
- Gerenciamento de Falhas: objetivo é registrar, detectar e reagir às condições de falha da rede;
- Gerenciamento de Configuração: objetivo é produção e gerenciamento de inventário de dispositivos;
- Gerenciamento de Contabilização: objetivo é especificar, registrar e controlar acessos de usuários e dispositivos aos recursos da rede;
- Gerenciamento de Segurança: objetivo é controlar o acesso aos recursos da rede de acordo com alguma política pré-definida.

Em ambientes onde os sistemas de computação estão distribuídos através de infra-estrutura de redes e são críticos para o ideal rendimento e evolução da atividade de um negócio de uma organização, o monitoramento e gerenciamento são atividades importantíssimas.

O acompanhamento da eficiência do ambiente proposto e implementado foi possível devido a utilização de ferramentas e técnicas de gerência de rede. Essas ferramentas na verdade são softwares personalizados, que são capazes de monitorar e intervir em áreas do sistema computacional analisado.

As ferramentas de gerência são fortes aliadas no dia-a-dia das atividades de gerenciamento de redes e sistemas computacionais. Elas auxiliam na detecção de problemas quando ocorrem, ou antes mesmo de ocorrer. Através destas ferramentas os administradores conseguem desenvolver suas tarefas adequadamente, garantindo um bom funcionamento, sem causar prejuízos para a instituição, com uma possível falha da rede ou outro recurso ou serviço do sistema de computação.

Existem diversos tipos de ferramentas de gerência, com finalidades e propósitos próprios. Para gerência de grandes redes e sistemas existem plataformas que oferecem aplicações de monitoramento e controles sofisticados. Com o crescimento dos serviços WEB, aplicações de gerência de redes e sistemas baseados em plataforma WEB estão sendo cada vez mais aceitos e utilizados. Neste trabalho as duas ferramentas utilizadas são baseadas em WEB, com interface simples, amigável e acessível de qualquer parte da rede. É apresentada a seguir uma breve discussão sobre os dois temas essenciais para atingir o objetivo proposto, que são a gerência de desempenho e gerência de falhas.

2.2.1 Gerência de Desempenho

Um sistema computacional típico geralmente é compostos por inúmeros elementos de hardware e de software. Em sistemas tipo cliente/servidor, por exemplo, existem estações clientes, servidores com seus processadores e discos e, é claro, toda a estrutura que dá sustentação ao sistema: a rede física, composta de LANs, WANs, roteadores, entre outros. Sobre toda esta estrutura de hardware, inúmeras aplicações, gerenciadores de bases de dados, protocolos, sistemas operacionais e gerenciadores de rede dividem e disputam os recursos disponíveis.

Segundo Kurose [KUROSE 2005] gerência de desempenho é a medição e disponibilização das informações sobre aspectos de desempenho dos serviços de rede. Estes dados são usados para garantir que a rede opere em conformidade com a qualidade de serviço acordados com seus usuários. Também são usados para análise de tendência e evolução.

Devido à existência de inúmeros clientes disputando os mesmos recursos, é muito comum a existência de problemas de contenção, ou seja, a aplicação usuária não

conseguir utilizar o recurso requisitado imediatamente, e, como conseqüência, o aparecimento das filas. Por exemplo, uma requisição originada de uma estação cliente em um sistema cliente/servidor deverá trafegar pelo sistema até o servidor a que se destina podendo sofrer, ao longo do trajeto, processos de espera. Alguns serão relativos a atividades realizadas nos vários recursos, tais como em discos, processadores, roteadores e nas redes. Outros, serão apenas conseqüências de esperas em filas de recursos, que foram previamente ocupados por outras transações.

Segundo Neil [NEIL 1996], a avaliação de desempenho utiliza modelos de desempenho e modelos de carga para poder representar o comportamento dos sistemas de computação. Dessa forma permitem a interferência de agentes externos sobre estes sistemas quando existir a necessidade. A representatividade dos modelos depende da qualidade dos dados que os alimentam e algumas informações são necessárias quando se trata de coletar dados de sistemas reais, tais como:

- as fontes de informações disponíveis para a coleta de dados;
- as ferramentas de monitoração disponíveis para tarefas como monitorar a utilização de recursos ou o tempo de resposta;
- as técnicas empregadas para transformar os dados coletados em números úteis e válidos para servirem de parâmetros de entrada de modelos.

De acordo com Neil [NEIL 1996], a fonte ideal para base de dados de avaliação de desempenho é o conjunto de medidas de desempenho coletadas nos próprios sistemas reais. Entretanto, na ausência destes, dados de sistemas semelhantes ou informações de vendedores e fabricantes podem ser adaptadas.

Existem ferramentas com objetivos específicos de fomentar um banco de dados com as principais informações de desempenho de sistemas computacionais, criando assim uma base histórica cronológica sobre essas informações de desempenho que poderão servir de plataforma para futuras intervenções e inserções de mecanismos que possuem a finalidade de melhorar o desempenho do sistema computacional em questão.

2.2.2 Gerência de Falhas

Uma parte significativa do processo de gerenciamento de redes e sistemas de computação baseia-se na aquisição de informações relevantes sobre a rede e demais recursos do sistema de computação, sendo as mais importantes àquelas relativas a erros, falhas e outras condições excepcionais. Os dados devem ser armazenados em forma bruta, sendo importante definir os valores aceitáveis como limiares de tolerância que, quando

ultrapassados, determinam uma sinalização para pedir intervenção de um operador, ou o início de uma operação corretiva automatizada.

A definição de gerência de falhas apresentada em [KUROSE 2005] é o conjunto de atividades de detecção, isolamento e correção das operações anormais em uma rede. Em outras palavras, a gerência de falhas tem a responsabilidade de monitorar os estados dos recursos, dar manutenção a cada um dos objetos gerenciados, e tomar decisões para restabelecer as unidades do sistema que venham a dar problemas. As informações que são coletadas sobre os vários recursos da rede podem ser usadas em conjunto com um mapa desta rede, para indicar quais elementos estão funcionando, quais estão em mau funcionamento, e quais não estão funcionado. Opcionalmente, pode-se gerar um registro das ocorrências na rede, um diagnóstico das falhas ocorridas e uma relação dos resultados deste diagnóstico com as ações posteriores a serem tomadas para o reparo dos objetos que geraram as falhas. O ideal é que as falhas sejam detectadas antes que os efeitos prejudiciais, decorrentes destas, possam vir a acontecer. Pode-se conseguir este ideal através do monitoramento das taxas de erros do sistema e da evolução do nível de severidade gerado pelos alarmes (função de relatório alarme), que permitem a emissão das notificações de alarme ao gerente, podendo assim definir as ações necessárias para corrigir o problema e evitar as situações mais críticas.

A gerência de falhas, então, requer constante observação do funcionamento dos dispositivos que a compõem de forma que se possa identificar rapidamente o problema e resolvê-lo sem grandes prejuízos.

2.3 Balanceamento de Carga

Problemas de desempenho, incluindo baixos tempos de resposta, congestionamento da rede e interrupções de serviço são conseqüências de crescimento constante e na maioria das vezes desarticulado dos sistemas computacionais e também de infra-estrutura de redes de comunicação de dados. Pode ser adicionado a este o fato de que todo hardware ou qualquer outro recurso de um sistema computacional possui um limite físico. Uma possível abordagem na tentativa de melhorar o desempenho do sistema é desenvolver o mecanismo de balanceamento de carga para a aplicação que se encontra sobrecarregada.

Balanceamento de Carga (*Load Balancing*) é um mecanismo usado para atingir escalabilidade, dividindo a carga de processamento entre um conjunto de duas ou mais máquinas [TEODORO 2004]. O objetivo específico de desenvolver balanceamento de carga para um sistema computacional crítico é promover a melhoria de desempenho do

mesmo, através da distribuição das tarefas a serem executadas. O funcionamento básico se resume em distribuir o tráfego das chamadas ao sistema, fazendo com que as diferentes máquinas do conjunto funcionem como uma única. Também mantém o tempo de resposta das requisições, de acordo com valores limites, e oferece escalabilidade de serviços e recursos, ou seja, à medida em que houver aumento de demanda (novas aplicações, maior número de usuários conectados, etc), mais máquinas podem ser incorporadas ao conjunto, otimizando assim o poder de resposta.

Estas soluções podem ser especializadas em pequenos grupos sobre os quais se faz um balanceamento de carga, o intuito é otimizar o uso de vários recursos, tais como processador, disco, ou de rede. Qualquer uma delas introduz o conceito de *cluster*, ou *server farm*, já que o balanceamento será, provavelmente, feito para vários servidores. Um *cluster*, ou aglomerado de computadores é formado por um conjunto de computadores, que utiliza-se de um tipo especial de sistema operacional classificado como um sistema distribuído [BUYYA 1999]. É construído muitas vezes a partir de computadores convencionais, sendo que estes vários computadores são ligados em rede e comunicam-se através do sistema de forma que trabalham como se fossem uma única máquina de grande porte.

Em [BARROSO 2003] é apresentado um trabalho de computação com *clusters* que possui uma grande visibilidade e um enorme número de usuários. Trata-se da arquitetura de *clusters* da máquina de busca do google. A arquitetura proposta visa melhorar o desempenho das consultas por textos e imagens na Internet através de um *cluster*. O funcionamento básico da estrutura consiste na distribuição da carga de trabalho entre os quinze mil computadores que compõe o aglomerado. Além disso, há também o objetivo de garantir que as funcionalidades da aplicação estejam sempre disponíveis aos seus usuários, utilizando para isto técnicas de tolerância a falhas.

Um dos objetivos deste trabalho é conseguir melhores índices de desempenho do sistema de computação através da distribuição do processamento entre os servidores na rede, assim como em [BARROSO 2003].

2.4 Alta Disponibilidade

Confiabilidade e disponibilidade são cada vez mais desejáveis e necessários em sistemas de computação. Isto devido ao aumento da dependência das organizações de sistemas automatizados e informatizados [PEREIRA 2004].

Tolerância a falhas e alta disponibilidade são algumas vezes citadas como

sinônimos. Segundo Pankaj [PANKAJ 1994] um sistema é tolerante a falhas se ele pode mascarar a presença de falhas no sistema utilizando mecanismos de redundância em nível de hardware e/ou software. A alta disponibilidade está ligada diretamente à crescente dependência de computadores. Com o avanço das atividades que necessitam de recursos computacionais, é cada vez maior o transtorno causado por eventuais falhas destes sistemas. O objetivo específico de promover alta disponibilidade resume-se na garantia de que o sistema estará sempre à disposição quando o cliente ou usuário o requisitar.

Como falhas são inevitáveis em ambientes computacionais, são utilizadas técnicas para garantir a disponibilidade dos recursos de sistemas críticos, mesmo na presença de falhas. Estas técnicas podem ser tanto em nível de hardware quanto de software. Em hardware, procura-se utilizar redundância de equipamentos, para que um componente em falha seja compensado por outro. Já em Software, *clusters* são desenvolvidos com configurações que permitem atingir comportamento similar em nível de hardware: a falha de um nó é compensada pela migração dos recursos comprometidos para outro nó operante [PEREIRA 2004].

Segundo Anderson [ANDERSON 1981], para tornar populares as soluções que nos garantem a confiança que depositamos em sistemas de computação, vários desafios devem ser vencidos. O primeiro deles é a detecção de falhas seguido de um projeto para solucioná-las. O objetivo de tolerância à falhas é alcançar dependabilidade, que indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido. Redundância é a palavra chave em tolerância a falhas, sendo que todas as técnicas a respeito envolvem alguma forma de redundância. Redundância está tão intimamente relacionada a tolerância à falhas que na indústria nacional o termo usado para designar um sistema tolerante a falhas é sistema redundante. Aplicações de banco de dados têm sido um campo com necessidades de tolerância à falhas, principalmente visando a integridade de dados e disponibilidade dos mesmos. Aplicações que empregam Sistemas Gerenciadores de Bancos de Dados (SGBDs) requerem que a integridade dos dados seja preservada durante a operação normal, bem como após a recuperação ocasionada por uma falha. Preservar a integridade e a disponibilidade implica em medidas extras de segurança na forma de verificação de consistência e redundância. Em caso de falhas, informações vitais podem ser perdidas. Desta forma, uma parte essencial no sistema de banco de dados é um esquema de recuperação responsável pela detecção de erros e pela restauração do banco de dados para um estado consistente, que existia antes da ocorrência da falha.

2.5 Balanceamento de Carga *versus* Alta Disponibilidade

Aparentemente não existe nenhuma relação entre os conceitos de balanceamento de carga e alta disponibilidade. Entretanto, na prática os conceitos são freqüentemente ligados, pois o investimento feito para se adquirir sistemas redundantes para alta disponibilidade não pode ser justificado se o equipamento adicional está ocioso, ou está apenas duplicando o trabalho executado nos servidores principais. Freqüentemente, o requisito é possuir um conjunto de máquinas que estará preparado tanto para substituir máquinas defeituosas em caso de falha, como também para dividir a carga de trabalho em uma situação normal.

Neste trabalho é concentrado esforço para mesclar a gerência de desempenho, com o uso do mecanismo de balanceamento de carga, com a gerência de falhas, utilizando-se de técnicas de alta disponibilidade, adotando assim um modelo híbrido para melhoria de sistemas computacionais críticos apresentado no capítulo 4.

3 SISTEMAS DE COMPUTAÇÃO DE ALTA DISPONIBILIDADE E ALTO DESEMPENHO

A evolução dos sistemas de computação é clara e pode ser percebida sem nenhum tipo de esforço. Cada vez mais presentes em nossas vidas, esses sistemas migraram de um uso restrito à grandes instituições para uso acadêmico, doméstico e em aplicações bem especificas de negócios em organizações. Pode-se prever para o futuro uma realidade ainda mais intensa para o uso desses sistemas, cotidianamente e totalmente transparente.

Existem sistemas computacionais que podem ser críticos, onde usuários e clientes possuem alto nível de dependência desses, como um sistema de tráfego aéreo por exemplo. Mas também existem sistemas onde essa dependência não existe ou existe apenas um desconforto de não usá-los para o desenvolvimento de algum tipo de atividade, como um sistema de correio eletrônico por exemplo, onde um usuário pode ter o transtorno e ficar incomodado por não conseguir acessar sua caixa postal e ler suas mensagens por algumas horas, mas este fato não acarretaria prejuízos enormes nem perdas de vidas.

3.1 Desafios Atuais

Segundo Weber [WEBER 2002] e Fujimara [FUJIMARA 2006], os desafios atuais para computação de aplicações críticas concentram-se principalmente em tornar populares soluções que garantam confiança. Alguns aspectos que devem ser tratados em projetos de aplicações críticas são:

- evitar, detectar e contornar *bugs* no projeto de hardware e software;
- gerenciar a altíssima complexidade dos sistemas atuais, tanto em hardware com dezenas de chips de milhões de transistores, quanto em software, com milhares de linhas de código;
- explorar paralelismo para aumentar o desempenho sem comprometer a qualidade dos resultados, mesmo em caso de falha de um ou mais componentes do sistema;
- aproveitar novas tecnologias mais rápidas, baratas e eficientes (mas ainda não totalmente provadas e testadas) sem saber ainda seu comportamento em situações inesperadas sob falha ou sobrecarga;
- aproveitar, para aplicações críticas e para operação em tempo real, o modelo de sistemas distribuídos construídos sobre plataformas não

confiáveis de redes, contornando os problemas de perdas de mensagens, particionamento de rede e intrusão de *hackers*;

- conciliar alta confiabilidade e alta disponibilidade com as crescentes demandas por alto desempenho.

Todos esses desafios ainda permanecem sem uma solução ideal. Este trabalho propõem uma heurística, descrita no capítulo 4, para conciliar confiabilidade e disponibilidade com alto desempenho.

3.2 Requisitos de Sistemas de Computação em Geral

Os sistemas computacionais possuem um conjunto específico de requisitos que devem ser satisfeitos. Estes requisitos dependem das características particulares de cada sistema, o que resulta em conjuntos de requisitos diferentes de um sistema para outro.

O conjunto de requisitos define o nível de complexidade de um sistema. Normalmente os requisitos são classificados em funcionais, que se referem ao comportamento que se espera do sistema, tais como: dados de entrada, metodologia do processamento das entradas e as saídas esperadas. Outro tipo de requisitos são os não-funcionais, ou operacionais, que descrevem as característica de usabilidade, disponibilidade, escalabilidade, tempo de resposta, custo financeiro e segurança que o sistema deve conter.

O grande problema para as aplicações distribuídas é o alto nível de interdependência que existe entre os requisitos desses sistemas. Por exemplo, é extremamente complexo tentar manter características como correção e segurança quando é necessário a existência de escalabilidade [FUJIMARA 2006]. Dessa forma, é praticamente impossível tentar tratar os requisitos individualmente, sendo necessário analisar o conjunto devido ao alto nível de integração desses requisitos.

3.2.1 Dependabilidade

O termo dependabilidade (*dependability*) pode ser definido como a confiança que se pode depositar no serviço oferecido por um dado sistema de computação. A dependabilidade envolve vários atributos como confiabilidade, disponibilidade, segurança funcional crítica (*safety*), integridade, facilidade de manutenção e garantias de desempenho adequado mesmo na ocorrência de falhas não previstas [AVIZIENIS 2004]. Os principais atributos da dependabilidade são disponibilidade e confiabilidade. Disponibilidade é

definida como a probabilidade do sistema estar operacional no instante em que for solicitado, enquanto que a confiabilidade se refere à probabilidade de que um sistema funcione corretamente durante um dado intervalo de tempo correspondente ao tempo de execução de uma atividade específica.

Segundo Avizienis [AVIZIENIS 2004] as técnicas que devem ser empregadas para cada caso são:

- Prevenção de Falhas: impede a ocorrência ou a introdução de falhas. Envolve a seleção de metodologias de projeto e de tecnologias adequadas para seus componentes;
- Tolerância a Falhas: fornece o serviço desejado mesmo na presença de falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização, confinamento, recuperação, reconfiguração e tratamento;
- Validação: remoção de falhas, verificação da presença de falhas;
- Previsão de Falhas: estimativas sobre presença de falhas e estimativas sobre a consequências de falhas.

Define-se latência de falha como o período de tempo desde a ocorrência da falha até a manifestação do erro devido aquela falha e latência de erro como o período de tempo desde a ocorrência do erro até a manifestação do defeito devido aquele erro [WEBER 1990].

Os conceitos de falha, erro e defeito devem ser claros quando trata-se de dependabilidade para um sistema qualquer. Segundo Weber [WEBER 1990], um defeito (*fault*) é definido como um desvio da especificação do sistema. Um sistema pode também encontrar-se em estado de erro ou errôneo (*error*), caso o processamento posterior ao estado atual leve o sistema a um estado de defeito. Finalmente define-se como falha (*failure*) como a causa física ou lógica do erro. Em outras palavras, falhas físicas ou lógicas do sistema evidenciam a execução de funcionalidades de maneira errada ou com erros, o que resulta em defeitos temporários ou permanentes no sistema.

As principais técnicas no contexto de utilização do atributo de alta disponibilidade para alcançar tolerância a falhas são mascaramento ou detecção e localização com reconfiguração. Com mascaramento as falhas não se manifestam como erros, pois são mascaradas na origem da ocorrência. Já a segunda técnica consiste exatamente em detectar, localizar e reconfigurar hardware e softwares para resolver a falha e não deixar que os estados de erro e defeito apareçam no sistema.

3.2.1.1 Medidas de Confiabilidade

As medidas de confiabilidade mais usadas na prática segundo Buyya [BUYYA 1999] são: taxa de defeitos, MTTF (M*ean Time to Failure* – tempo esperado até a primeira ocorrência de defeito), MTTR (M*ean Time to Repair* – tempo médio para reparo do sistema), MTBF (M*ean Time Between Failure* – tempo médio entre falhas no sistema). Cada fabricante, tanto de hardware quanto de software, deve fornecer essas medidas para seus produtos comercializados. Entretanto este fato não acontece na prática para boa parte dos fabricantes. Essas medidas são determinadas pelo fabricante estatisticamente, observando o comportamento dos componentes e dispositivos fabricados.

Em termos técnicos, a disponibilidade de certo dado ou serviço é a probabilidade de encontrá-lo operando normalmente em determinado momento. Esta probabilidade considera o provável *uptime* (tempo de funcionamento da aplicação crítica) e o provável *downtime* (tempo em que a aplicação crítica não está funcionando).

Uma outra notação para medir disponibilidade é a que considera o número de noves de disponibilidade. Ou seja, dizer que uma aplicação crítica possui 6 noves de disponibilidade é dizer que a aplicação está 99,9999% disponível. A equação que calcula o grau de disponibilidade de uma aplicação crítica é: A = MTBF/(MTBF+MTTR) [BUYYA 1999].

3.2.1.2 Escalabilidade

A definição de escalabilidade depende exclusivamente do contexto abordado. Segundo Buyya [BUYYA 1999], escalabilidade em sistemas computacionais surge quando esses sistemas são capazes de satisfazer maior demanda ou prestar um volume maior de serviços com um aumento adequado de recursos. Para um sistema distribuído, uma das características mais importantes é ser escalável no número de componentes envolvidos e com isto o sistema deve manter, por exemplo, o mesmo nível de disponibilidade à medida em que o número de componentes aumenta, aumentando o desempenho global do sistema.

Em termos mais práticos a escalabilidade é a habilidade que uma aplicação tenha de suportar um crescente número de requisições a serviços e demais recursos do sistema oriundos dos usuários. Exemplo: se uma aplicação leva 10 ms para responder uma requisição, quanto tempo ela levará para responder 10.000 requisições concorrentes? Escalabilidade infinita permitiria que a aplicação respondesse essa carga em 10 ms, ou

seja, gastaria-se o mesmo tempo para responder uma requisição ou 10.000 requisições. Escalabilidade, então, pode ser entendida como uma medida que depende de vários fatores, incluindo o número de usuários simultâneos que um sistema computacional configurado como um *cluster* pode suportar e o tempo que se leva para responder uma requisição.

3.3 Computação com *Clusters*

Clusters têm sido construídos e utilizados por mais de duas décadas. Segundo Dantas [DANTAS 2005] o modelo de *clusters* surgiu em duas diferentes áreas da computação com ligeiras diferenças e motivações. Resume-se em Alto Desempenho (HP – *High Performance*) e Alta Disponibilidade (HA – *High Availability*). *Clusters* podem ser definidos genericamente segundo Tanenbaum [TANENBAUM 2007] como "um sistema paralelo ou distribuído que consiste de uma coleção de computadores todos interligados, utilizados como um único, unificando assim os recursos de computação envolvidos". Já a comunidade de computação paralela visualiza os *clusters* sob a perspectiva de alto desempenho e escalabilidade e refere-se às máquinas envolvidas como WSCs (*Workstation Clusters*) ou NOWs (*Network Of Workstations*). Para a área da computação de aplicações de missão crítica, os *clusters* possuem um valor um pouco diferente. Estão relacionado à alta disponibilidade e são geralmente referenciados como HA *cluster*.

Os modelos de *clusters* podem fornecer tanto alta disponibilidade quanto alto desempenho, e também outras características como gerenciabilidade, escalabilidade e acessibilidade. Ou seja, as principais necessidades de qualquer departamento de tecnologia da informação.

No campo da computação paralela, um *cluster* é principalmente um meio de prover escalabilidade e por consequência aumento do poder de processamento do sistema de computação, permitindo assim que uma aplicação paralela possa ser executada em todos os *nodos* existentes no *cluster*. Uma vantagem de um *cluster* quando comparado com computadores de grande porte tradicionais é o baixo custo. Durante a década de noventa os *clusters* surgiram como uma resposta às caras arquiteturas proprietárias de computadores de grande porte dos anos oitenta. É fácil observar que as aplicações dos dias de hoje estão bem mais exigentes quanto ao poder de processamento e armazenamento que as aplicações das décadas de oitenta e noventa. É possível perceber também que, para as aplicações atuais, há necessidade de mesclar arquiteturas mais

elaboradas como processadores com vários núcleos e tecnologias avançadas de memória primária, com o desenvolvimento de *clusters* capazes de atender seus principais objetivos como escalabilidade, aumento do poder de processamento e maior disponibilidade dos recursos envolvidos.

A tecnologia de *clusters* usa muitos conhecimentos adquiridos das pesquisas de sistemas distribuídos. Em ambientes computacionais com *clusters* é inerente a possibilidade de problemas, pois existe um número elevado de componentes de hardware que provavelmente falharão durante a execução do sistema. Conseqüentemente é necessário o desenvolvimento de mecanismos específicos que garantam a configuração e execução dos aplicativos diante da presença de falhas.

Clusters de missão crítica foram originalmente construídos, em muitos dos casos, com apenas duas máquinas (chamadas *nodo* primário e *nodo* secundário) e normalmente suportados por OLTP (*Online Transaction Processing* – Processamento de Transações *Online*) para aplicações bancárias, seguros, e outras aplicações críticas similares. A idéia é simples em conceito: no caso de uma falha de um componente (hardware ou software) no sistema primário, a aplicação crítica pode ser transferida para o servidor secundário, evitando assim a paralisação e garantindo a disponibilidade da aplicação. Este processo é definido como *failover*. Inevitavelmente ocorre, durante este processo, uma perda de rendimento temporária do serviço, mas sem perda total permanente. Este procedimento é uma forma de utilizar redundância no nível de componentes de hardwares com esquemas de votações para mascarar as falhas. A redundância neste mecanismo pode ser no nível de software também.

A convergência de *clusters* de alta disponibilidade em *clusters* de alto desempenho não é coincidência. Ambos utilizam as tecnologias existentes de cada universo. O processamento paralelo utiliza o poder de processamento de muitas unidade de processamento (nodos do *cluster*) para conseguir maior desempenho. Contudo, a utilização de tolerância a falha é essencial e extremamente eficaz para a computação paralela, pois permite maior disponibilidade da aplicação crítica. Por outro lado, o paralelismo permite alta disponibilidade, pois um sistema executado em paralelo possui intrinsecamente alta disponibilidade devido a redundância dos equipamentos envolvidos.

Os problemas em sistemas computacionais podem ser causados por falhas permanentes (por exemplo, danos permanentes de um componente de hardware), mas principalmente por aqueles transitórios (como por exemplo, devido ao aquecimento, interferências eletromagnéticas, erros de projeto, falha transitória de software, falha e

energia, distúrbios elétricos, entre outros). O impacto das falhas transitórias podem ter conseqüências graves ou até mesmo catastróficas, como exemplo em sistemas de controle de tráfego aéreo, onde um cálculo errado devido a uma falha transitória de software pode provocar perda de vidas.

3.3.1 Computação de Missão Crítica

Organizações de vários fins dependem exclusivamente de recursos e sistemas de computação. A consequência deste fato é o aumento da demanda de aplicações que viabilizem alta disponibilidade. Esta classe de programas incluem: processamento de transações *on-line*, comércio eletrônico, mineração de dados, sistemas de suporte a decisão, *switches* de telecomunicações, sistemas de controle e servidores de aplicações que precisam funcionar 24 horas por dia. Nesses ambientes, o custo de qualquer interrupção pode ser substancial, quer em termos de perdas de oportunidades, como um bom momento de mercado, ou até mesmo a perda de clientes para concorrentes. Com a dependência das empresas e organizações por aplicações críticas ocorre um aumento considerável em termos de investimento.

Nas últimas décadas, os *clusters* estão sendo construídos para suportar sistemas de computação com altos níveis de dependabilidade. Historicamente, segundo Buyya [BUYYA 1999], o primeiro mecanismo de proteção para aplicações críticas foi a cópia de segurança dos dados da aplicação em questão, esquema conhecido como *data offline backup* ou *cold-backup*. Geralmente é construído como um utilitário de sistema operacional, cujo único objetivo é realizar cópias de segurança da base de dados da aplicação crítica de forma desligada, ou seja, a aplicação deve estar completamente parada.

Uma evolução do conceito de cópia de dados é o *hot-backup*. Neste sistema, as cópias de segurança podem ser feitas continuamente durante a execução da aplicação, não sendo necessário que a aplicação crítica esteja desligada. O modelo em questão utiliza um esquema de armazenamento espelhado entre dois discos (primário e secundário), onde todos os dados do disco primário são espelhados no disco secundário. Este sistema também é chamado de RAID (*Redundant Array of Independent Drives* – Conjunto Redundante de Discos Independentes), cujo único objetivo é garantir a integridade dos dados da aplicação crítica de forma contínua e automática. As principais vantagens no uso de RAID são: ganho de desempenho no acesso; redundância em caso

de falha em um dos discos; uso de várias unidades de discos e facilidade em recuperação de conteúdo perdido. Apesar destas vantagens, este modelo também não garante a disponibilidade da aplicação crítica, caso um servidor venha a falhar devido a interrupção de energia, por exemplo. Mesmo com o RAID, haverá necessidade de re-estabelecimento de forma manual. Uma outra desvantagem é o aumento do custo devido ao(s) disco(s) extra(s).

O fornecimento de energia elétrica é outro fator preocupante para a computação de missão crítica. Existe uma dependência enorme das empresas responsáveis por tais serviços. Não é conveninente apostar na qualidade de serviço oferecido por essas empresas, por isso estão sendo construídos há várias décadas sistemas *no-breaks* que impedem a parada de fornecimento de energia para o sistema de computação, utilizando-se de baterias ou de geradores de energia privativo. Claro que este tipo de sistema serve apenas para falhas transitórias, caso contrário, o sistema deverá conter redundância também na geração de energia, o que pode elevar de forma considerável os custos do projeto.

Até há alguns anos atrás, as empresas montavam seus sistemas com os melhores componentes de hardwares disponíveis, incluindo algumas das estratégias mencionadas anteriormente, e monitoravam seus sistemas na esperança de que tudo funcionaria da maneira certa e que não teriam problemas. A alternativa é exatamente a construção de sistemas avançados de hardware redundante, como foi o caso das aplicações altamente disponíveis nos setores das telecomunicações e das redes bancárias. A redundância nestes sistemas é explorada em todos os níveis, incluindo as fontes de alimentação, portas de entrada e saída (*I/O ports*), processadores, discos, adaptadores de rede, e redes físicas, a fim de elminar qualquer ponto único de falha no interior da plataforma suportada pela aplicação crítica. Na eventualidade de uma falha no hardware executando uma aplicação crítica, um segundo componente de hardware estará sempre disponível para assegurar que o pedido tem recursos para manter a execução da aplicação. O objetivo é sempre resultados próximos a 100% da disponibilidade do sistema.

Os níveis de disponibilidade nestes sistemas, geralmente requerem o uso de hardware e software proprietários, personalizados para as aplicações forçando os usuários a manter o padrão do proprietário para qualquer atualização que venha a acontecer durante a evolução do sistema. Entretanto, o ideal é combinar sistemas de computação altamente confiáveis e altamente escaláveis com soluções abertas e componentes de hardware acessíveis. Isto pode ser feito com *clusters*. O primeiro passo neste sentido é

empregar dois sistemas de computação similares conectados por uma infra-estrutura de rede tipo LAN ou MAN (nodos primário e secundário). O nodo secundário seria responsável então não só pelos dados (através de espelhamento), mas também por todos os processos da aplicação crítica do servidor primário, de modo a entrar em ação no caso de uma falha. Esta estrutura marca exatamente o surgimento de *clusters* de missão crítica, pois começa a explorar as possibilidades de utilização de redundância em vários níveis.

Em paralelo a busca e evolução de alta disponibilidade para as aplicações críticas, uma outra tendência ganhou forma na área de sistemas distribuídos e paralelos. Devido à crescente difusão de sistemas de informação e automação de tarefas via sistemas de computação, tornou-se vital para os negócios críticos, tais como: comércio eletrônico, *WEB sites*(portais em geral), bases de dados corporativas, mecanismos eficientes para tratar a alta demanda de carga de trabalho gerado por estas aplicações.Ou seja, em adição às vantagens da alta disponibilidade, que concentram-se principalmente em garantir o fornecimento/disponibilidade dos recursos, dados e serviços da aplicação, seria necessário também uma forma de satisfazer às exigências de crescimento da aplicação. O alto desempenho junta-se à alta disponibilidade como um requisito adicional para aplicações de missão critica.

Um *cluster* intrinsecamente tem a capacidade de fornecer alto desempenho e escalabilidade. Ao contrário do esquema de *failover* de alta disponibilidade, um nodo do *cluster* não precisa necessariamente ficar ocioso à espera de um falha. Pode-se utilizar o processamento desse nodo e seus demais recursos, a fim de que com o software adequado, a carga de trabalho possa ser compartilhada com outros membros do *cluster*, conseguindo assim obter um melhor desempenho. Esta evolução marcou o surgimento dos *clusters* de alta disponibilidade escaláveis.

3.3.2 Redundância

Redundância surge no contexto de sistemas tolerantes a falhas ou de alta disponibilidade como requisito indispensável para atendimento às funcionalidades de um projeto de missão crítica. Todas as técnicas de tolerância a falhas envolvem alguma forma de redundância.

A utilização de redundância pode servir tanto para detecção quanto para mascaramento de falhas. O que se diferencia é o grau de redundância em cada caso. Para mascarar são necessários geralmente mais componentes do que para detectar uma falha.

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer segundo Weber [WEBER 2002] como:

- redundância de informação: bits ou sinais extras são transmitidos junto aos dados, servindo para detecção de erros ou mascaramento de falhas. Exemplos clássicos são: códigos e bits de paridade para detecção de erros, *checksums,* códigos de duplicação e códigos cíclicos. Códigos de correção de erro (ECC) estão sendo usado exaustivamente pela indústria para construção de memórias mais confiáveis e para transferência de informação de forma confiável da memória para o processador e vice-versa;
- redundância temporal: repete a computação em um tempo pré-determinado. Evita custo de hardware adicional, aumentando neste caso o tempo necessário para realizar a computação de uma aplicação crítica. Geralmente este tipo de redundância é aplicado em sistemas cujo o tempo para computação não é crítico ou quando o processador possui ociosidade durante a execução da aplicação;
- redundância de hardware: está baseada na replicação de componentes. definida assim:
  - redundância passiva ou estática: usada por técnicas de mascaramento de falhas e apresenta como principal vantagem não requerer ação do sistema, não indica a falha;
  - redundância ativa ou dinâmica: usada por técnicas de detecção, localização e recuperação e apresenta vantagens de substituição de módulos para aplicações de longa vida útil;
  - redundância híbrida: é a combinação de técnicas ativa e passiva de redundância usada principalmente para garantir mascaramento de falhas e longa vida útil do sistema, geralmente de alto custo.
- redundância de software: replicação de componentes idênticos em software é estratégia inútil para redundância em software. Pois, componentes idênticos de software apresentarão os mesmos erros. As principais técnicas de detectar e mascarar falhas utilizando redundância de software são:
  - programação n-versões;
  - blocos de recuperação;
  - e verificação de consistência.

Para softwares que foram projetados corretamente desde o início de sua construção, não são necessárias técnicas de tolerância a falhas. Entretanto, na prática não

é o que acontece, desenvolvedores criam softwares sem processos bem definidos e sem utilização de técnicas de engenharia de software e produzem programas incorretos.

Todas essas formas de redundância possuem um impacto significativo no sistema, seja no custo, no desempenho, na energia que consome e outros. Dessa forma, a redundância deve ser projetada para que não seja nem excessiva e também para que não falte recursos para provê-la.

3.3.3 Verificação por Temporizadores

Verificação por temporização é eficaz na detecção de erros. Neste mecanismo, a execução de uma operação não deve exceder um tempo máximo pré-determinado, caso contrário um *timeout* (tempo ultrapassado) é sinalizado para indicar que um defeito ocorreu.

Esta técnica é normalmente associada com o uso de temporizadores, que podem ser implementados tanto em hardware quanto em software. Quando um temporizador de hardware é utilizado, ele é chamado de temporizador "cão de guarda" (*watchdog timer*) e tem que ser periodicamente reinicializado pelo programa. Caso este temporizador não seja reinicializado devido a um problema qualquer no hardware ou software, o sistema irá levantar uma exceção.

Esta forma de redundância temporal é usada em conjunto com outras formas, para que possa cobrir uma porcentagem maior de falhas, pois embora possa indicar a presença de falhas, não existe como garantir que elas não aconteçam [ANDERSON 1981]. O grande diferencial desta técnica é seu custo baixo. As principais vantagens são: o desempenho do sistema não é afetado significativamente, pois a checagem é feita de forma concorrente, e o processo de checagem é totalmente independente do processo checado.

Um software *watchdog* é um processo simplificado que monitora outro(s) processo(s) por erros. O mais simples *watchdog* apenas assiste a aplicação até um acontecimento de algum eventual erro e a única ação é retomar o relançamento da aplicação. Se o processo monitorado está configurado para cooperar com o *watchdog*, então aumenta a eficiência da vigilância consideravelmente. Existem alguns mecanismos que podem auxiliar e cooperar com o *watchdog*. Os mais utilizados são:

- *Heartbeats*: este termo significa batimento cardíaco, e é geralmente utilizado para descrever que notificações periódicas são enviadas por uma aplicação para o software *watchdog* para confirmar a sua vida/existência. Consiste em uma aplicação iniciada, enviando mensagens com o

significado “eu estou vivo?”, ou um esquema de requisição-resposta em que o software *watchdog* requisita da aplicação a afirmação de sua existência através da resposta para a pergunta “você está vivo” e aguarda-se mensagem de aviso. Quando um *timeout* especificado expira no *watchdog*, é considerado que a aplicação caiu e que o processo da mesma deve ser finalizado e re-iniciar toda a aplicação novamente. *Watchdogs* podem coexistir em um mesmo sistema com a aplicação protegida ou em outro sistema. No primeiro caso, o *watchdog* é inútil se acontecer algum problema com o sistema operacional, mas caso o *watchdog* se encontre em outro sistema este problema não existiria;

- notificações ociosas: a estratégia de notificações ociosas consiste em informar ao software *watchdog* sobre períodos em que a aplicação crítica está ociosa, ou não está realizando nenhum trabalho previsto em suas funcionalidades. O *watchdog* então pode realizar ações preventivas, como um simples re-inicio da aplicação. Softwares que são executados continuamente e por um longo tempo, segundo Buyya [BUYYA 1999], possuem probabilidade alta de falhas;
- notificações de erros: o raciocínio para este tipo de notificação é obrigar a aplicação a enviar um sinal para o software *watchdog*, relacionando os erros encontrados durante a execução do sistema, e requisitando uma resposta de ação para a recuperação desses erros. Um reinício na aplicação pode ser o suficiente, mas outras ações corretivas podem ser executadas, como uma limpeza de arquivos ou até mesmo o reinício de todo o sistema computacional.

3.3.4 Recuperação de Falhas

Após a detecção e diagnóstico de falhas, em geral, são necessárias ações de recuperação e reconfiguração. A seguir são apresentadas as principais técnicas utilizadas referentes à recuperação e reconfiguração de sistemas mediante a presença de algum tipo específico ou não de falha.

3.3.4.1 *C*heckpointing e *R*ollback

Checkpoint é uma técnica que geralmente permite a um processo salvar seu estado em

intervalos de tempos regulares durante a execução normal da aplicação. O objetivo é a viabilidade de restauração deste processo, mais tarde, depois de uma falha no sistema. Este mecanismo reduz drasticamente a quantidade de trabalhos perdidos devido a ocorrência da falha no momento da execução da aplicação crítica. Usando *chekpoints*, quando ocorre uma falha, o processo afetado simplesmente é re-iniciado a partir da última checagem salva válida, e não iniciando o processo do início. Esta técnica é especialmente interessante para proteger longas execuções de aplicações críticas contra falhas transitórias. Essas aplicações, geralmente, são programas que rodam por dias e semanas e o reinício do zero às vezes pode ter consequências ruins.

Técnicas de *checkpointing* geralmente são classificadas de acordo com as seguintes características: transparência, dados a incluir na verificação e intervalos de verificação. Primeiro, a verificação pode ser transparente e automaticamente inserida na execução pelo compilador, ou inserida manualmente através de codificação pelo programador da aplicação. Na abordagem transparente, o *checkpoint* consiste de um instante do estado do processo, incluindo todos os dados dinâmicos do sistema operacional. Outras abordagens transparentes incluem apenas o contexto interno do processador, a pilha e os dados estáticos e dinâmicos. Em ambos os casos, *checkpointings* transparentes devem guardar grandes quantidades de dados, a maioria desnecessários, pois não se consegue nesse nível distinguir que dados são relevantes para a aplicação. Na abordagem manual isto já é possível, pois o programador consegue definir quais dados são cruciais para aplicação e assim pode-se obter *checkpoints* menores.

Outra característica importante é o intervalo do *checkpointing*. Ou seja, o intervalo de tempo entre controles consecutivos. O intervalo ideal não é fácil de prever, pois depende de vários fatores, tais como a freqüência de insucesso, a carga de trabalho do sistema, o tempo global de execução. Geralmente o intervalo do *checkpointing* está associado ao nível de confiabilidade e disponibilidade do sistema/aplicação. Mais uma vez, este problema é crítico principalmente para *checkpointing* automático. Entretanto, quando as verificações são inseridas através do programador da aplicação este procedimento é mais fácil, principalmente por poder ser inserido em pontos chaves do algoritmo.

O local onde os dados do *checkpointing* são armazenados também é crucial para o sucesso de utilização desta técnica. Em média, geralmente é conhecido como armazenamento estável e deve ser resistente às falhas de hardware, erros de software e imune aos problemas de memória. Além disso, as operações de leitura e escrita devem ser

realizadas de forma atômica. Armazenamento estável é geralmente obtido usando replicação em discos, cartões de memória especiais, replicação de memória, ou até arquivos em dispositivos de fitas.

3.3.4.2 Transações

Uma transação, segundo Buyya [Buyya 1999], é um grupo especial de operações de transformações de estados de processos em execução no sistema de computação. Em *clusters*, de acordo com os fundamentos de sistemas distribuídos, existem vários tipos de transações que necessitam ser realizadas. Dentre elas destacam-se: atualização e consultas em bases de dados, mensagens de controle, ou ações externas de computadores e operadores da aplicação. As transações em *clusters*, como em Bancos de Dados, geralmente são classificadas e caracterizadas de acordo com as seguintes propriedades, conhecidas como ACID:

- Atomicidade: significa dizer que ou acontecem todas as ações da transação ou nenhuma;
- Consistência: cada transformação deve ser vista como uma imagem correta do estado, mesmo para transações concorrentes;
- Isolamento: a transação deve ser uma transformação correta do estado;
- Durabilidade: mesmo que uma transação seja comprometida, todos os seus efeitos devem ser preservados, mesmo se houver um falha permanente ou temporária.

Transações em *clusters* possuem interface simplificada. As operações primitivas com transações são: início de transações, fim de transações e abortar transações. Essas primitivas são utilizadas no nível do software e o objetivo principal é tentar garantir a efetiva comunicação entre os processos da aplicação de acordo com escopo do projeto de *cluster*.

3.3.4.3 *Failover* e *Failback*

O processo de *failover* é o cerne do modelo de recuperação em *cluster*. Este processo pode ser entendido como a troca, através de transações, para um nodo alternativo ou um nodo de *backup*, devido a uma situação simples de falha em um dos nodos do *cluster*. O processo de *failover* deve ser totalmente transparente e automático, sem a intervenção de um administrador do sistema computacional e principalmente sem necessidade de re-conexão manual do cliente da aplicação crítica.

Para algumas aplicações, não críticas, que podem suportar um tempo maior até a recuperação da falha, pode-se utilizar *failover* manual. Além do tempo entre a falha e a sua detecção, existe também o tempo entre a detecção e o re-estabelecimento da disponibilidade da aplicação e dos recursos envolvidos no sistema computacional. Grandes bancos de dados, por exemplo, podem exigir um considerável período de tempo até que ajustem suas tabelas. E durante este tempo a aplicação ainda não estará disponível. Para realizar o *failover* de uma aplicação crítica, é necessário que os nodos do *cluster* envolvidos na utilização deste mecanismo possuam recursos equivalentes. Um recurso pode ser uma interface de comunicação de rede, um disco rígido e o mais importante, os dados existentes neste disco, e todo e qualquer elemento necessário à prestação de um determinado serviço para a aplicação crítica em questão.

Dependendo da natureza da aplicação e do sistema de computação, 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 a aplicação e os serviços disponíveis pela aplicação estejam novamente disponíveis.

O processo reverso do *failover* é denominado *failback* e consiste basicamente em mover de volta a aplicação crítica e os clientes desta, para o nodo/servidor original totalmente livre dos defeitos, erros ou falhas. Da mesma forma que o *failover*, o *failback* deve ser transparente e automático para os clientes.

O *failover* então pode ser usado para um outro importante propósito, a manutenção do nodo/servidor original da aplicação crítica. Ações simples e rotineiras, como atualização do sistema operacional ou outros softwares, e, principalmente adição ou manutenção de hardwares no sistema computacional são executadas sem a necessidade de parada de fornecimento dos serviços e recursos da aplicação crítica.

Em alguns casos, em função da possível nova interrupção na prestação dos serviços envolvidos no ambiente computacional, o processo *failback* pode não ser atraente.

3.5 Balanceamento de Carga em Redes de Computadores

Balanceamento de carga (*load balancing*) em redes de computadores significa a utilização de recursos de hardware e softwares para o processamento de nodos em um *cluster* com o objetivo de distribuir a carga de trabalho ou a carga de tráfego. A decisão de distribuir a carga para um nodo particular pode ser por opções estáticas pré-programadas, ou pode mudar dinamicamente dependendo do estado atual da rede. Os

nodos do *cluster* são interconectados ao dispositivo de balanceamento que faz a distribuição de carga para o efetivo balanceamento da mesma [BUYYA 1999].

Cada nodo participa do *cluster* fornecendo várias informações sobre os recursos existentes, tais como: estado da carga atual do processador, a aplicação de carga de sistema, o número de usuários ativos, a disponibilidade de *buffer* de protocolos, a disponibilidade de memória do sistema e outros recursos específicos. Essas informações são passadas para o dispositivo de balanceamento que monitora o status de todos os nodos, e este consegue então escalonar a próxima tarefa a ser executada para um nodo do *cluster* através de uma política de consumo desses recursos estabelecida. O dispositivo de balanceamento pode ser uma unidade única ou um grupo de unidades trabalhando em paralelo.

O dispositivo de balanceamento usa algoritmos especiais e alguns métodos para conseguir decidir qual nodo será o responsável pela próxima requisição de chegada de conexão. Esses métodos podem ser otimizados para cada tipo de aplicação dependendo exclusivamente do protocolo de tráfego da rede.

A pilha de protocolos de rede mais freqüentemente utilizada, inclusive neste trabalho, é a pilha TCP/IP, um sistema baseado em pacotes onde a informação da rede é contida em cada um desses pacotes [TANENBAUM 2003]. Isso permite que cada pacote de dados seja considerado como uma chave no algoritmo de balanceamento.

3.5.1 Métodos de Balanceamento de Carga

Balanceamento de Carga em redes pode ser desenvolvido através de vários métodos básicos que podem ser combinados para criar sistemas avançados de distribuição de tarefas em *nodos* de um *cluster*. O dispositivo de balanceamento monitora o tráfego de rede e o estado dos nodos do *cluster* usando esses métodos para determinar quando, como e onde a carga de tarefas pode ser distribuída através dos nodos do *cluster*. Os parâmetros principais, tráfego da rede e estado dos nodos, são essenciais para determinar qual nodo do *cluster* está apto à receber a nova carga. Cada uma dessas funções e métodos são influenciados por vários fatores no ambiente computacional que definem o comportamento do dispositivo de balanceamento.

Vários fatores envolvidos no sistema computacional afetam os métodos de balanceamento de carga. Esses fatores definem de fato as capacidades e os limites do dispositivo de balanceamento. Este fato está totalmente relacionado com as influências do ambiente onde se encontra o dispositivo.

O fator básico neste contexto é a estrutura da pilha de protocolos TCP/IP. Os quatro protocolos mais importantes na utilização deste modelo de referência para o dispositivo de balanceamento são: o protocolo IP (*Internet Protocol* – Protocolo Internet), o ICMP (*Internet Control Message Protocol* – Protocolo de Mensagens de Controle Internet), o TCP (*Transmission Control Protocol* – Protocolo de Controle de Transmissão) e o UDP (*User Datagram Protocol* – Protocolo de Datagrama de Usuário). Outros detalhes de outros fatores que afetam os métodos de balanceamento estão descritos e comentados a seguir:

- tradução do endereço de redes: processo que converte endereços privados ou internos e realiza o roteamento da informação em endereços públicos. Qualquer dispositivo de balanceamento requisitado para a tradução de endereço de rede deve manter tabelas separadas para representações internas e externas de informação dos computadores. Geralmente há um teste para identificar endereços que são ilegais ou proibidos. Este processo é caro do ponto de vista do processamento. O uso de NAT (*Network Address Translation* – Tradução de Endereço de Rede) é aplicado para permitir a um cliente descobrir o endereço de um nodo;
- nomes de domínios : o nome de um computador na rede tem como objetivo fazer com que este possa ser reconhecido com maior facilidade e seja entendido por humanos. Além disso pode ser usado como uma chave ou ponto de referência pelo dispositivo de balanceamento. Em muitos casos, nomes de *hosts* e nomes de domínios formam o elemento primário na decisão de um algoritmo de distribuição. O sistema de nome de domínio é um mecanismo de tradução padrão, mapeando nomes para endereços IP e vice-versa. Um nome de domínio pode mapear múltiplos endereços de redes (um domínio virtual). Esse passo da tradução ocorre no sistema do servidor de DNS (*Domain Name System* – Sistemas de Nome de Domínios). Como muitos computadores são referenciados pelos seus nomes e não diretamente pelos seus endereços IP, o servidor de DNS se torna um auxílio crucial para o dispositivo de balanceamento para auxiliar na distribuição da carga;
- processamento de alta velocidade via enlace guiado: é a habilidade de realizar o processamento do tráfego pela rede e o redirecionamento à toda velocidade dos pacotes de entrada no nodo, para prevenir engarrafamentos no dispositivo de balanceamento. Embora o hardware de interface de rede no nodo possa ser capaz de receber e transmitir informação a toda velocidade na rede, o sistema

operacional pode limitar essa capacidade. Isso pode resultar em respostas mais lentas, ou a falta da habilidade de aceitar novas conexões em um *nodo* individual do *cluster*;

- limitação do Sistema Operacional (SO) de um nodo: alguns sistemas operacionais têm limitações quanto a velocidade de processamento dos pacotes, o número de conexões que podem suportar e o tipo do tráfego que podem aceitar. Redes de alta velocidade podem resultar em um grande número de interrupções quando um novo pacote chega. Alguns sistemas operacionais podem não ser capazes de processar números grandes de interrupções tão rápido quanto necessário. Além disso, podem ser limitados por múltiplas camadas dos componentes de softwares do SO, cada camada tendo que processar informação de entrada em níveis diferentes;
- limitações do dispositivo de balanceamento: todo dispositivo de balanceamento tem limitações práticas devido à memória e o tempo de processamento. Dispositivos que mantêm tabelas de informações de conexões de entrada e status do nodo limitam o tamanho do *cluster* e a taxa de processamento do tráfego dos dados. Métodos de balanceamento que trabalham bem em *clusters* pequenos podem não ser eficazes em um com grande número de *nodos*. Um outro detalhe importante sobre método de balanceamento diz respeito à utilização em cenários diferentes, o que pode não ser viável;
- tráfego baseado em sessão e sem sessão: uma sessão TCP consiste em um conjunto de pacotes com números seriais. Dispositivos de balanceamento geralmente mantém informações de sessões TCP ativas nas suas tabelas de conexão, para manter o tráfego fluindo de maneira apropriada para um nodo. Tráfego baseado em sessão é comum na maioria dos algoritmos de balanceamento. Uma sessão TCP tem início e fim marcados por pacotes como *flags* de sincronismo (TCP_SYN) e (TCP_FIN) respectivamente. Tráfegos não baseados em sessão, como o do protocolo UDP, não podem ser completamente contados. Pacotes UDPs são independentes uns dos outros e não existe necessariamente uma continuidade entre um pacote e o próximo [TANENBAUM 2003]. Mas para uma aplicação é totalmente viável receber um pacote da mesma máquina por vez. Um *cluster* que espalha pacotes UDP na rede entre seus nodos pode deixar a aplicação em um estado ambíguo. Alguns fabricantes de aplicações para *clusters* criam um sistema de *pathwork* para UDP, com o objetivo de manter informações dos datagramas de entrada de uma origem e estabelecem um tempo

limite para uma sessão. Quando um novo pacote UDP é recebido, o dispositivo verifica seu endereço de destino e o marca para um nodo específico. Todos os pacotes desse endereço IP de origem são recebidos depois de um certo intervalo de tempo definido pelo usuário, a sessão UDP é considerada fechada e o dispositivo remove as informações da tabela na memória. Embora não seja 100% efetivo, este procedimento no nível da camada de aplicação resolve o balanceamento de carga UDP [ALTEON 1999] ;

- dependências da aplicação: algumas aplicações necessitam que, primeiro um computador de origem tenha acessado um nodo em particular do *cluster* e assim, no futuro, novas conexões neste nodo continuam sendo possíveis Este procedimento é muito comum em *clusters* onde cada nodo é independente dos outros. Por exemplo, um usuário de uma aplicação WEB cria uma sessão com a informação pessoal daquele usuário. Embora a conexão TCP para o nodo seja transiente para cada acesso à página, o servidor WEB necessita manter registro da atividade do usuário até que o mesmo desconecte da aplicação. Esse tipo de dependência de aplicação pode ser resolvida mudando o código da aplicação para construir uma aplicação mais cuidadosa de *cluster*, embora não seja sempre possível. O dispositivo de balanceamento deve estar ciente dos requerimentos da aplicação para determinados casos, e isso afeta a escolha de algoritmos de balanceamento.

4 ARQUITETURA PARA *CLUSTER* DE ALTA DISPONIBILIDADE & ALTO DESEMPENHO

Segundo Pankaj [PANKAJ 1994] um sistema distribuído é um conjunto de processos concorrentes, em computadores diferentes, que cooperam uns com os outros para executar uma determinada tarefa. Sistemas distribuídos, geralmente, são constituídos de vários nodos que se utilizam da rede de comunicação para troca de mensagens entre os nodos. *Clusters*, como verificado anteriormente, são uma sub-classe de sistemas distribuídos que agrupa computadores/nodos de forma dedicada para atingir um objetivo específico e é visto pelos seus usuários como um sistema de recurso único, totalmente transparente. O uso de *clusters* resume-se em alta disponibilidade de dados e serviços e alto desempenho para aumentar a capacidade e velocidade de resposta da execução de uma aplicação distribuída.

Neste trabalho a proposta de alto desempenho é utilizar *clusters* para distribuir a carga de trabalho de uma aplicação crítica. Ou seja, os nodos serão configurados para atender uma demanda de trabalho a partir de definições de políticas de escalonamento definidas em um nodo especial chamado balanceador de carga. Uma tarefa não será dividida entre os nodos, mas sim alocada para um dos nodos processar. A proposta de alta disponibilidade também consiste em utilizar a estrutura de *cluster* através de seus nodos, como redundância dos recursos físicos e lógicos, para garantir maior disponibilidade do sistema de computação. A seguir é discutido um modelo bem abrangente para a implementação de *clusters* que mesclam técnicas de alta disponibilidade e alto desempenho.

De acordo com os objetivos de implementação do *cluster* neste trabalho, é necessário, inicialmente, mapear os componentes e domínio da disponibilidade. Ou seja, quão disponíveis devem ser os dados e serviços oferecidos pela aplicação crítica. Em seguida, deve-se colocar redundância suficiente nestes componentes para que situações de falha possam ser contornadas. A estratégia para este procedimento é o processo de *failover* dos recursos afetados, ou seja, há uma reconfiguração dinâmica no *cluster* a fim de que ele continue provendo seus serviços básicos e disponibilizando os dados da aplicação.

Outra especificação importante do modelo se refere ao balanceamento de carga. Na tentativa de obter um melhor desempenho dos serviços oferecidos pela aplicação crítica será configurado um nodo especial para receber todas as requisições dos usuários e distribuir essas requisições pelos nodos do *cluster* para processamento.

O principal atributo do modelo em questão se refere aos usuários da aplicação terem a impressão de que somente um único sistema responde, criando assim uma ilusão de um recurso único (computador virtual), requisito principal em sistemas distribuídos. Serão definidos nas próximas seções esquemas e estratégias para atender as linhas gerais aqui propostas.

4.1 Esquema Proposto para *Cluster* de HA e HP

O esquema proposto nesta arquitetura aborda aspectos e características de alta disponibilidade e alto desempenho para aplicações críticas. A Figura 4.1 apresenta o esboço do modelo de arquitetura adotado neste trabalho.

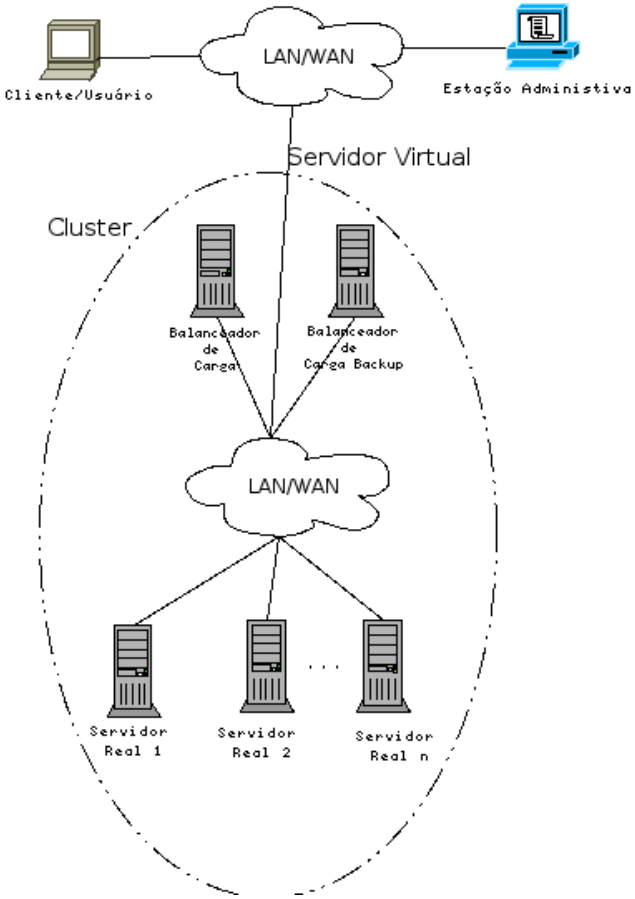


Figura 4.1 Modelo de *Cluster* HA/HP

Na Figura 4.1 destaca-se uma área externa ao *cluster* composta por clientes/usuários, uma rede para conectividade dos elementos envolvidos e uma estação especial para administração de todo o ambiente. Já a composição da área interna, que pode ser também definida como área do *cluster*, compreende uma rede para evidenciar a conectividade dos elementos, um nodo balanceador de carga, um nodo balanceador de

carga backup e *n* servidores reais.

Como a proposta dessa arquitetura pode ser entendida como um aglomerado de máquinas, todas comprometidas com um objetivo específico, devemos distinguir os elementos fundamentais envolvidos e seus papéis. Na Figura 4.1 são definidos os seguintes elementos:

- Área Externa ao *cluster*, comporta por: a) Cliente/usuário, responsável por enviar para a aplicação crítica estruturada no *cluster* a demanda de serviços; b) Estação administrativa, que tem o objetivo de servir como uma interface para o administrador do sistema de computação, de maneira que o mesmo possa interagir com os demais elementos envolvidos no ambiente, acompanhando a evolução desses e permitindo interação através de intervenções para manutenção quando for necessário; c) O elemento LAN/WAN, que evidencia a inter-conectividade entre os elementos da área externa ao *cluster* e a conectividade aos elementos da área interna ao *cluster*. Este elemento pode ser tanto uma rede local como ethernet ou gigabit ethernet ou uma rede de longa distância qualquer.
- Existe também a área Interna ao *cluster*, onde aparecem os seguintes elementos: a) Balanceador de carga, nodo do *cluster* responsável por receber as requisições de trabalho do cliente/usuário e distribuir para outros nodos a demanda de tarefas existente através de uma política de escalonamento definida previamente; b) Balanceador de carga *backup,* que possui a função de verificar através de técnicas de monitoramento o estado atual do nodo balanceador de carga, caso algum estado de falha venha a acontecer em algum serviço ou dado oferecido pela aplicação crítica, o nodo balanceador de carga *backup* assume o lugar do nodo balanceador de carga evitando que o sistema de computação entre em estado de erro ou defeito; c) LAN/WAN, elemento necessário para evidenciar a inter-conectividade entre os elementos da área interna ao *cluster* e a conectividade aos elementos da área externa ao *cluster*. Este elemento pode ser tanto uma rede local como ethernet ou gigabit ethernet ou uma rede de longa distância qualquer. Entretanto, utilizando uma LAN o desempenho será consideravelmente superior devido a computação do *cluster* utilizar alta largura de banda proporcionada por uma ethernet ou gigabit ethernet; d) E o elemento servidores reais, que são os nodos do *cluster* responsáveis por executar de fato a tarefa enviada pelo cliente/usuário. Recebem do balanceador de carga a tarefa a ser processada e após o processamento enviam a resposta para o cliente/usuário através da estrutura e organização da conectividade existente.

Em sistemas implantados de maneira distribuída, como é a proposta do *cluster* deste trabalho, a idéia de servidor virtual é importantíssima. Deve ser totalmente transparente para o cliente/usuário a existência do *cluster*. Ou seja, o cliente/usuário envia uma requisição de dado ou serviço para um servidor processar, dando a impressão de que a estrutura funciona como um sistema único.

É totalmente dispensável para o cliente/usuário saber da existência do *cluster*, o que realmente importa é a disponibilidade da aplicação e bons índices de desempenho. Para o cliente/usuário ter a percepção que o sistema é único, deve ser definido na estrutura do ambiente o responsável por receber as requisições do cliente/usuário, que neste caso é o nodo balanceador de carga do *cluster*.

De forma geral, o funcionamento do ambiente usado neste trabalho consiste no envio de requisições de serviços dos clientes/usuários para o sistema de computação.

Os clientes/usuários terão a impressão de que a requisição em questão foi enviada para um sistema único. Entretanto, as requisições estão sendo enviadas para um ambiente de *cluster* de computação distribuída. O responsável por receber essas requisições é o nodo balanceador de carga, definido como servidor virtual, que possui configurado uma política de escalonamento, discutido em detalhes na seção 4.5. O objetivo do nodo balanceador de carga é direcionar para os servidores reais as demandas de serviços recebidas dos clientes/usuários.

A partir da conclusão do processamento do serviço alocado em um dos nodos de servidor real, o resultado gerado deve ser entregue ao cliente/usuário que requisitou o serviço. Este procedimento pode ser realizado através de duas formas, principalmente. Ou através do *Network Address Translation* (NAT), que coloca o balanceador de carga como intermediário nas comunicações entre clientes/usuários e servidores reais, ou através de esquemas de roteamento que viabilizam a comunicação direta entre o servidores reais e os clientes/usuários.

Ambas as técnicas citadas serão vistas em detalhes no capítulo de Estudo de Caso. Também faz parte do funcionamento do ambiente computacional em questão um nodo especial, o nodo balanceador de carga *backup*, cujo único objetivo é verificar a situação ou estado do balanceador de carga principal. Caso alguma falha aconteça, seja de hardware ou software, o nodo balanceador de carga *backup* substitui o principal dando continuidade ao atendimento da demanda de serviços.

4.2 Domínio da Disponibilidade

Segundo Buyya [BUYYA 1999], a disponibilidade em *clusters* é bastante significativa e natural devido a redundância física e lógica. Um exemplo, citado em [WEBER 2002] apresenta que 99% de disponibilidade significa o sistema executar 24 horas por dia, 7 dias por semana, 52 semanas por ano com uma possível indisponibilidade de aproximadamente 3,65 dias para manutenção, seja programada ou não. Este resultado pode ser interessante e satisfatório para muitas empresas. Entretanto, alguns sistemas que executam aplicações críticas, tais como sistemas de tráfego aéreo, bolsas de valores, comércio eletrônico, monitoramento militar e vários outros, necessitam de um grau mais elevado de disponibilidade.

Nesses casos críticos, alguns *clusters* devem alcançar 99,999% de disponibilidade, o que significa uma indisponibilidade de cerca de 5 minutos por ano [WEBER 2002]. A dependabilidade é mais alta em um *cluster* do que a alcançada por meio de vários servidores independentes. Uma vez que o *cluster* é visto como uma única máquina, existe apenas um ambiente para monitorar [BUYYA 1999].

O objetivo da disponibilidade proposta na arquitetura deste trabalho é garantir o atendimento à demanda de serviços da aplicação crítica. O domínio da disponibilidade possui dois níveis básicos. O primeiro diz respeito à disponibilidade do nodo balanceador de carga, que apesar de não conter a aplicação crítica e não executar os serviços enviados pelos clientes/usuários, é responsável por ditar o ritmo do *cluster* alocando a carga de tarefas para os nodos servidores reais.

Como pode ser observado na Figura 4.1, são centralizadas no nodo balanceador de carga todas as requisições dos clientes/usuários da aplicação crítica, com o objetivo de simular um servidor virtual. A principal desvantagem neste caso é a dependência que o ambiente computacional possui deste nodo, ou seja, cria-se um ponto único de falha. Caso o balanceador de carga pare por algum motivo, todo o *cluster* deixaria de funcionar. Este fato é exatamente o que não se deseja em sistemas computacionais críticos. Ou seja, é indispensável o aumento da disponibilidade do servidor virtual através da inserção do nodo balanceador de carga *backup* como propõe a arquitetura da Figura 4.1

O segundo nível de disponibilidade diz respeito aos nodos servidores reais. As tarefas destinadas a aplicação crítica tratadas pelo balanceador de carga são distribuídas pelos servidores reais. Há necessidade de atendimento dessa demanda de tarefas de forma eficiente de maneira que os clientes/usuários recebam retorno de suas requisições.

Para o primeiro nível de disponibilidade, do nodo balanceador de carga, a possível solução para o problema é a adição de um nodo especial no ambiente computacional

O nodo chamado balanceador de carga Backup é inserido na estrutura e tem como objetivo acompanhar, através da técnica de verificação por temporização, a qualidade dos serviços prestados pelo balanceador de carga. Caso hajam anormalidades no funcionamento do sistema há a possibilidade de algumas decisões serem tomadas. Uma dessas decisões é a utilização da técnica de *failover*, ou seja, a substituição do nodo balanceador de carga pelo nodo balanceador de carga Backup, tentando assim dar continuidade ao atendimento da demanda de serviços.

A disponibilidade no segundo nível, na área dos servidores reais, é visível e espontânea. A proposta desta arquitetura é ter mais de um nodo servidor real para efetivo funcionamento do *cluster*, podendo assim melhorar o desempenho do processamento das requisições dos clientes/usuários. Este fato resulta indiretamente em maior disponibilidade da aplicação crítica, pois caso um nodo servidor real pare de funcionar, uma perda no desempenho do sistema pode acontecer. Entretanto, os dados e serviços do sistema continuarão disponíveis mesmo que com desempenho prejudicado.

4.3 Ganhando Desempenho com Balanceamento de Carga

Quando se projeta um sistema computacional, deve-se prever a carga média e máxima que este irá suportar. A partir do momento em que a carga de utilização do sistema começa a se tornar excessiva, é preciso buscar uma solução para o aumento da capacidade do sistema. O modelo usado neste trabalho busca exatamente isso, desempenho ideal para sistemas que executam aplicações críticas através de *cluster* para balanceamento de carga.

Para o efetivo funcionamento do balanceamento de carga, há a necessidade de implementação e implantação de um processo de escalonamento de tarefas no ambiente de *cluster*. Escalonamento é um processo de tomada de decisões que se preocupa com a alocação de recursos limitados para tarefas ao longo do tempo, e possui como meta a otimização de uma ou mais funções.

Escalonadores de tarefas são componentes de software integrados ao sistema operacional e que têm como função a distribuição de trabalho computacional para os nodos integrantes de um *cluster*, de modo a maximizar o desempenho global do processamento realizado, isto é, promover o balanceamento de carga entre os nodos de processamento envolvidos [TANENBAUM 2007].

Em sistemas homogêneos, o problema de balanceamento de carga pode ser reduzido a uma divisão de um determinado trabalho computacional em N porções iguais

e que possam ser distribuídas e executadas por N nodos de processamento do sistema, supostamente idênticos. Neste caso, o problema está fortemente relacionado à maneira de como representar o trabalho computacional a ser processado e a melhor maneira de dividi-lo em várias partes iguais.

Em sistemas heterogêneos, o problema de balanceamento de carga é consideravelmente mais complexo. Nestas circunstâncias, o escalonador de tarefas ganha especial importância. Para que o conjunto heterogêneo de nodos de processamento possa ser utilizado de maneira eficiente, questões como previsão e monitoramento de desempenho passam a integrar o problema de balanceamento de carga.

Os métodos de balanceamento de carga, vistos na seção 3.5, concentram as principais metodologias de escalonamento de tarefas desenvolvidas atualmente para balanceamento de carga com a utilização de *clusters*. Nesse contexto, existem duas linhas de desenvolvimento de escalonadores, estáticos e dinâmicos:

- estáticos: seu cálculo se faz de maneira totalmente independente da distribuição das tarefas. O escalonamento é feito em duas etapas: na primeira efetua-se o cálculo do escalonamento, ou seja, a atribuição das tarefas aos nodos de processamento (servidor real) e na segunda etapa, um mecanismo de distribuição de tarefas entra em ação para promover a distribuição previamente calculada;
- dinâmicos: pode ser entendido como a aplicação de sucessivos escalonamentos estáticos sobre estados intermediários de execução da aplicação crítica. O escalonamento dinâmico é feito de maneira concorrente com a distribuição e execução das tarefas críticas.

O escalonamento estático, apesar de simples, é implementado em larga escala em métodos de balanceamento de carga, principalmente devido a sua simplicidade implicar em robustez e facilidade de manutenção. A utilização de escalonamento estático está relacionada a sistemas homogêneos, ou seja, quando os nodos de processamento são idênticos. Um exemplo clássico é o algoritmo de Round-Robin, rotacional ou cíclico, que simplesmente rotaciona linearmente através de uma lista de nodos no *cluster* a carga a ser processada [BUYYA 1999].

O escalonamento dinâmico é utilizado geralmente quando o *cluster* possui sistemas heterogêneos, ou seja, quando os nodos de processamento não são idênticos. A metodologia dinâmica para escalonamento possui a vantagem da habilidade em lidar com grande parte das decisões de escalonamento em tempo real. Assim, atrasos ou adiamentos

no tempo de conclusão de uma determinada tarefa podem ser utilizados em tempo real para re-escalonamento das tarefas restantes a serem executadas. Entretanto o desenvolvimento desse tipo de método pode ser complexo e trabalhoso.

Seja qual for o método escolhido para escalonar as tarefas a serem executadas, o balanceamento de carga, quando bem implementado, evidência melhor desempenho para uma aplicação crítica. Um dos principais benefícios que este modelo tenta alcançar é a escalabilidade. Pois, a necessidade de melhor desempenho está relacionada a inserção de novos nodos de processamento no *cluster*.

No próximo capítulo é apresentado um estudo de caso com ferramentas e sistemas que viabilizam a implementação da arquitetura descrita. Será observado o funcionamento da arquitetura com um algoritmo de escalonamento para balanceamento de carga estático e outro dinâmico, afim de analisar e comparar o comportamento do *cluster* para cada tipo.

4.4 Sistemas de *Clusters* para HA e HP

Como discutido anteriormente, *clusters* podem ser projetados tanto para melhoria de disponibilidade quanto para melhoria de desempenho de sistemas computacionais críticos. Na maioria das vezes, ocorre uma implementação mista das técnicas de alto desempenho e de alta disponibilidade para evitar ociosidade e aumentar o número de funcionalidades do sistema.

Organizações que trabalham com desenvolvimento de soluções para sistemas computacionais críticos visam construção de *clusters* que atendam as expectativas do mercado crescente de aplicações computacionais que são vitais para as empresas.

O sistema operacional GNU/Linux, desde sua criação, trouxe muita atenção e movimentação para o mundo *open-source*. Vários projetos de software foram articulados devido ao fácil acesso ao código fonte e ao compilador do mesmo, o que resultou em autonomia para personalização de projetos de softwares com real integração a um sistema operacional.

Um grande avanço fomentado pelo GNU/Linux e pelo movimento *open-source* é a possibilidade de se usar soluções de baixo custo para missões críticas com qualidade e eficiência. Nos últimos anos, esta possibilidade tornou-se real com a criação de várias soluções de *clusters open-sources*. O estudo de caso em questão está focado em soluções que viabilizam a implementação da arquitetura apresentada na seção 4.1 para sistemas computacionais que executam aplicações críticas através de *clusters* GNU/Linux.

4.4.1 Algumas Soluções

Quando se pensa em um plano de contingência para máquinas que executam aplicações críticas, todos os esforços devem se concentrar ou em alta disponibilidade dos dados e recursos ou em alto desempenho para a execução das tarefas da aplicação crítica. Uma outra possibilidade é uma possível soma de alta disponibilidade e alto desempenho.

No universo Unix-*like*, como o próprio GNU/Linux, existem algumas alternativas livres, estruturadas em *clusters*, que visam fornecer alta disponibilidade ou alto desempenho e algumas técnicas que permitem agregar as principais vantagens das duas estratégias.

A solução de *cluster* de alto desempenho com maior visibilidade para Linux é o *Beowulf* [BEOWULF 2007]. O *cluster Beowulf* nasceu de pesquisas da NASA com objetivo de prover elevado poder de processamento. Também possui uma arquitetura multi computador que pode ser usada para processamento de algoritmos paralelos.

Beowulf é um sistema que usualmente pode ser construído usando componentes de hardware padrão como arquitetura i386, GNU/Linux, adaptadores *ethernet* padrão e *switches* de comunicação. O *Beowulf* foi um dos primeiros *clusters* de computação para alto desempenho que utilizou equipamentos e recursos comerciais populares para construção de sistemas com grande poder computacional.

Outra solução de *cluster* para Linux é o MOSIX, sigla para *Multicomputer Operating System for UnIX*. Trata-se de um conjunto de ferramentas de *cluster* para Linux, voltado ao tipo balanceamento de carga, objetivando aumento de desempenho. Uma de suas principais características é a não necessidade de aplicações e recursos de software voltados ao *cluster*, como acontece com o *Beowulf*.

O MOSIX é eficiente na tarefa de distribuição dinâmica de processamento entre os computadores do *cluster*. De maneira generalizada, o MOSIX é uma extensão para Linux (ou sistemas baseados em Unix) de um sistema de *cluster* que trabalha como se fosse um único supercomputador, por meio de conceitos de Distribuição de Processos e Balanceamento de Carga [BARAK 2008].

O primeiro *cluster* de sucesso para atender alta disponibilidade foi o sistema VAXcluster da DEC (*Digital Equipment Corporation*). Era formado por nodos VAX fisicamente conectados a um dispositivo central em uma topologia estrela. No sistema operacional, cada nodo tinha sua própria identidade, *drivers* e periféricos [WEBER 2002].

Um outro exemplo comercial de HA-*clusters* são os desenvolvidos pela SUN

Enterprise (http://www.sun.com). *Clusters* SUN possuem caminhos redundantes entre todos os nodos, entre todos os subsistemas de disco e para todas as redes externas. Uma das principais vantagens é não existir nenhum ponto único de falha – de hardware, software ou rede. O conjunto de softwares do *cluster* inclui funcionalidades de gerência de falha e gerência automática de recuperação [WEBER 2002].

Para servidores GNU/Linux, existem várias soluções baseadas em software livre. Uma com grande destaque, é a solução de *cluster* Piranha distribuído pela Red hat, que permite servidores proverem serviços de alta disponibilidade sem hardware especializado. Este *cluster* é totalmente baseado em software, com os servidores interagindo através de uma rede de alta velocidade. Pode ser configurado para alta disponibilidade ou para balanceamento de carga [REDHAT 2008] .

Um outro projeto com grande visibilidade, relacionado a *clusters* de alta disponibilidade GNU/Linux, é o linux-HA [LINUX-HA 2008]. Estima-se que mais de trinta mil implementações de sistemas de missão crítica utilizam o projeto linux-HA ou pelo alguma de suas derivações. Uma ferramenta desenvolvida no projeto Linux-HA, denominada Heartbeat permite configurar um nodo de *backup* para qualquer outro nodo em um *cluster*. Segundo Linux-HA [LINUX-HA 2008], a ferramenta Heartbeat possui um histórico de registros de *downloads* de mais 100 por dia para todo o ano de 2006.

O interesse do estudo de caso deste trabalho é implementar, testar e analisar resultados de uma solução que atenda a arquitetura descrita neste capítulo, promovendo tanto alta disponibilidade quanto alto desempenho. A solução pesquisada, que atende totalmente a arquitetura proposta é o projeto Linux Virtual Server (LVS) [KOPPER 2008]. Na próxima sessão é apresentado em detalhes a solução LVS e as personalizações realizadas para atender as especificações existentes na arquitetura proposta.

4.4.2 Linux *Virtual Server* (LVS)

Linux *Virtual Server* (servidor virtual Linux) é um projeto altamente escalável e com suporte para alta disponibilidade construído em um *cluster* de servidores reais [KOPPER 2008]. A arquitetura do *cluster* é totalmente transparente para os usuários finais, interagindo com o sistema como se fosse um único servidor virtual de alto desempenho. Exatamente o que se busca em sistemas distribuídos.

Nesta arquitetura encontram-se dois tipos de máquinas: balanceador de carga e os servidores reais. O balanceador de carga realiza o balanceamento da carga, ou seja, quando é feito um pedido para o sistema ele direciona para um dos servidores reais

posicionados no *cluster* para executar o pedido, e no próximo pedido ele direciona para outro servidor real, e assim por diante. Dessa forma a carga será executada com maior desempenho e velocidade. A grande vantagem desta solução está no fato do projeto ser de código aberto. Este fato garante maior personalização do sistema, pois é disponibilizado o código fonte para futuras contribuições da comunidade.

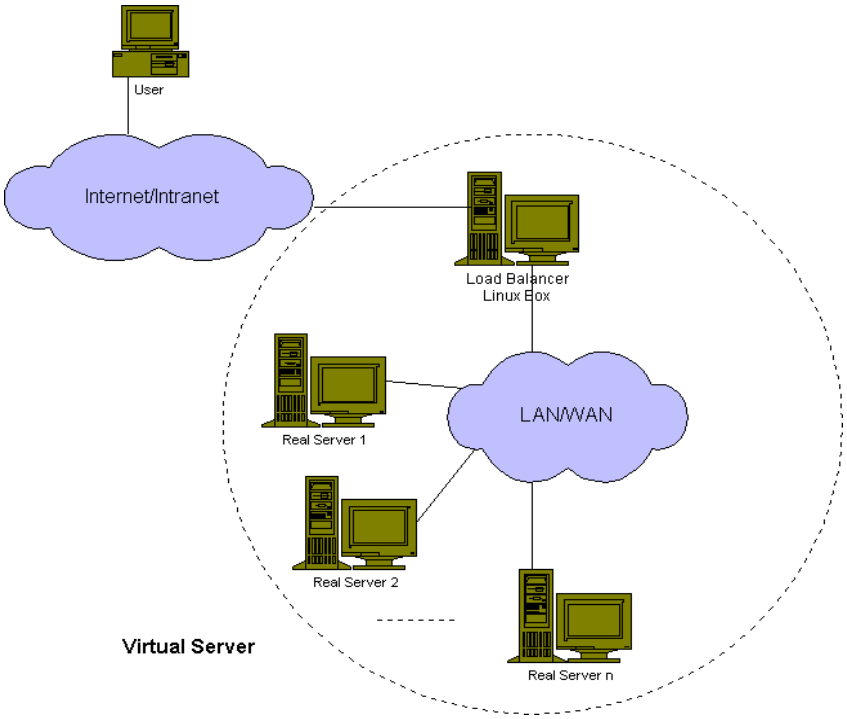


Figura 4.2 Visão geral da Arquitetura LVS [LVS, 2008]

O balanceador de carga só é possível ser implementado no sistema operacional Linux, já os servidores reais podem ser executados na maioria dos sistemas operacionais existentes, como o GNU/Linux, BSDs, Solaris, MS Windows, etc. O desempenho do LVS depende diretamente do hardware em que o mesmo está sendo executado.

A meta básica do Linux *Virtual Server Project* é construir um servidor com alto desempenho e alta disponibilidade utilizando tecnologia de *clusters* que forneça escalabilidade, confiabilidade e manutenção em alto nível. Ultimamente as principais aplicações que utilizam o LVS como resposta a necessidade de alto desempenho e alta disponibilidade são serviços de rede, tais como: servidor WEB, cache, correio eletrônico, ftp, meios de comunicação social e serviços VoIP [KOPPER 2008].

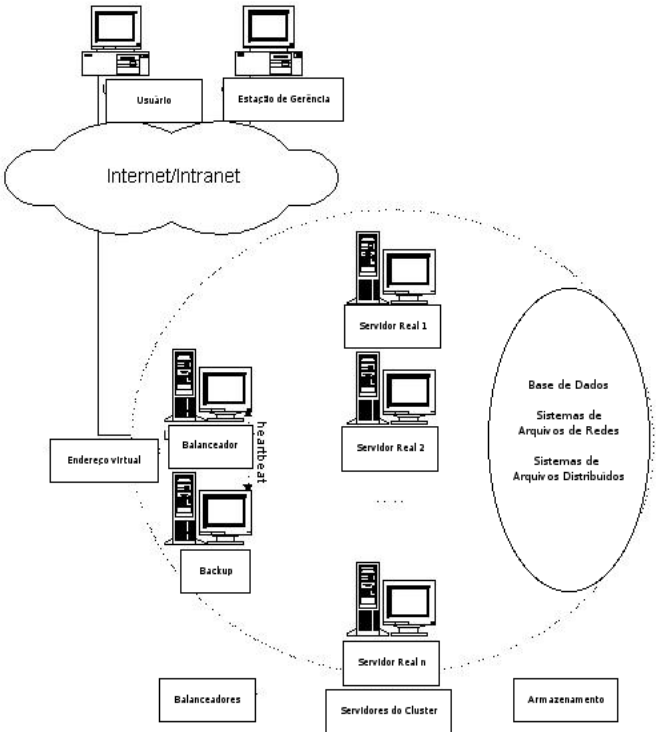


Figura 4.3 Arquitetura *Three-tie* LVS [LVS, 2008]

De acordo com LVS [KOPPER 2008], para transparência, escalabilidade, gerenciabilidade e disponibilidade de todo o sistema, geralmente, adota-se a arquitetura *three-tie* em LVS *clusters,* ilustrado na Figura 4.3.

A arquitetura *three-tie* consiste dos seguintes elementos que podem ser visualizados na Figura 4.3:

- **balanceador de carga***:* máquina de interface de todo o sistema de *cluster.* Realiza a distribuição dos pedidos dos clientes sobre um conjunto de servidores aglomerados, de modo que os clientes consideram que todos os serviços podem ser acessados a partir de um único endereço IP;
- **servidores reais***:* conjunto de servidores que executam serviços reais de rede. Tais como: WEB, Mail, FTP, etc;
- **Armazenamento Compartilhado***:* proporciona um espaço para o armazenamento compartilhado, dos dados e serviços da aplicação crítica, para os servidores posicionados na área do *cluster*. O objetivo é facilitar o acesso aos dados comuns.

O balanceador de carga é o único ponto de entrada ao sistema de *cluster* que possui a aplicação crítica e pode ser implementado a partir das seguintes técnicas:

- ***IPVS (IP Virtual Server):*** implementa o Balanceamento de carga através da camada de transporte no interior do *kernel* GNU/Linux, também chamado comutação camada 4. Atua redirecionando os pedidos baseados nos serviços TCP/UDP para os servidores reais, e cria uma abstração desses servidores, parecendo que os serviços foram acessados a partir de um único endereço IP;
- ***KTCPVS (Kernel TCP Virtual Server):*** implementa balanceamento de carga no nível de aplicação no interior do *kernel* GNU/Linux, também chamado comutação camada 7. Devido a sobrecarga (*overhead*) da camada 7, de aplicação, no espaço de usuário ser elevada, é ideal que a implementação ocorra dentro do *kernel*, a fim de evitar sobrecarga de comutação de contexto e cópia de memória entre o espaço do usuário e espaço do *kernel.*

Quando o balanceador de carga é implementado com o IPVS, todos os servidores são obrigados a prestar os mesmos serviços e conteúdos, o balanceador de carga encaminha uma nova requisição de cliente para um servidor real especificado de acordo com um algoritmo de escalonamento de carga. É indiferente qual servidor recebeu a requisição, a resposta deve ser a mesma. Entretanto, quando o balanceador de carga utilizado for o KTCPVS, os servidores podem possuir conteúdos diferentes, o *balanceador de carga* pode encaminhar uma requisição para um Servidor específico de acordo com o teor da requisição

O número de nodos de Servidores no *cluster* pode ser mudado de acordo com a carga que o sistema recebe. Quando todos os servidores estão sobrecarregados, novos servidores podem ser adicionados, aumentando assim a carga de trabalho que poderá ser manipulada no sistema.

O armazenamento compartilhado pode ser um sistema de base de dados, um sistema de arquivo de rede ou um sistema de arquivo distribuído. Os dados que necessitam atualizações dinâmicas podem ser armazenados em sistemas de base de dados, quando um nodo servidor lê ou escreve dados no sistema de base de dados em paralelo, ou seja, de maneira concorrente, o sistema de base de dados pode garantir a consistência dos dados acessados concorrentemente. Os dados estáticos, geralmente, são mantidos em rede, como o sistema de arquivo NFS (*Network Files System*), de tal forma que os dados podem ser compartilhados por todos os nodos servidores do sistema. No entanto, a escalabilidade é comprometida devido ao número de nodos de servidores limitados para garantir *overhead* controlado. A forma de armazenar os dados da aplicação crítica é particular de cada sistema. Há possibilidade também de armazenar os dados nos

próprios servidores, o que evidencia *overhead* menor devido a diminuição de tráfego de requisições para acesso aos dados. Entretanto, neste caso, é importante viabilizar sincronização dos dados existentes nos servidores para eficácia das operações realizadas com a aplicação crítica, garantindo principalmente integridade e consistência dos dados.

Os elementos da arquitetura *three-tie*, geralmente, são conectados por uma rede de alta velocidade, como uma *Ethernet* ou uma *Gigabit Ethernet*, de modo que a rede não seja o gargalo do sistema quando o mesmo crescer.

4.4.2.1 Alta Disponibilidade e Balanceamento de Carga (LVS)

Uma das vantagens de sistemas baseados em tecnologias de *clusters* é a redundância de hardware e software, isto por que este tipo de sistema consiste em um número de nodos independentes, e cada um executa uma cópia do sistema operacional e dos softwares aplicativos do ambiente computacional.

A alta disponibilidade pode ser alcançada através da detecção do nodo ou subsistemas de controle em estado de falha e a reconfiguração dos mesmos de maneira conveniente para que a carga de trabalho atual seja executada nos nodos restantes no *cluster*.

A Figura 4.4 mostra o princípio de trabalho para alta disponibilidade com LVS. Este princípio, de forma geral, consiste na existência de serviços para monitoramento de *daemons* (subsistemas de controle) executados no servidor balanceador de carga, com o objetivo de verificar periodicamente se o comportamento da aplicação crítica bem como seus recursos e serviços estão dentro da normalidade, ou seja, atendendo às especificações originais.

Se não houver resposta para uma requisição de serviço ou também chamado ICMP ECHO_REQUEST de um servidor em um determinado tempo, o monitor do serviço irá considerar que o servidor está parado e remove-o da lista de servidores disponíveis para processamento da carga de trabalho existente no servidor balanceador de carga. Quando o monitor de serviços detectar que o servidor parado voltou a responder, imediatamente o monitor adiciona-o na lista de servidores disponíveis para processamento da carga de trabalho. Por isso, o balanceador de carga pode, automaticamente, mascarar as falhas de serviços de *daemons* ou servidores no sistema de computação. Além disso, os administradores do sistema podem também utilizar ferramentas para adicionar novos servidores com o objetivo de manter ou aumentar o desempenho global do sistema.

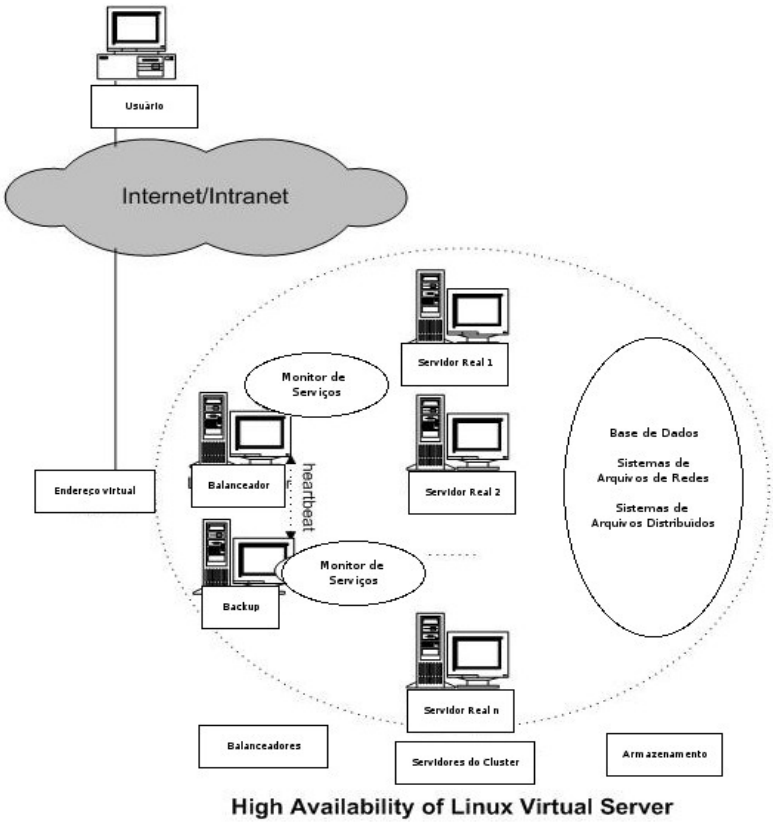


Figura 4.4 LVS com alta disponibilidade [LVS, 2008]

De acordo com a Figura 4.4 o balanceador de carga, também chamado *linux director ou* direcionador, pode ser um ponto único de falha de todo o sistema. A fim de evitar que todo o sistema computacional fique indisponível, devido a uma eventual falha no balanceador de carga, é possível e imprescindível a configuração de um *backup* (ou vários *backups*) do balanceador de carga.

Dois *daemons* da ferramenta *heartbeat* [LINUX-HA 2008] são executados, um no nodo primário e outro no nodo *backup do* balanceador de carga. A mensagem do *heartbeat* "Eu estou vivo" é propagada pela infra-estrutura de rede entre os dois servidores periodicamente. Quando o *daemon heartbeat* do nodo *backup* do balanceador de carga não puder "escutar" a mensagem *heartbeat* "Eu estou vivo" do nodo primário *balanceador de carga* no tempo especificado, o nodo *backup* assume o endereço IP virtual para prestar o serviço de balanceamento de carga. Este procedimento é denominado *failover* como apresentado na seção 3.4.3.3.

Quando o nodo primário do balanceador de carga voltar a funcionar, há duas soluções possíveis. Na primeira o nodo primário se torna automaticamente a cópia de segurança do balanceador de carga, começando a monitorar, através das mensagens

heartbeat o outro nodo balanceador de carga. A segunda solução consiste em retornar para o nodo primário o endereço IP virtual para prestar o serviço de balanceamento de carga, recuperando assim o controle do sistema e colocando o nodo *backup do* balanceador de carga novamente de prontidão para uma nova falha eventual do nodo primário de balanceamento de carga. Durante o processo de *failover*, o servidor primário deve repassar ao servidor backup a lista de requisições, pois sempre há uma perda no atendimento dessas. Dessa forma, o servidor backup poderá informar aos clientes que deve re-enviar os pedidos de requisição novamente para o devido processamento.

Existem para o direcionador três técnicas de balanceamento de carga por IP definidas assim:

- servidor virtual via NAT: usada para re-escrever o endereço de destino (endereço IP e o número da porta de serviço virtual), no pedido de pacotes que selecionou o servidor real, e quando ele volta para o servidor virtual, re-escreve a fonte do endereço das respostas nos pacotes, de modo que os clientes não saibam qual o verdadeiro servidor que executou o serviço;
- servidor virtual via IP *tunneling*: usada para re-direcionar o pacote destinado a um certo endereço IP para outro endereço IP entre diferentes servidores reais, e esses servidores retornam respostas diretamente aos usuários. Com esse método os servidores reais não precisam estar na mesma rede física;
- servidor virtual via *Direct Routing*: utiliza roteamento direto ao invés de Tunelamento IP. Quando é feita uma requisição de serviço para o LVS, o nodo direcionador repassa a conexão dos pacotes para os servidores reais que processam o serviço e respondem diretamente para os usuários, diminuindo o trabalho do direcionador, aumentando o desempenho da rede.

A máquina que executa o IPVS age como um balanceador de carga, que é conectado aos clientes que conhecem um único endereço IP de um serviço, e distribui, através de uma das técnicas citadas, o serviço a um conjunto de servidores que farão o trabalho.

5 ESTUDO DE CASO

O estudo de caso realizado neste trabalho tem como objetivo comprovar a viabilidade e eficácia da arquitetura do *cluster* de alta disponibilidade e alto desempenho para sistemas computacionais que executam aplicações críticas através da arquitetura apresentada no capítulo 4.

O estudo de caso em questão não é uma simulação, mas sim uma implementação de um ambiente com plano de contingência para melhoria de desempenho da execução das atividades de um sistema crítico com possibilidade de utilização de técnicas capazes de garantir a disponibilidade dos serviços, mesmo na presença de falhas.

5.1 Visão Geral

O sistema de missão crítica para estudo neste trabalho é uma aplicação que necessita do protocolo HTTP para efetivar o processamento da demanda de serviços e funcionalidades do sistema. O protocolo HTTP (*HyperText Transfer Protocol*) é um protocolo da camada de aplicação do modelo de referência TCP/IP [STALLINGS 1993].

O HTTP é responsável pelo tratamento de pedidos/respostas entre cliente e servidor na WWW (*Word Wide Web*) ou em Intranets. Este protocolo surgiu devido a necessidade de distribuição de informação na Internet, padronizando e especificando a comunicação dessas informações entre os clientes e servidores. No início de sua utilização, o procolo HTTP em sua versão 0.9 era simples, possui-a como missão básica apenas a transferência de dados no formato de hipertexto.

Devido ao avanço das linguagens de programação e a evolução do protocolo HTTP com novas funcionalidades e possibilidades, o número de aplicações que executam sobre a Internet aumentou muito na última década. Aplicações que executavam apenas em redes locais, a partir de algumas configurações e programações específicas migraram para um novo ambiente, a Internet, com maior flexibilidade devido a exigência apenas de um navegador para executar os métodos do protocolo HTTP e com maior acessibilidade devido ao fato da aplicação poder ser acessada de um ponto qualquer da Internet, independente da localização geográfica.

Essas qualidades, flexibilidade e acessibilidade, foram estratégicas para grandes corporações começarem a desenvolver suas novas aplicações voltadas para a Internet. Vários tipos de aplicações, que variam de simples portais eletrônicos a aplicações que conduzem a vida da organização tratando das principais atividades da missão da empresa.

Por este motivo, o plano de contingência proposto neste trabalho, através da arquitetura apresentada no capítulo 4, visa obter alta disponibilidade e alto desempenho para aplicações que executam o protocolo HTTP.

5.2 Solução Adotada

A Figura 5.1 resume a proposta deste trabalho, integrando a arquitetura apresentada na Figura 4.1 com a estrutura *three-tie* do *Linux Virtual Server*.

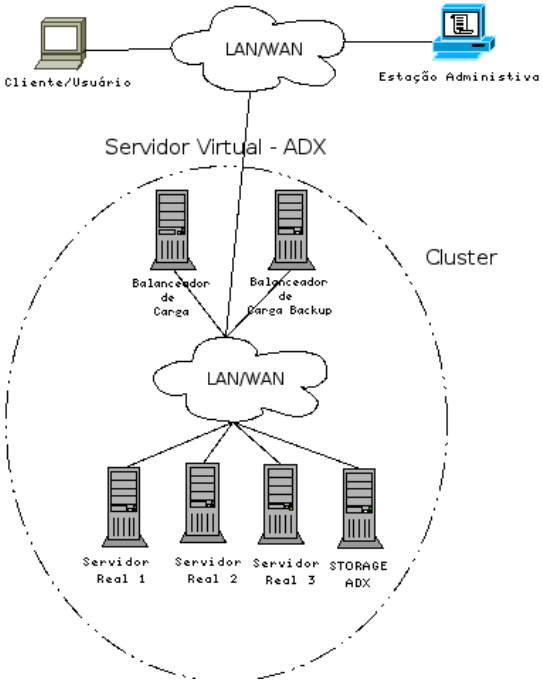


Figura 5.1 Metodologia da arquitetura adotada

Utilizando sistemas GNU/Linux e outras tecnologias, foi considerada a aplicação de gerência acadêmica de instituições de ensino superior denominada ADX, apresentada com mais detalhes na seção 5.3.2, como parte integrante do estudo de caso. Esta aplicação utiliza dos métodos do protocolo HTTP para execução de suas funcionalidades.

A escolha do sistema operacional GNU/Linux, bem como todas as demais soluções livres para o estudo de caso, se justifica principalmente pela consolidação desses sistemas no ambiente de servidores corporativos, devido ao fácil acesso aos códigos fontes e compiladores que fomentam personalização de alto nível para aplicações emergentes.

A estratégia para melhoria do desempenho e também para contornar as eventuais falhas do sistema ADX, utilizando técnicas e ferramentas específicas que viabilizem alto desempenho e alta disponibilidade, está estruturada da seguinte forma:

(a) Hardwares: os hardwares utilizados neste estudo de caso tiveram como objetivo principal o desenvolvimento de redundância de máquinas, para melhor desempenho e disponibilidade da aplicação ADX. O hardware foi dimensionado e estruturado da seguinte forma:

- ***cluster*:** 03 computadores de arquitetura *i386* de mesma configuração (processador Intel Celeron CPU 2.26 GHz e 768 Mbytes de memória RAM) foram utilizados para permitir distribuição da carga da aplicação ADX, com o objetivo de "desafogar" servidores que estejam sobrecarregados;
- **balanceador de carga:** 02 computadores de arquitetura *i386* de mesma configuração (processador Intel Celeron CPU 2.26 GHz e 768 Mbytes de memória RAM) foram utilizados para desenvolvimento do direcionador, ou balanceador da carga, para todas as requisições destinadas a aplicação crítica de comércio eletrônico. Uma das máquinas é denominada primária e a outra denominada secundária que fica em espera para um eventual problema físico ou lógico com a máquina primária;
- **armazenamento compartilhado:** 01 computador de arquitetura i386 (processador Intel Celeron CPU 2.26 GHz e 768 Mbytes de memória RAM) foi posicionado para servir como máquina de armazenamento, armazenando o banco de dados da aplicação ADX através de um sistema gerenciador de banco de dados;
- **estação administrativa:** 01 computador de arquitetura i386 (processador Intel(R) Celeron(R) CPU 2.26 GHz e 768 Mbytes de memória RAM) foi utilizado para servir de estação para gerência do ambiente computacional;

(b) Softwares: os principais softwares envolvidos no ambiente computacional são:

- **GNU/Linux:** sistema operacional para tratamento interrupções, entrada e saída, e gerenciamento de recursos de hardware e outros softwares. A distribuição utilizada é a Debian etch GNU/Linux 4.0, com kernel 2.6.18-6-686. Como este software é básico, ou seja essencial, foi utilizado em todas as máquinas especificadas no ambiente computacional deste estudo de caso;
- **APACHE:** servidor de páginas Web, será responsável por tratar todas as requisições HTTP dos clientes das páginas do sistema ADX. Este software foi instalado e configurado em todos os nodos do *cluster* com o objetivo de processamento das requisições e dos métodos do protocolo HTTP oriundos do nodo balanceador;

- **PHP:** linguagem de programação script, de código aberto e de uso geral, utilizada para o desenvolvimento da aplicação Web ADX. Este software foi instalado em todos os nodos do *cluster* para execução dos códigos do sistema ADX;
- **MySQL:** Sistema Gerenciador de Banco de Dados (SGBD) que o sistema ADX utiliza para armazenamento e manipulação das informações do sistema. Este software foi instalado no primeiro momento na máquina/nodo de armazenamento com o objetivo de centralização da base de dados do ADX e após a análise do comportamento do *cluster* foi instalado nos nodos servidores reais do *cluster* com recursos de replicação para manutenção da integridade e consistência dos dados da aplicação do ADX com objetivo de novas análises e confronto das duas situações;
- **Munin:** é um programa que produz gráficos sobre variados temas como monitoramento de processador, carga dos discos, consultas do SGBD MySQL, uso de memória, rede e outros recursos. Este software foi de grande utilidade para o monitoramento e análises do comportamento do ambiente computacional [POHL 2008];
- **MRTG:** o MRTG (*Multi Router Traffic Grapher*) é uma ferramenta baseada em interface Web para monitorar a carga de tráfego em *links* de rede. O MRTG foi utilizado para monitorar, principalmente, o comportamento das interfaces de rede de todos os nodos envolvidos no ambiente computacional [KRETCHMAR 2003];
- **HTTPERF:** é uma ferramenta para medir o desempenho de servidores WEB. Fornece um mecanismo flexível para a geração de várias cargas HTTP medindo o desempenho do servidor em questão. As três características distintas do HTTPERF são: robustez, que inclui a capacidade de gerar e sustentar a sobrecarga, suporte para os protocolos SSL e HTTP/1.1, e sua extensibilidade aos novos geradores de trabalho e desempenho medições [MOSBERG 1998]. O HTTPERF foi utilizado para realização dos testes de desempenho e disponibilidade do ambiente;
- **Softwares LVS:** um dos principais projetos LVS – *Linux Virtual Server* é o *Ultra Monkey* [HORMAN 2008] . Este projeto foi criado para garantir balanceamento de carga e alta disponibilidade de dados e serviços em uma rede usando componentes *open source* com o sistema operacional GNU/Linux. Os principais softwares que compõem a estrutura de *cluster Linux Virtual Server* com o projeto *Ultra Monkey* são *heartbeat* e *ldirector* do projeto Linux-HA [LINUX-HA 2008]. Mais detalhes serão discutidos na seção de desenvolvimento.

(c) Dados: para garantir a consistência e integridade dos dados, o sistema de arquivo indicado para desenvolver alta disponibilidade são aqueles que possuem *journaling* [NEMETH 2004] . Estes sistemas armazenam em um arquivo de *log* (registro) as ações antes de serem efetuadas. Desta forma, em uma parada no sistema, seja por pane ou queda de energia, a recuperação é muito mais rápida e promove recursos mais eficientes para recuperação de dados danificados.

(d) Espelhamento de dados: os dados devem ser espelhados(replicados) em tempo real para a completa disponibilidade do sistema, em caso de falha.

(e) Balanceamento de carga: com foco no ganho de desempenho da aplicação ADX, será utilizado um recurso para que a carga de trabalho da aplicação possa ser distribuída pelos nodos servidores reais do *cluster*.

(f) 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 às requisições do clientes ou usuários da aplicação ADX de forma transparente.

(g) Monitoração: o sistema deve monitorar os principais serviços do ambiente computacional para detectar uma falha, disparando assim os mecanismos de tolerância a falhas para garantir a dependabilidade da aplicação ADX.

(h) Alerta: o sistema deve alertar os responsáveis pela infra-estrutura de tecnologia da informação para que as análises e eventuais intervenções para manutenção sejam realizadas no sistema.

A seguir é apresentado o processo de desenvolvimento do trabalho de acordo com a metodologia proposta. Os softwares utilizados serão comentados com maior profundidade em momentos estratégicos para descrever como foi desenvolvido o estudo de caso deste trabalho. A metodologia discutida apresenta, de forma sucinta, as linhas gerais seguidas para o desenvolvimento do projeto que terá em determinados pontos do trabalho variações e adaptações para enriquecer os resultados esperados.

5.3 Desenvolvimento

O objetivo geral de desenvolvimento deste estudo de caso é implementar, testar utilizando baterias de testes bem definidas, monitorar e comentar os resultados da implementação da arquitetura apresentada no capitulo 4. Como objetivo específico

pretende-se comprovar que, com o uso ideal de recursos da gerência de desempenho e gerência de falhas, é possível obter melhorias, de forma generalizada, nos sistemas computacionais que executam aplicações críticas, atendendo assim expectativas de usuários, empresários e corporações que utilizam a computação e recursos tecnológicos como instrumentos principais da atividade empresarial em questão.

Como mencionado anteriormente, a aplicação crítica que se deseja obter melhoria de desempenho e disponibilidade é o sistema de gestão integrada acadêmica ADX do instituto Doctum. O ADX utiliza plataforma Web e para isso faz uso do principal recurso desta plataforma, o protocolo HTTP. A seguir são discutidos assuntos referentes a implementação de todo ambiente computacional apresentado no capítulo 4 através da metodologia da solução adotada discutida neste capítulo.

O *cluster* deste trabalho foi implementado utilizando os recursos computacionais disponíveis no laboratório de Ensino do Instituto Tecnológico de Computação (ITC) das Faculdades Integradas de Caratinga (FIC) [FIC, 2008]. Foram utilizados sete computadores deste laboratório, tentando representar de maneira fiel a solução proposta na Figura 5.1. O Anexo I apresenta os detalhes técnicos da instalação e configuração do *cluster* de alta disponibilidade e alto desempenho discutido neste capítulo.

5.3.1 Aplicação ADX como Sistema de Missão Crítica

Um importante passo na consolidação do estudo de caso deste trabalho é a definição do sistema de missão crítica a ser estudado. O objetivo específico deste estudo é a constatação da eficiência da arquitetura proposta no capítulo 4 para sistemas que são vitais para as organizações. Estes sistemas geralmente necessitam de disponibilidade máxima e desempenho ideal para tratamento das demandas de serviços existentes.

O Instituto Doctum de Educação e Tecnologia [DOCTUM 2008] possui um sistema de gestão acadêmica integrada denominado ADX. O sistema em questão foi desenvolvido para gerenciar os vários aspectos de uma instituição de ensino superior, possibilitando aos usuários funções que vão desde a inclusão de um novo curso até o requerimento de documentos oficiais pelos alunos. Ou seja, a estrutura organizacional da instituição depende fortemente do sistema ADX para controlar os processos envolvidos.

Uma das características mais importantes em sistemas de gestão acadêmica é o fato de serem softwares construídos para atender a um número muito grande de usuários, com necessidades distintas mas que exercem forte influência sobre o fluxo de trabalho uns dos outros.

O registro acadêmico tem a necessidade de gerenciar informações a respeito dos cursos, disciplinas, grade horária, turmas, calendário escolar e todo processo de matrícula dos alunos. Sem esses dados, previamente inseridos na base de dados, os alunos não poderiam ser matriculados corretamente, inviabilizando o trabalho do professor, da biblioteca, do setor financeiro e de todos os outros que prestam serviços aos alunos.

Para atender a essas peculiaridades, o uso da Web torna-se impositivo por permitir disponibilizar os serviços prestados pelo sistema de forma abrangente. O fato do ADX utilizar este tipo de tecnologia, incluindo o protocolo HTTP, faz com que a utilização do software não se limite apenas ao ambiente institucional, dando liberdade ao usuário de trabalhar em qualquer lugar, a qualquer hora.

O sistema ADX, para gestão acadêmica, atualmente atende aos nove *Campi* do Instituto Doctum de Educação e Tecnologia e as Faculdades Integradas de Caratinga. Conta, atualmente, com mais de 13.000 usuários entre alunos, professores e funcionários administrativos. As bases de dados ultrapassam 25 milhões de registros, com banco de dados na ordem de Gbytes. A movimentação do dados, seja por por operações de inserção, alteração ou consulta, passa de 200.000 registros por semana [DOCTUM 2008]. Nos períodos de maior utilização já foram registrados 1.341 usuários conectados simultaneamente.

O que define o ADX como um sistema de missão crítica é afirmativa de seus administradores e desenvolvedores que dizem: “24 horas por dia e 7 dias por semana o sistema precisa estar ativo e cem por cento operacional. Por isso, este trabalho considera a aplicação ADX como sistema de missão crítica a ser estudado. Mais adiante será especificada a arquitetura do ambiente para implantação e adaptação segundo o ambiente proposto no capítulo 4 bem como o roteiro de instalação e configuração.

5.3.2 Implantação do Sistema ADX

O sistema de gestão acadêmica ADX segue a tendência da distribuição de serviços computacionais usando o ambiente Web visando amplo alcance a todos os seus usuários. A tecnologia usada para atingir estes objetivos é muitas vezes referenciado pela sigla LAMP (Linux, APACHE, MySQL e PHP). Cada um dos softwares desempenham papéis diferentes e complementares para que o sistema execute suas funções adequadamente.

O APACHE, o MySQL e o PHP, apesar de serem softwares livres, não precisam necessariamente ser usados apenas com o sistema operacional GNU/Linux, fator que contribui significativamente para o sucesso dos três softwares em várias plataformas.

5.3.2.1 Arquitetura do Sistema ADX

O ADX não é um aplicativo de gestão de informação do tipo *standalone*, isto é, que executa todo código da lógica do negócio em cada terminal cliente. Pelo contrário, foi totalmente modelado e desenvolvido de acordo com o modelo de arquitetura cliente/servidor, onde o software é dividido basicamente em dois ou três níveis.

Com esse esquema de implementação, o cliente precisa apenas ter um navegador Web capaz de "compreender" html, Java Script e CSS, possibilitando total liberdade na escolha da plataforma de hardware e software nas máquinas cliente. Além disso, o servidor APACHE, o módulo PHP e o SGBD MySQL possuem versões para as plataformas de hardware/software mais utilizadas atualmente, o que também possibilita um certo grau de desenvolvimento multi plataforma.

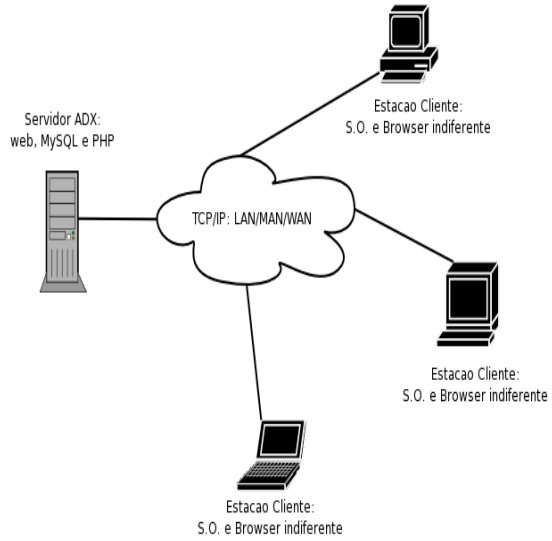


Figura 5.2 Arquitetura do Sistema ADX

Neste estudo de caso foram instalados e configurados o sistema de gestão acadêmica ADX de forma tradicional, ou seja, centralizado em uma única máquina como apresentado na Figura 5.2, e também de acordo com a arquitetura de *cluster* de alto desempenho e alta disponibilidade apresentada no capítulo 4. O objetivo é submeter as implementações a uma bateria de teste com o objetivo de verificar a disponibilidade e o desempenho, analisando, comparando e comentando os resultados. A seguir são discutidas duas possíveis configurações para a implantação do ADX.

5.3.2.2 Possibilidades de Configuração do Sistema ADX

O sistema ADX possui várias funcionalidades implementadas, quando essas funcionalidades são executadas os recursos de CPU, memória primária (RAM) e vazão de rede são consumidos. Algumas funcionalidades podem consumir mais recursos que outras, principalmente aquelas que realizam operações como consultas, inserções ou alterações em tabelas no banco de dados da aplicação.

A forma como o banco de dados da aplicação está configurado é extremamente relevante para análise deste trabalho. Em uma implantação tradicional do sistema ADX, os códigos executáveis da aplicação mais o banco de dados estão instalados e configurados em uma mesma máquina. Este fato limita o ambiente computacional envolvido, no que se refere a disponibilidade e desempenho. Exatamente por isso este trabalho tenta, através da arquitetura do capítulo 4, propor um novo ambiente com melhor desempenho global.

A metodologia da solução proposta, visualizada na Figura 5.1, apresenta um modelo diferente para a implantação do sistema ADX, onde o servidor virtural receberá todas as requisições da aplicação e distribuirá essas requisições para a área central de um *cluster* onde se encontram os servidores reais com aplicação de fato funcionando. Um dos nodos do *cluster* é definido como *storage*, ou seja, será responsável por armazenar a base de dados da aplicação ADX. Além disso, é para este nodo que todas as requisições com operações na base de dados serão destinadas.

Observe novamente a Figura 5.1 que, apesar das requisições serem processadas nos servidores reais, deverão ser redirecionados para o nodo *storage* as operações com o banco de dados. Este fato pode evidenciar altas taxas de consumo dos recursos mencionados. Uma outra possibilidade de configuração para a implantação do sistema ADX utilizando o modelo e arquitetura descritos no capítulo 4 pode ser visualizado na Figura 5.3.

Nesta configuração, o banco de dados da aplicação encontra-se implantado em cada um dos nodos servidores reais do *cluster*. A preocupação maior nesta estrutura é com a atualização dos dados entre as bases posicionadas nas diferentes máquinas. Esta configuração é possível através da utilização de técnica de replicação de bases de dados tipo mestre-escravo, entre máquinas diferente e bancos em comuns do sistema gerenciador de banco de dados MySQL.

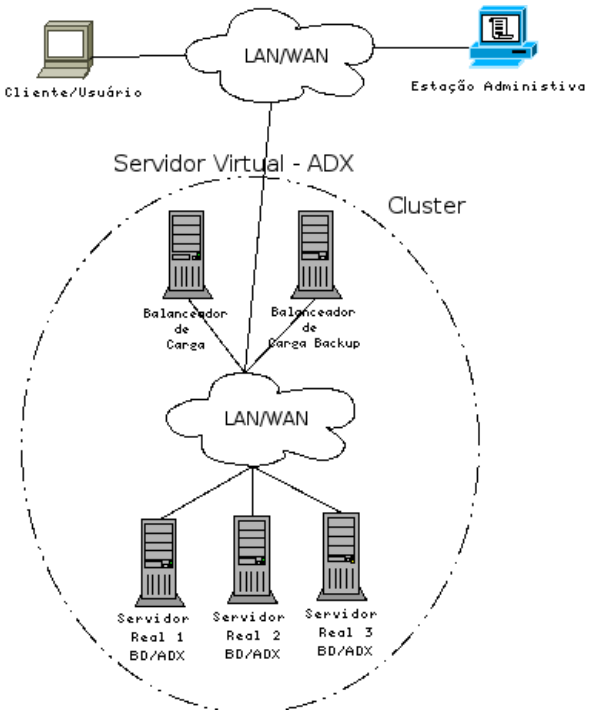


Figura 5.3 Arquitetura com BD da aplicação replicado

Nesta configuração, o banco de dados da aplicação encontra-se implantado em cada um dos nodos servidores reais do *cluster*. A preocupação maior nesta estrutura é com a atualização dos dados entre as bases posicionadas nas diferentes máquinas. Esta configuração é possível através da utilização de técnica de replicação de bases de dados tipo mestre-escravo, entre máquinas diferente e bancos em comuns do sistema gerenciador de banco de dados MySQL.

Na seção de testes, será considerado e analisado a metodologia de desenvolvimento da arquitetura do capítulo 4 com o banco de dados da aplicação ADX centralizado em um nodo como armazenamento compartilhado (*storage*). E também de forma replicada, com atualizações periódicas das bases de dados dos nodos servidores reais. Isto será realizado para comparar os resultados e definir a melhor configuração.

5.3.3 Monitoramento e Acompanhamento do Ambiente

O acompanhamento do ambiente computacional deste estudo de caso, compreendeu os aspectos de desempenho e disponibilidade da aplicação crítica. Através da gerência de redes e sistemas foi possível coletar informações sobre o desempenho do parque computacional (incluindo hardware e software) e a sua utilização, e também subsídios

para decisões relacionadas à atualização, ampliação e dimensionamento do sistema, em função do crescimento da carga, do seu comportamento temporal e da otimização do uso dos recursos disponíveis.

Todos os experimentos de medição de desempenho utilizaram o sistema sob teste, os geradores de carga de trabalho e as ferramentas de gerência que coletaram os dados de desempenho para o desenvolvimento das análises. As principais métricas utilizadas de acordo com o escopo do trabalho, para o desenvolvimento das análises, foram tempo de resposta, taxa de serviço e disponibilidade.

Durante o monitoramento e acompanhamento do ambiente computacional, os recursos de processador, memória principal e vazão de interface de redes, foram os que concentraram maior esforço para o desenvolvimento da análise e conclusões dos resultados encontrados. O objetivo principal é analisar o quanto, destes três recursos básicos, são utilizados dos nodos do *cluster*, de acordo com as condições definidas pelas baterias de testes, que serão detalhadas no capítulo 6.

5.3.4 Testes de Desempenho

A arquitetura apresentada no capítulo 4 não é um novo modelo. É, na verdade, uma proposta que soma os conceitos, recursos e tecnologias da gerência de desempenho e gerência de falhas para elaboração de estratégia para melhoria de sistemas computacionais que executam aplicações críticas.

A principal contribuição deste trabalho, para área de gerência de redes, é demonstrar o funcionamento do ambiente de *cluster,* através de implementação e execução de baterias de testes, utilizando recursos de monitoramento e análises. O objetivo dos testes, de acordo com a seção 1.3, é comprovar que, somando técnicas de alta disponibilidade e alto desempenho é possível obter melhor rendimento de um sistema computacional.

Nas próximas seções são apresentados os principais argumentos e esforços de testes no ambiente computacional desenvolvido para atender aos objetivos deste estudo de caso.

6 TESTES E RESULTADOS

Como saber se um servidor de grande capacidade, que executa uma aplicação crítica, é suficiente? O que afeta a disponibilidade do sistema? Como a carga de tarefas do sistema é atendida? As perguntas em questão são o eixo norteador para a definição e desenvolvimento da bateria de testes deste trabalho. Ao longo do texto desta seção, serão relatados os procedimentos e resultados dos testes realizados no ambiente de *cluster* para computação de alto desempenho e alta disponibilidade.

Estimar a capacidade real de um sistema de computação requer o uso de vários e diferentes tipos de análises. Existem várias técnicas e métodos para calcular a capacidade de executar a carga de trabalho imposta a um sistema de computação. Entretanto, uma maneira mais prática e simples é medir o tempo de resposta para atender uma determinada quantidade de carga de trabalho. Os principais métodos de testes utilizados neste trabalho são teste de carga de trabalho e teste de capacidade.

Os testes de carga de trabalho foram utilizados para mensurar o comportamento da resposta do sistema de computação implementado, de acordo com uma demanda de tarefas criada e designada para o ambiente computacional atender. Já o teste de capacidade é voltado para o desempenho e capacidade de resposta do ambiente computacional, no que se refere a disponibilidade do sistema continuar atendendo a demanda de serviços, mesmo na presença de falhas.

A principal preocupação na realização dos testes foi comprovar que é possível mesclar gerência de desempenho e gerência de falhas para melhorar o aproveitamento e comportamento de uma aplicação específica. Para esta constatação foram definidas as seguintes diretrizes para o desenvolvimento dos testes:

- através dos testes, deve ser possível avaliar e comparar a execução da aplicação crítica em um servidor/sistema centralizado e a execução da mesma, também, no ambiente de *cluster* de alto desempenho e alta disponibilidade;
- o ambiente de *cluster* permite o balanceamento da carga destinada à aplicação crítica entre os nodos servidores reais do *cluster* que de fato executam as tarefas. Deve ser possível visualizar a distribuição da carga através de registros e da verificação do consumo dos recursos básicos do sistema, tais como processador, memória principal e vazão de rede;
- para reforçar a melhoria de desempenho proposto através da implementação do *cluster*, deve ser possível verificar a possibilidade de escalabilidade. Ou seja, ser

possível acrescentar nodos servidores reais quando for necessário aumentar o poder de execução da aplicação crítica e conseqüentemente melhorar o desempenho;

- quanto a alta disponibilidade da aplicação crítica, os testes devem realizar esta verificação em dois níveis:
  - através da possibilidade de escalabilidade, no nível do servidores reais, deve ser constatado que mesmo com a perda de um nodo com a função de servidor real, a aplicação crítica continua sendo executada. Ou seja, a falha de um nodo servidor real pode resultar em perda momentânea de desempenho, e não em parada total da aplicação crítica;
  - no nível do servidor balanceador de carga existe um ponto crítico. Se o servidor em questão passar por uma eventual falha provocando um defeito no sistema, a distribuição das requisições à aplicação crítica deixa de acontecer e conseqüentemente a aplicação também deixa de funcionar. Deve ser possível constatar alta disponibilidade através da utilização do servidor balanceador de carga *backup,* mostrando que, apesar do nodo em questão estar desativado, ele esta pronto para assumir a posição do servidor balanceador de carga primário. Ou seja, atender as demandas dos clientes e usuários do sistema.

6.1 Definições

Para o desenvolvimento dos testes deste estudo de caso, as principais definições podem ser resumidas em ambiente de desenvolvimento dos testes e a definição da baterias de testes.

O ambiente dos testes foi o mesmo ambiente de implantação descrito na seção 5.2. As principais característica deste ambiente, que merecem mais detalhes, são: a rede de comunicação de dados e o tipo de computador utilizado.

A rede de comunicação de dados utilizada, tanto na implantação do ambiente quanto no desenvolvimento dos testes, foi uma rede local padrão ethernet de 100 Mbps. A rede foi totalmente isolada com o objetivo de evitar qualquer tipo comunicação indesejada que pudesse interferir na análise dos resultados dos testes.

Os computadores utilizados para o estudo de caso são de arquitetura i386 e possuem configurações idênticas, descriminadas também na seção 5.2. Este fato é importante citar e evidenciar para que os resultados dos testes possam ter conclusões

pertinentes ao tipo de arquitetura e configuração dos recursos de hardware utilizados.

A principal definição para os testes de desempenho foi a página principal ou inicial da aplicação crítica (ADX) ser requisitada/acessada durante duas horas, através de rajadas de um mil requisições, em intervalo de tempo de um minuto, utilizando para isto a ferramenta de *benchmark hettperf*. As mil requisições são feitas de forma consecutiva, não dependendo de tempo de resposta ou outra variante qualquer, utilizando o *banchmark* HTTPERF. Isto significa que durante duas horas a aplicação será acessada cento e vinte mil vezes.

6.2 Testes de Balanceamento de Carga e Alta Disponibilidade

Uma tarefa importante, no processo de implantação do *cluster* de alta disponibilidade e alto desempenho para o protocolo HTTP e conseqüentemente para a aplicação crítica ADX, é a validação de todo ambiente. Ou seja, constatar que o ambiente está funcionando corretamente.

Para validar se o *cluster* está funcionando em perfeitas condições, várias verificações preliminares foram realizadas, tais como testes básicos de hardware e software do sistema, além de certificação de operacionalidade da rede de comunicação de dados que interconecta o ambiente de *cluster* com outras redes de possíveis clientes.

Com objetivo de demonstrar a eficiência do ambiente, foram desenvolvidos dois testes básicos para a validação através de execução da aplicação crítica. Como a aplicação crítica é baseada no protocolo HTTP da camada de aplicação, foram construídas páginas Web de teste para cada nodo servidor real. Essas páginas nada mais são do que um texto simples informando o nome dado a cada servidor real. Foram nomeados servidor 1, servidor 2 e servidor 3, para cada nodo servidor real. Quando um cliente realizar uma solicitação de acesso ao servidor virtual, que neste primeiro teste se resume apenas em uma página Web, o cliente recebe no seu *browser* o texto informando o nome do servidor acessado.

Foram realizados dois testes básicos para a validação. Após o desenvolvimento das páginas, utilizou-se um cliente configurado para pertencer à outra rede, sendo "invisível" para os servidores posicionados no *cluster* e vice versa. No primeiro processo de validação, utilizou-se o navegador Lynx. O Lynx é um navegador executado em modo texto em um terminal. Este navegador foi utilizado por não armazenar cache e dar a falsa impressão que estava acessando o mesmo servidor. O comportamento e resultados obtidos são detalhados a seguir.

O primeiro teste de validação do ambiente consistiu na execução de três requisições de acesso às páginas Web através do servidor virtual e utilizando toda a estrutura configurada no nodo direcionador/balanceador de carga. O resultado obtido em cada uma das requisições foi o navegador cliente visualizar os nomes de todos os servidores reais. Neste caso ficou comprovado que o *Linux Virtual Server* e o balanceador de carga estão funcionando corretamente, pois o navegador cliente recebeu as respostas das requisições da maneira como era esperado. Para confirmar esta afirmativa retiramos em tempo de execução um extrato do comportamento do balanceador de carga ativo. Foi observado no registro da execução e funcionamento do nodo balanceador de carga que a requisição HTTP chegou até o servidor virtual representado pelo endereço de rede 192.168.0.254 e foi utilizado pelo balanceador de carga uma espera circular para a lista de requisições, as quais foram re-direcionadas para processamento nos servidores reais 1, 2 e 3 representados, respectivamente, pelos endereços de rede 192.168.0.200, 192.168.0.201 e 192.168.0.202.

O objetivo do segundo teste foi validar o comportamento dos servidores de balanceamento de carga e reais em uma circunstância com mais requisições. A idéia básica foi fazer com que o servidor virtual recebesse mil conexões HTTP, utilizando o escalonador RR (Round-Robin) para direcionar a carga de trabalho para os servidores reais. Para efetivar o teste foi utilizado o *benchmark HTTPERF*. A ferramenta em questão é utilizada para medir o desempenho de servidores Web. Ele fornece um mecanismo flexível para a geração de trabalho e de várias requisições HTTP para medir o desempenho do servidor [MOSBERG 1998]. O trabalho principal do balanceador é direcionar ou distribuir as mil requisições para os nodos servidores reais posicionados no *cluster*.

O resultado da execução do comando HTTPERF com os parâmetros para que o ambiente atendesse as mil conexões foi o esperado. De acordo com o monitoramento, foi possível constatar que as mil conexões foram atendidas perfeitamente, no tempo de 0.787s.

O que interessa de fato, para a validação do ambiente de *cluster* de servidores, é o funcionamento eficiente do balanceamento de carga, ou seja, a distribuição das mil conexões entre os servidores reais. Foi monitorado o comportamento do balanceador de carga ativo e constatado o registro que as mil requisições HTTP chegaram até o servidor virtual. Foi utilizado pelo balanceador de carga uma espera circular para as requisições, as quais foram re-direcionadas para processamento nos servidores reais. Através do monitoramento, constatou-se que o servidor real 1 processou 334 requisições, o servidor

real 2 processou 333 requisições e o servidor real 3 também processou 333 requisições. Através dos registros do funcionamento do ambiente foi possível comprovar a eficiência do ambiente de *cluster* para computação distribuída com foco em otimização de desempenho.

A seguir são apresentadas as ações para execução de todos os seis testes realizados bem como os resultados e comentários das análises dos resultados encontrados.

6.2.1 Teste 1 – Ambiente Centralizado

Nesta primeira bateria de teste a aplicação crítica, ADX, foi instalada e configurada em apenas um servidor. Este fato caracteriza uma implantação convencional e tradicional de sistemas computacionais, onde a aplicação está operacional para atendimento a demanda de serviços e execução de suas funcionalidades em apenas uma máquina.

Este tipo de implantação de sistemas evidencia várias desvantagens. A principal delas é ser um ponto único de falha. Ou seja, caso o servidor em questão pare por uma eventual falha, erro ou defeito, a aplicação ADX para de funcionar por completo, deixando assim de atender a demanda de serviços vigente.

Foi utilizado, na execução desta bateria de testes, o servidor de número IP 192.168.0.200 para a configuração em questão. O teste consistiu em acessar a página principal da aplicação ADX, utilizando as definições citadas anteriormente.

A intenção de executar esta bateria de testes é fazer o levantamento do consumo dos recursos básicos do sistema de computação, tais como uso do processador, memória e rede, para compara-los com o consumo da configuração do *cluster* de alta disponibilidade e alto desempenho. Assim poderemos concluir, para a aplicação ADX, que tipo de configuração tem melhor desempenho.

A seguir são apresentados gráficos de comportamento do uso dos recursos de processador, memória e rede da aplicação da máquina com o ADX centralizado.

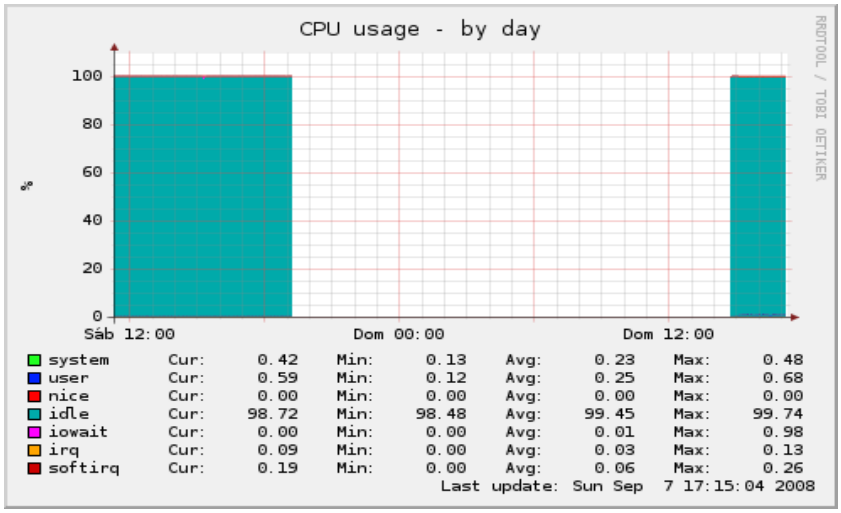


Figura 6.1 Consumo de CPU, Teste 1

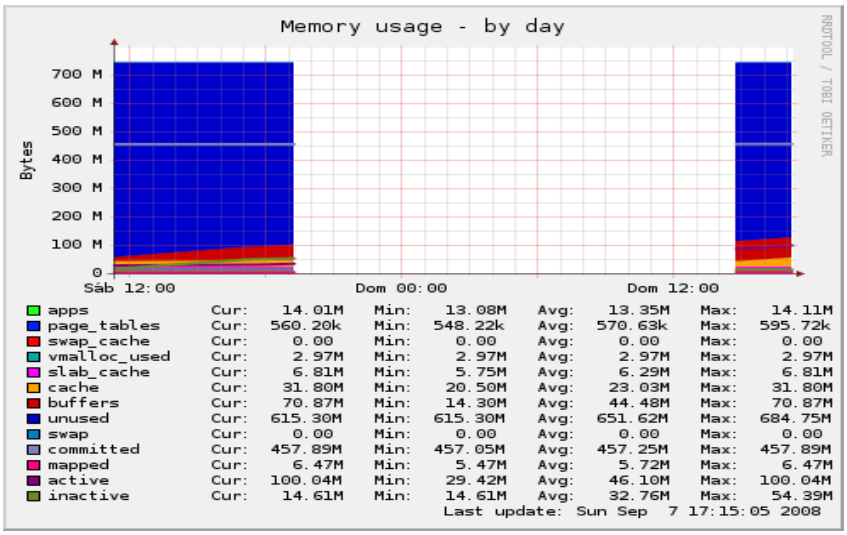


Figura 6.2 Consumo de RAM, Teste 1

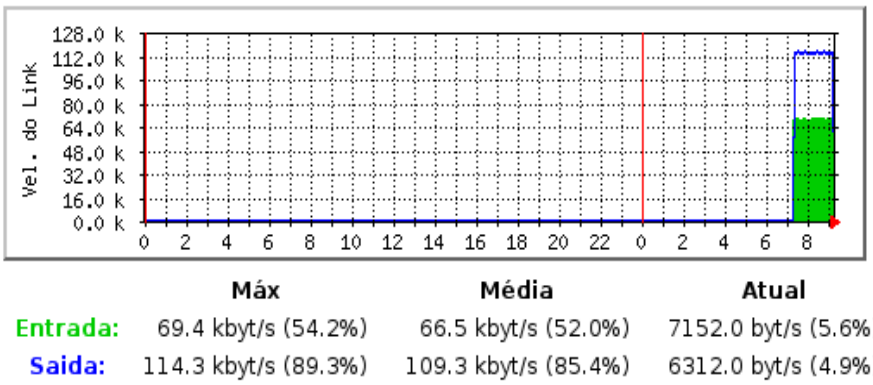


Figura 6.3 Consumo da Rede, Teste 1

A Figura 6.1 apresenta um gráfico da utilização do processador durante a execução do teste 1 para a aplicação ADX centralizada em apenas um servidor. A leitura do gráfico mostra que as 120.000 requisições utilizaram apenas 1,29% dos recursos do processador, incluindo uso de instruções e dados de usuário, sistema, interrupções de entrada e saída, etc. Ou seja, 98,71 % do processador ainda ficou ocioso.

A Figura 6.2 apresenta as informações de uso da memória principal, durante a execução do teste 1. No gráfico, fica fácil visualizar que mesmo alocando recursos de memória primária para as 120.000 requisições, 615,30 MBytes permaneceram ociosos. Ou seja, como o tamanho da memória utilizada pelo sistema de computação é 768 MBytes, 80,12% de recursos da memória continuam sem utilização.

Embora a aplicação ADX tenha sido submetida a um número grande de requisições, grande parte dos recursos do processador e da memória ainda continuam à disposição. Isto ocorreu devido às características de complexidade e custo de processamento da página inicial da aplicação ADX, a qual foram submetidas as 120.000 requisições. A página principal executa os métodos GET e POST do protocolo HTTP apenas para a construção de uma seção de página inicial da aplicação, contendo imagens, textos e barras de menus da aplicação. Ou seja, o código da página inicial não possui grande complexidade que evidencie processamento e uso demasiado de recursos de memória.

Estes resultados não evidenciam necessidade de desenvolvimento de técnicas de balanceamento de carga, com intuito de melhorar desempenho. Entretanto, nos fornece subsídios para nossas análises.

Já a Figura 6.3 apresenta um gráfico com a vazão de rede, para o tráfego de entrada e saída de dados, utilizado através da interface de rede do servidor centralizado para a execução do teste 1. A linha azul demonstra o comportamento da entrada dos dados e a linha verde demonstra o comportamento da saída de dados pela interface de rede.

Durante a execução das 120.000 requisições, o tráfego de entrada de dado foi 69,4 Kbytes/s e o tráfego de saída de dados pelas interfaces foi 114,3 Kbytes/s. Estes valores se referem ao tráfego máximo de dados pela interface de rede, o que provocou o consumo da largura de banda em 54,3% para entrada e 89,3% para saída.

6.2.2 Teste 2 – Ambiente de *Cluster*

A segunda bateria de testes foi desenvolvida a partir da implementação da solução adotada para *cluster* de alta desempenho e alta disponibilidade citada na seção 5.2. O escopo das definições desta bateria de teste é o mesmo da bateria de teste 1, exatamente, para que os resultados possam ser confrontados e analisados.

A política de escalonamento selecionada para o nodo balanceador de carga distribuir a carga de trabalho, ou seja, as 120.000 requisições para o processamento nos nodos servidores reais foi *Round-Robin*. O método *Round-Robin*, rotacional ou cíclico, simplesmente rotaciona linearmente através de uma lista de *nodos* no *cluster* a carga a ser processada. O dispositivo de balanceamento designa a próxima requisição à aplicação ADX para o próximo *nodo* na lista e o rotaciona através da lista continuamente para as próximas requisições. Este método foi selecionado devido ao baixo custo de processamento e a simplicidade de implementação.

Para a bateria de teste 2, o esquema utilizado é definido na Figura 5.1 exatamente com 3 nodos servidores reais, tendo como objetivo executar as 120.000 requisições HTTP da aplicação ADX em um período de duas horas.

A seguir são apresentados os gráficos de comportamento do uso dos recursos de processador, RAM e rede da aplicação crítica durante o teste 2. A priori a intenção é apresentar o desempenho do ambiente mediante análise de consumo dos recursos envolvidos. Posteriormente, será desenvolvida uma análise comparando os resultados do teste 2, processamento distribuído, com os resultados do teste 1, processamento centralizado.

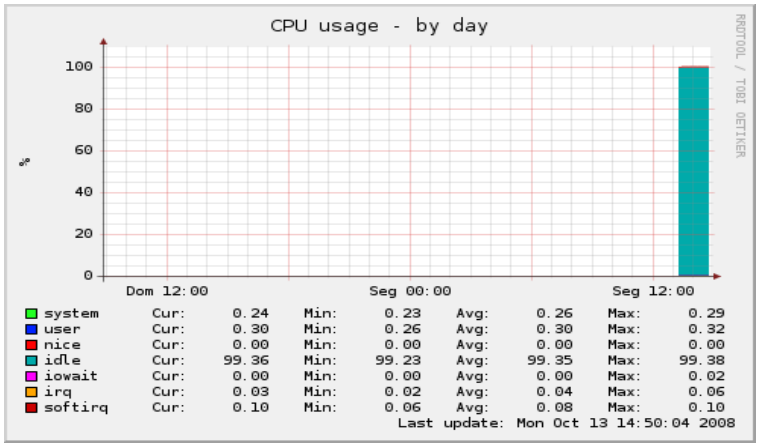


Figura 6.4 Consumo de CPU do servidor real 1 (*cluster)*

O consumo dos recursos dos processadores dos três nodos servidores reais do *cluster* após a execução do teste foi basicamente o mesmo: 0,67% para o processador do servidor 1, apresentado na Figura 6.4, 0,71% para o processador do servidor 2 e 0,69% do servidor 3.

Comparando os resultados encontrados com os resultados do teste 1, constata-se que utilizando o *cluster* de alta disponibilidade e alto desempenho o consumo de recursos do processador é menor. No ambiente centralizado o consumo do processador foi de 1,29%, já no ambiente de *cluster* o consumo foi de no máximo 0,71%.

Comparando os gráficos de consumo dos recursos do processador do ambiente de *cluster* com os do ambiente centralizado fica fácil visualizar que no ambiente de *cluster* são menos requisições a serem processadas e menos tempo de uso dos recursos do processador. De acordo com o monitoramento do comportamento do *cluster* de um dado momento, durante um minuto para a execução de mil requisições, comprova-se que o número médio de requisições atendidas é dividido exatamente por três e dessa forma ocupando menos recursos de processamento.

Este fato evidenciou um menor número de requisições atendidas simultaneamente. Enquanto o servidor centralizado atendeu a 1000 requisições, cada nodo servidor real do *cluster* atendeu em média 333. Isto implica em uma entrega de respostas mais rápida ao usuário/cliente devido a menor concorrência pelos recursos.

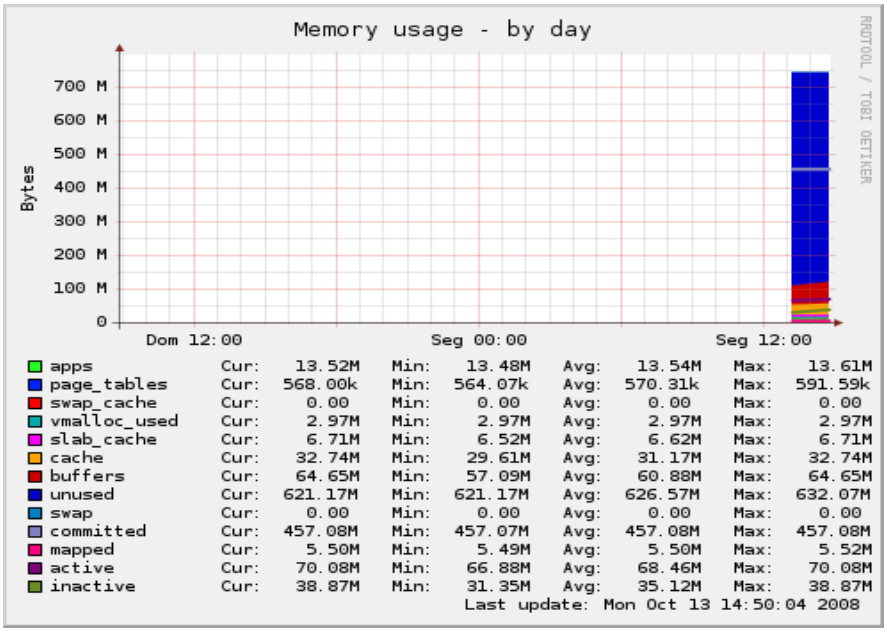


Figura 6.5 Consumo de RAM do servidor real 1 (*cluster)*

O consumo dos recursos dasmemórias primária dos três nodos servidores reais do *cluster* também tiveram um comportamento semelhante. Os recursos ociosos foram de 621,17 MBytes para a memória do servidor 1, apresentados na Figura 6.5, 605,72 MBytes para a memória do servidor 2 e 624,45 MBytes para a memória do servidor 3. Isto significa que, como o tamanho da memória é de 768 Mbytes, o consumo de recursos de memória para os três nodos servidores reais foram 11,12%, 21,14% e 18,7% respectivamente.

Comparando os resultados com os resultados do teste 1, do servidor centralizado, constatamos que a quantidade de recursos da memória primária utilizados foram praticamente iguais para ambas as baterias de testes 1 e 2. O servidor centralizado teve consumo de 19,88% enquanto a média do consumo dos três nodos servidores reais foi 16,98%. Apesar da média da quantidade de recursos de memória do nodos servidores reais ser menor, este fato não provoca impacto direto ao tempo de respostas das requisições.

Entretanto, observando e comparando os gráficos de consumo de memória do ambiente centralizado com o ambiente de *cluster*, pode-se concluir que o ambiente centralizado ocupou os recursos por um tempo maior, pois executa a rajada de mil requisições por minuto em apenas um servidor. Já os nodos servidores reais que executam em média 333 requisições, o tempo de ocupação dos recursos da memória foi menor.

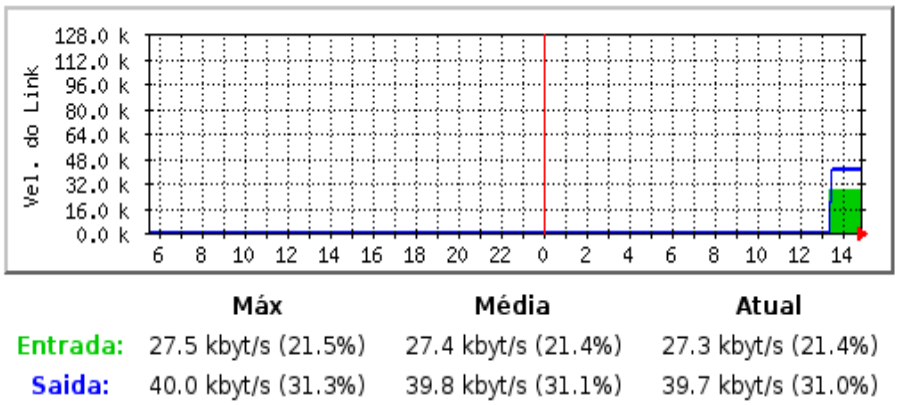


Figura 6.6 Consumo de Rede do serv. real 1 (*cluster*)

A principal característica de uma aplicação de interface Web é o uso constante e indispensável da rede de comunicação de dados para trafegar com os pedidos de requisições e respostas para execução de suas funcionalidades, utilizando para isto os

métodos GET e POST do protocolo HTTP.

A Figura 6.6 mostra o comportamento do tráfego de dados de entrada e saída, da interface de rede do nodo servidor real 1. O comportamento foi exatamente o mesmo para os três nodos servidores reais. O tráfego de entrada de dados foi 27,5 Kbytes/s, o que resulta no uso máximo de 21,5% da largura de banda disponível. Já o tráfego de saída de dados pelas interfaces foi 40 Kbytes/s, o que resulta no uso máximo de 31,2% da largura de banda disponível.

Comparando o comportamento da vazão de rede do ambiente centralizado com o ambiente de *cluster*, de forma isolada por servidores reais, tanto para a entrada quanto para a saída de dados o ambiente de *cluster* teve menor tráfego. Entretanto, quando se compara a carga média transitada em cada ambiente constata-se que no ambiente de *cluster* o tráfego foi de 82,2 Kbytes/s para entrada e 119,4 Kbytes/s para a saída de dados. Já no ambiente centralizado, apresentado na Figura 5.6, o tráfego foi um pouco menor, resultando em 66,5 Kbytes/s para entrada e 109,3 Kbytes/s para saída de dados.

Este fato evidencia que a carga média de tráfego dos dados no ambiente centralizado é menor, embora, os nodos do *cluster* tenham um consumo menor. Desta forma, pode-se concluir que no ambiente de *cluster*, apesar do tráfego ser maior, há uma melhor utilização largura de banda por máquina. Pois, existe mais máquinas na rede repassando as requisições, contudo o tráfego é menor por que cada máquina trata uma fração dos pedidos enviados ao ambiente centralizado.

6.2.3 Teste 3 – Ambiente de *Cluster* (Escalabilidade)

Alto desempenho não está relacionado apenas ao menor tempo de resposta ou ao consumo otimizado de recursos. A terceira bateria de testes mostra que alto desempenho também pode ser obtido através de escalabilidade, principalmente no que se refere a um comportamento robusto e arrojado do sistema de computação. Na prática, escalabilidade significa que o sistema pode suportar um crescente número de requisições oriundos dos usuários. Escalabilidade é um benefício enorme para aplicações críticas, como o ADX, que estão em crescimento constante no que se refere ao número de usuários.

Para a terceira bateria de testes, foram utilizadas as mesmas técnicas e configurações da solução adotada para o desenvolvimento do *cluster* de alta disponibilidade e alto desempenho, entretanto, foi adicionado um quarto nodo servidor real, de endereço IP número 192.168.0.100. O objetivo é ajudar na execução das requisições enviadas para o servidor virtual. A política de escalonamento do nodo

balanceador de carga também foi Round-Robin.

De acordo com o monitoramento do comportamento do *cluster* em um momento definido, durante um minuto para a execução de mil requisições, comprova-se que o número médio de requisições atendidas é dividido exatamente por quatro. Ou seja, 250 requisições atendidas por cada servidor real. Este fato comprova que o número de requisições atendida por cada interface de rede dos nodos servidores reais é menor que o ambiente proposto com três nodos servidores reais.

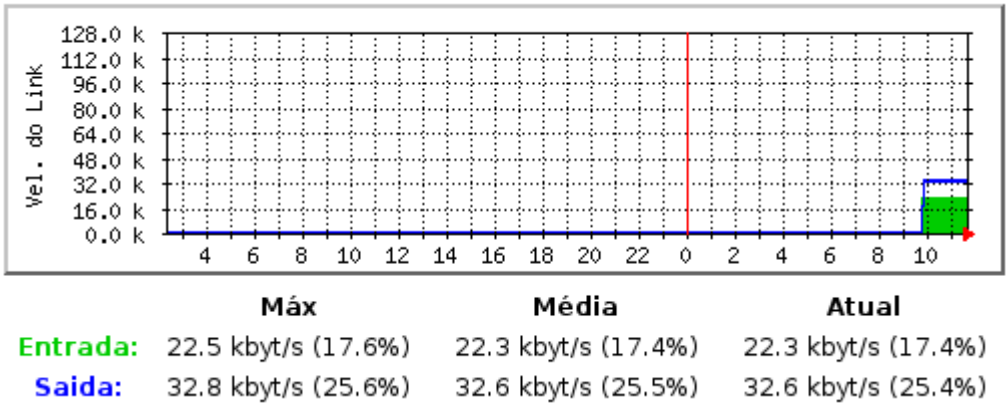


Figura 6.7 Consumo de Rede do serv. real 1, Teste 3

O comportamento se manteve estável durante quase todo o período de teste. O tráfego de entrada de dados foi 22,5 Kbytes/s e o tráfego de saída de dados pelas interfaces foi 32,8 Kbytes/s para o nodo servidor real 1 apresentado na Figura 6.7. Para o nodo servidor real 2 o tráfego foi de 23,3 Kbytes de entrada e 34,3 Kbytes de saída. Os nodos servidores reais 3 e 4 tiveram tráfego idêntico de no máximo 26,1 Kbytes de entrada e 38,5 Kbytes de saída.

Comparando o comportamento da vazão de rede do ambiente de *cluster* com três nodos servidores reais e o ambiente de *cluster* com quatro nodos servidores reais, de forma individual por servidor real, tanto para a entrada quanto para a saída de dados o ambiente de *cluster* com quatro nodos teve um menor tráfego. Desta forma, pode-se concluir que com o acréscimo de um nodo servidor real no *cluster* há uma melhor utilização e otimização da largura de banda por interface dos servidores reais, resultando assim em melhor desempenho, no que se refere a escalabilidade, para a vazão de rede.

Tabela 1: Vazão média do tráfego - Teste 3

	Centralizado	***Cluster* com 3 nodos reais**	***Cluster* com 4 nodos reais**
Entrada	66,5 Kbytes/s	82,2 Kbytes/s	89,2 Kbytes/s
Saída	109,3 Kbytes/s	119,4 Kbytes/s	130,5 Kbytes/s

Comparando a carga média de tráfego de dados dos três ambientes: centralizado, *cluster* com três nodos servidores reais e *cluster* com quatro nodos servidores reais, através da Tabela 2, observa-se que no ambiente centralizado a carga média foi menor, tanto para a saída quanto para a entrada de dados. Apesar do tráfego por interface de rede dos servidores reais do *cluster* com quatro nodos ser menor, constata-se que transita mais dados que os demais ambientes. Este fato se deve à informações de controle que trafegam no ambiente de *cluster* para auxiliar o nodo balanceador de carga a decidir para qual nodo servidor real enviar a próxima requisição.

O comportamento do consumo de recursos da memória principal e processamento, referente aos testes do *cluster* com três nodos servidores reais e com quatro nodos servidores reais, se mantiveram praticamente idênticos. O consumo de ambos foram insignificantes devido às características da aplicação e tiveram um comportamento semelhante.

Entretanto uma observação importante realizada, neste teste de escalabilidade, foi referente ao seu tempo de duração, ou seja, o tempo para entrega do processamento das rajadas de mil requisições por minuto destinada a aplicação ADX. Foi detectado que o tempo para entrega das respostas aos clientes aumentou comparando com o teste realizado para o *cluster* com três servidores reais. Devido a este fato, foi realizado a execução do teste 6, que tem como objetivo básico um teste de disponibilidade no nível dos nodos de servidores, que será mais adiante detalhado. Neste momento, a bateria de teste em questão será utilizada apenas para comparar o tempo de duração dos testes para as rajadas de mil requisições. Além disso utilizamos para efeito de análise do tempo de duração dos testes os valores encontrados para o servidor centralizado.

Os testes para todos os ambientes, centralizado, dois, três e quatro servidores reais, tiveram as mesmas configurações. Com o objetivo de comparar o tempo de duração de teste de cada ambiente, foram extraídos dez valores aleatórios dos resultados da execução dos testes, como amostra para observações.

As médias dos tempos encontrados foram 1,05s, 0,8s, 0,77s e 1,09s para os ambientes centralizado, dois, três e quatro servidores reais, respectivamente. Observa-se

que há uma redução do tempo do ambiente centralizado para o ambiente de dois e três servidores. Entretanto há um aumento no tempo do ambiente com quatro servidores reais quando comparado a todos os outros ambientes.

Este comportamento se manteve estável para dez repetições deste testes, sugerindo a conclusão de que o aumento de nodos servidores reais, no ambiente de *cluster*, pode levar a *overhead* de processamento, quando submetido ao mesmo número de requisições. Ou seja, apesar de otimizar o tráfego de dados originados pelas requisições e transitados pelas interfaces de redes do ambiente, este fato resulta em um tempo maior para a conclusão do processamento da carga de trabalho. Sendo assim, para a realização da operação de inserção de um novo nodo servidor real, o administrador do ambiente deve se preocupar com qual benefício deseja-se obter, ou tempo de duração, ou otimização da largura de banda.

Uma importante contribuição deste teste é mostrar que a escalabilidade deve ser aliada a uma análise profunda da carga de trabalho que a aplicação precisa executar, para assim, definir o melhor momento para adição de um novo nodo servidor real no *cluster*.

6.2.4 Teste 4 – Nova Carga para Aplicação Crítica Crítica

Os testes das baterias 1, 2 e 3 tiveram como objetivo o acesso à página principal ou inicial da aplicação ADX. Esta página inicial, como mencionado anteriormente, não possui grande complexidade, custo de processamento e consumo de memória primária. Devido a este fato, foi pesquisado, mediante análise da aplicação, qual a funcionalidade teria maior custo de processamento.

Uma funcionalidade na aplicação ADX, chamada gerador de relatório de descontos, que tem como objetivo relacionar o percentual de desconto do valor da mensalidade para cada aluno matriculado em todas as unidades do instituto Doctum em um único relatório, foi a selecionada. Esta funcionalidade realiza uma séria de consultas MySQL nas bases de dados que resultam em uso excessivo de processador e memória. O objetivo é analisar se o balanceamento de carga resultará em ganho de desempenho significativo, quando a funcionalidade executada realizar operações com a base de dados que evidencia grandes índice de processamento.

A seção 5.3.4.2 relaciona as possibilidades de configurações do sistema ADX. Essas possibilidades são:

1. Ambiente centralizado: aplicação e banco de dados no mesmo servidor;
2. *cluster* com armazenamento compartilhado (*storage*);

3. *cluster* com armazenamento replicado e atualização periódica.

O intuito desta bateria de testes é a execução da funcionalidade em questão em ambientes computacionais com as três possibilidades de configurações do ADX para observar o tempo de resposta, processamento, consumo de memória e vazão de rede.

Não foi possível executar a funcionalidade de geração de descontos utilizando o HTTPERF devido a características particulares do código em questão. Por isso, foi utilizado o navegador *lynx*, em modo terminal do console GNU/Linux, para evitar utilização do recurso de *cache*.

a-) Ambiente centralizado

O primeiro teste realizado no ambiente centralizado teve como objetivo a avaliação inicial do tempo que seria necessário para execução da funcionalidade uma única vez. O tempo gasto para uma única execução foi em média vinte minutos e dezessete segundos. Foi repetido dez vezes para constatação deste valor, e nunca excedeu 21 minutos.

Como o tempo para execução da funcionalidade uma vez é de vinte minutos, aproximadamente, foi estabelecido o número de seis repetições para totalizar a execução num período de no máximo duas horas. O interesse, desta definição, foi poder manter as mesmas condições das outras baterias de testes, ou seja, submeter o ambiente computacional avaliado a duas horas de execução da aplicação ADX.

Para geração de registro de informações sobre o resultado das execuções da funcionalidade, foi criado um *script* de arquivo texto para armazenar o resultado para cada repetição da execução da rotina do ADX avaliada no ambiente centralizado. Seis execuções da funcionalidade foram iniciadas simultaneamente. A formatação dos dados deste teste pode ser visualizada na Tabela 3 a seguir. O Anexo II apresenta os arquivos *scripts*, separados por testes, gerados no momento das repetições da execução da funcionalidade de geração de relatórios de descontos do ADX.

Tabela 2: Tempos de execução – Teste 4 (centralizado)

Testes	Tempo em horas
1	1,964
2	1,950
3	1,931
4	1,941
5	1,944
6	1,927
Média	**1,943**

Pode-se observar, na Tabela 3, que a média do tempo de duração dos testes foi de 1,943 hs. Ou seja, para a execução das seis repetições da funcionalidade de geração de relatório de descontos no ambiente centralizado, foram necessárias praticamente duas horas.

A seguir são apresentados os registros de gráficos de processador, memória, rede e consumo de consultas na base de dados MySQL para o ambiente centralizado durante o período de duas horas executando as repetições da funcionalidade do ADX.

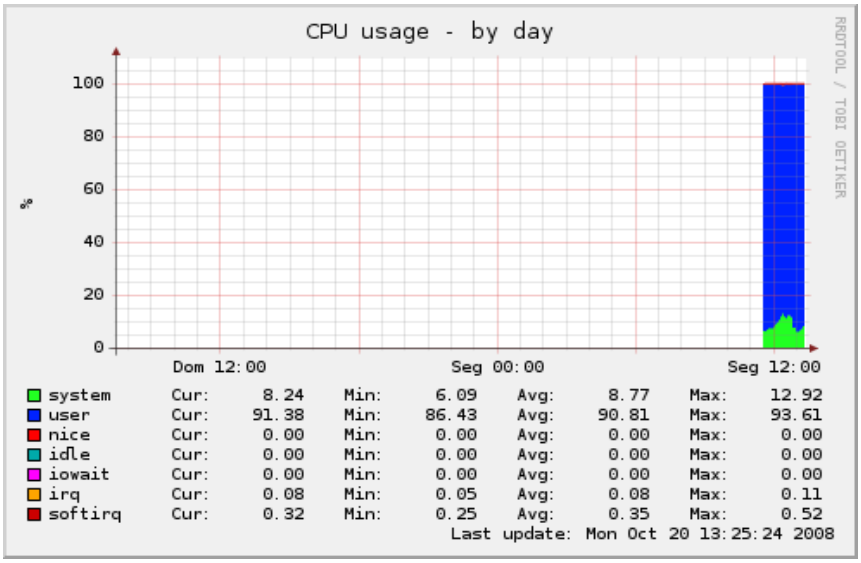


Figura 6.8 consumo de CPU, Teste 5, Centralizado

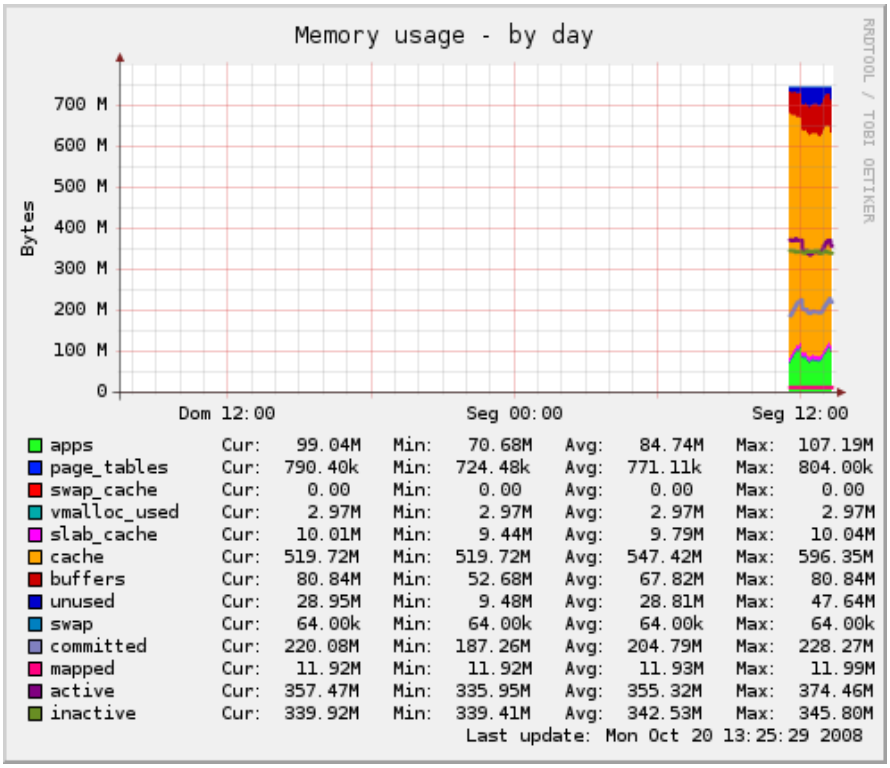


Figura 6.9 Consumo de RAM, Teste 4, centralizado

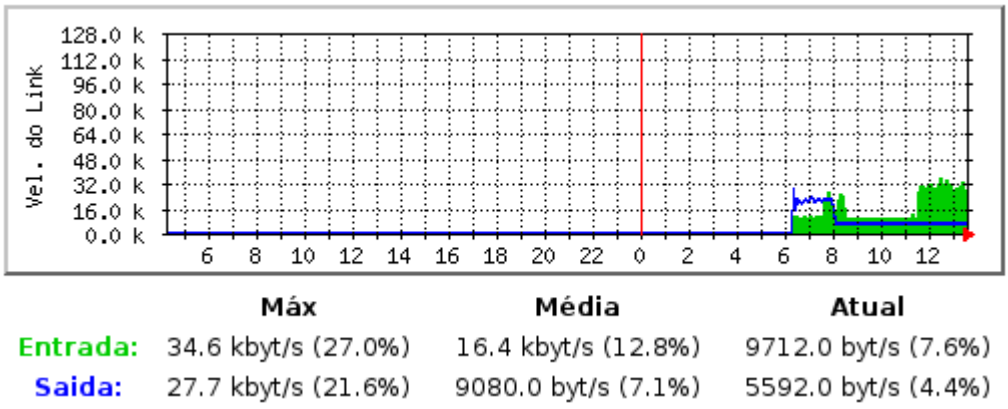


Figura 6.10 Consumo de Rede, Teste 4, centralizado

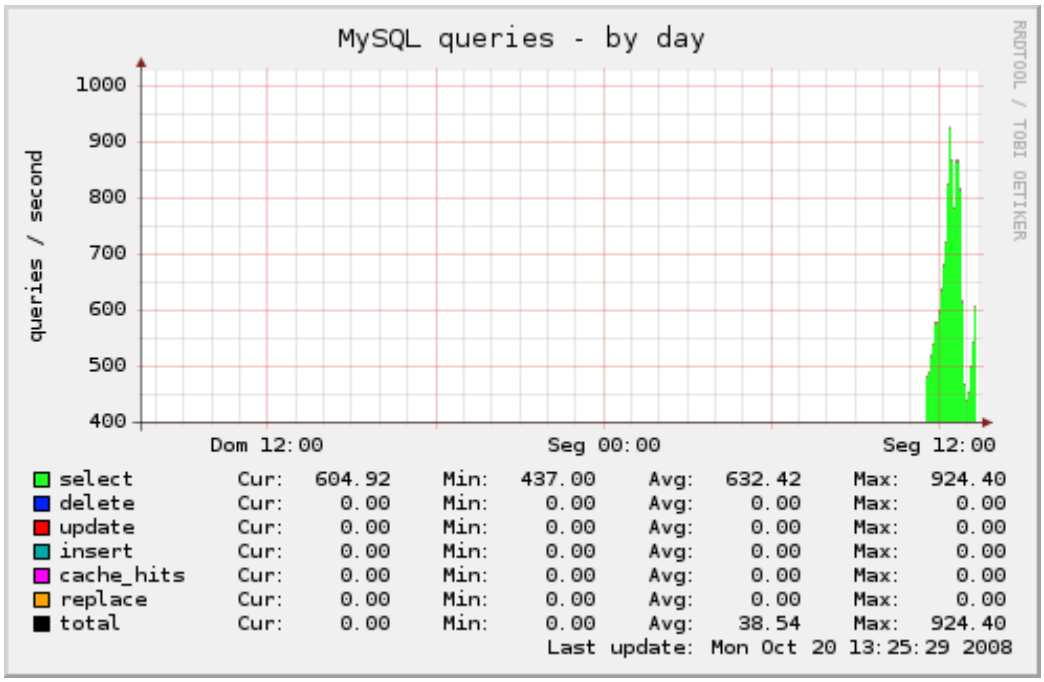


Figura 6.11 Consultas, Teste 4, centralizado

O consumo de processador e memória principal da bateria de teste 4, para o ambiente centralizado, utilizaram praticamente todos os recursos disponível, como pode ser constatado nas Figuras 6.8 e 6.9, respectivamente. Na Figura 6.8, pode ser visto que 0% dos recursos do processador estão disponíveis durante as duas horas de execução da funcionalidade de geração de relatórios de descontos. Já na Figura 6.9, pode-se verificar que apenas 28,95 Mbytes dos 764 da memória estão ociosos. Ou seja, 96,22% dos recursos da memória RAM foram utilizados durante as duas horas de execução.

O consumo de vazão de rede, apresentado na Figura 6.10, mostra o comportamento médio de 16,4 Kbytes/s para entrada e 0,908 Kbytes/s para saída de dados durante as duas horas de execução. Estes resultados serão úteis quando comparados com a vazão de rede para o ambiente de *cluster* com banco de dados compartilhado (*storage*) e replicado utilizando a nova carga de trabalho através da funcionalidade de gerador de relatório de descontos da aplicação ADX.

A Figura 6.11 apresenta o comportamento de atividades de consultas (*selects*), realizadas durante as duas horas de execução da funcionalidade de geração de relatórios de descontos do ADX no ambiente centralizado. Uma média de 632,42 consultas por segundo foram realizadas durante a execução dos testes. Este valor será comparado com o ambiente de *cluster* com banco de dados compartilhado (*storage*) e replicado.

b-) *Cluster* com armazenamento compartilhado (*storage*)

Neste tipo de configuração, apesar da aplicação ADX estar implementada no ambiente de *cluster* com três nodos servidores reais, o banco de dados da aplicação ADX está localizado em uma quarta máquina, chamada nodo de armazenamento compartilhado (storage). Ou seja, os nodos servidores reais possuem instâncias do ADX, mas toda e qualquer operação na base de dados é realizada utilizando os recursos do nodo de armazenamento compartilhado (*storage*). Esta configuração é apresentada na Figura 5.1.

Para geração de registro de informações sobre o resultado das execuções da funcionalidade, foi criado um *script* de arquivo texto para armazenar o resultado para cada repetição da execução da rotina do ADX avaliada no ambiente com banco de dados configurado em máquina de armazenamento compartilhado (*storage*). Seis execuções da funcionalidade foram iniciadas simultaneamente. O IP 192.168.0.254 é o endereço virtual do ambiente de *cluster*. O funcionamento consiste na chegada das requisições ao nodo balanceador de carga, através do endereço virtual, e este distribui essas requisições para os nodos servidores reais e para o nodo de armazenamento compartilhado (*storage*). A formatação destes dados pode ser visualizadas na Tabela 4 a seguir. O Anexo III apresenta os arquivos *scripts*, separados por testes, gerados no momento das repetições da execução da funcionalidade de geração de relatórios de descontos do ADX.

Tabela 3: Tempos de execução – Teste 4 (*storage*)

Testes	**Tempo em horas**
1	1,619
2	1,680
3	1,611
4	1,616
5	1,615
6	1,615
Média	**1,626**

Comparando os dados de tempo de execução dos testes da Tabela 3 (ambiente centralizado) com os dados da Tabela 4 (ambiente de *cluster* com nodo *storage*), podemos perceber que a média dos tempos de execução no ambiente *storage* é menor em cerca de 16%. Ou seja, enquanto no ambiente centralizado a repetição de seis vezes da execução da funcionalidade de geração de relatório de desconto gasta 1,943 horas, no ambiente de *cluster* com nodo *storage* este valor cai para 1,626 horas. Portanto, pode-se

afirmar que a execução da bateria de testes 4 no ambiente de *cluster* com nodo *storage* tem um desempenho melhor para a mesma demanda de trabalho no ambiente centralizado.

Analisando o comportamento do ambiente, podemos constatar que este resultado acontece devido aos nodos servidores reais executarem todo processamento lógico da funcionalidade que não se refere a operação de banco de dados. Já o nodo *storage* é responsável por executar todas as operações de banco de dados, como consultas (*select*) da funcionalidade de geração de relatório de descontos. Esta configuração permite uma otimização dos recursos computacionais envolvidos o que provocou melhor desempenho no que se refere a tempo de execução da funcionalidade. O monitoramento do comportamento do balanceamento de carga constatou que cada nodo servidor real executou exatamente duas requisições.

Este fato resultou em uso menor dos recursos de processador, memória, vazão de rede e operações na base de dados MySQL, devido a configuração do balanceamento da carga no ambiente. A seguir são apresentadas os gráficos que comprovam a otimização dos recursos computacionais para o ambiente de *cluster* com *storage*.

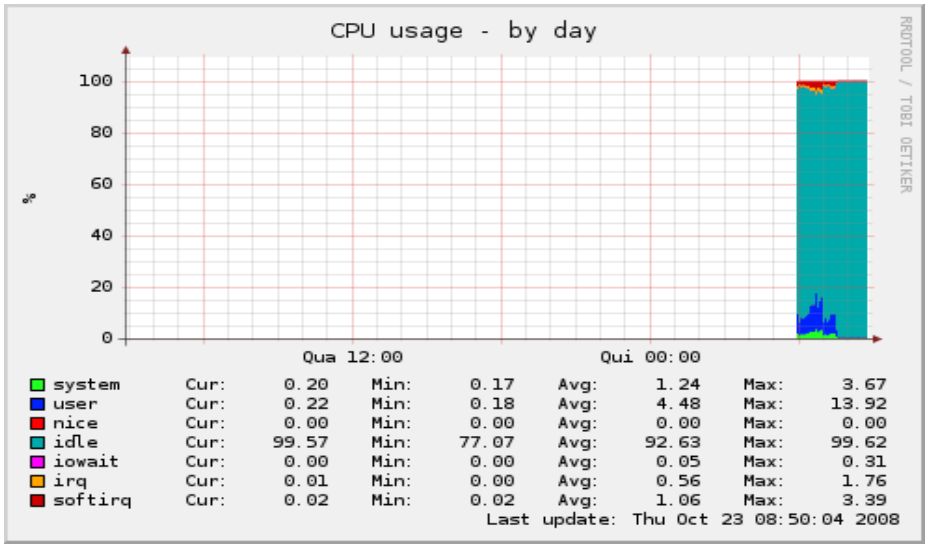


Figura 6.12 CPU do servidor 1, Teste 4, *storage*

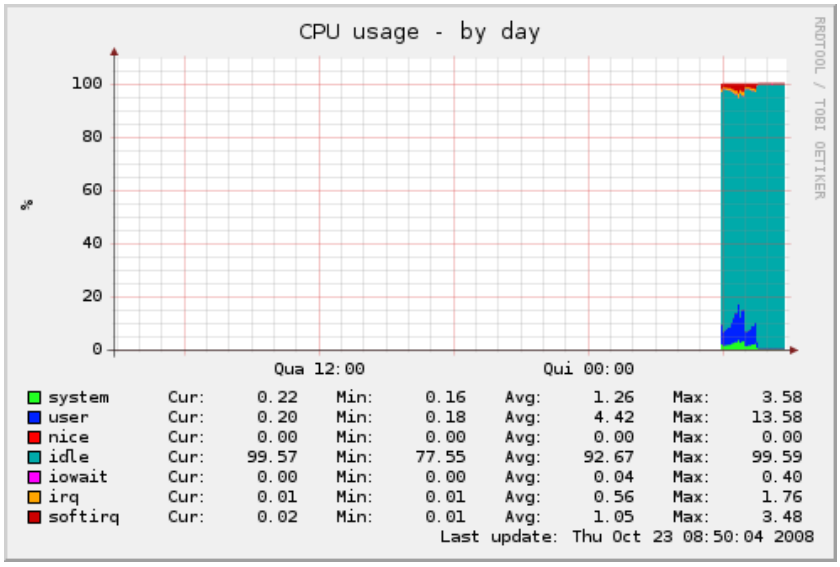


Figura 6.13 CPU do servidor 2, Teste 4, *storage*

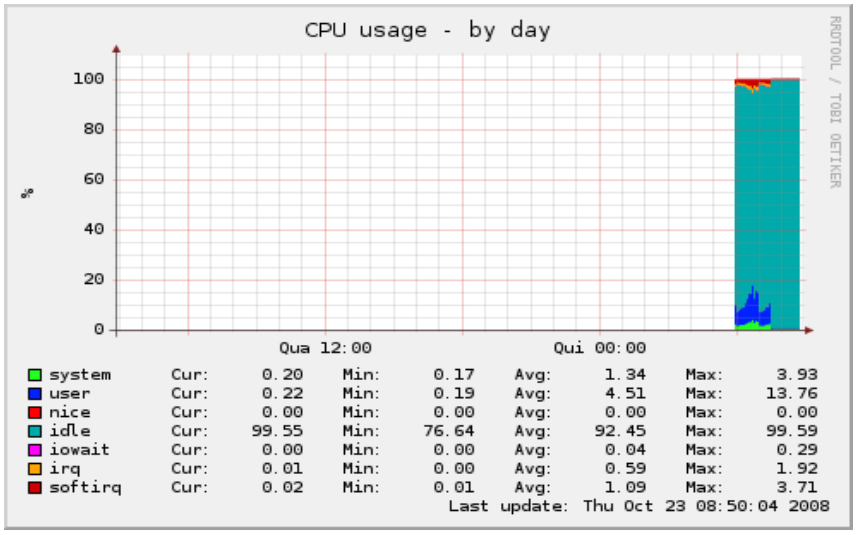


Figura 6.14 CPU do servidor3, Teste 4, *storage*

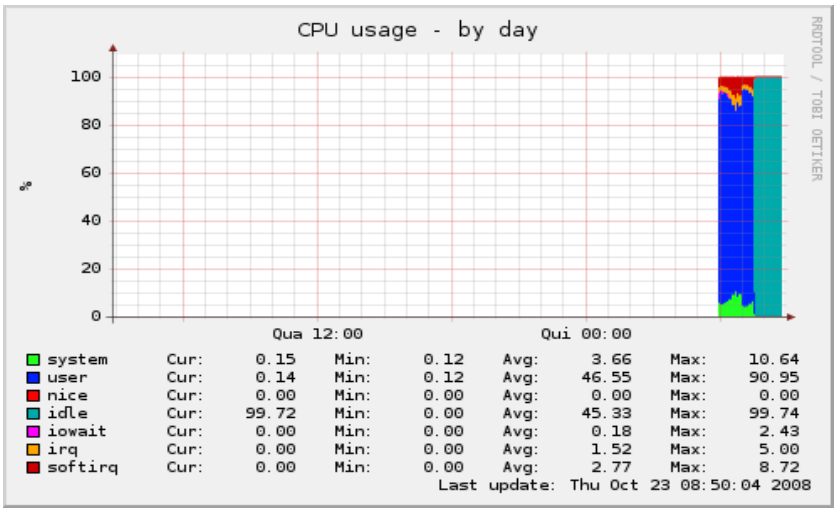


Figura 6.15 CPU do nodo *storage*, Teste 4, *storage*.

As Figuras 6.12, 6.13 e 6.14 apresentam o comportamento dos processadores dos servidores reais durante o período de duas horas de teste. Pode-se observar que o consumo dos recursos (*user* + *system*) foi pequeno. No processador do servidor 1, da Figura 6.12 o consumo médio foi de 5,72% dos recursos. Para o processador do servidor 2, da Figura 6.13 o consumo médio foi de 5,68%. Para o processador do servidor real 3, da Figura 6.14 o consumo médio foi de 5,85%. E para o processador do servidor *storage*, da Figura 6.15 o consumo médio foi de 50,21%. A somatória dos recursos de processador, utilizados no ambiente de *cluster* com armazenamento compartilhado, é de 67,48%. Isto implica numa redução do uso dos recursos do processador em média de 31,1%, visto que o ambiente centralizado gastou 98,58% dos recursos disponíveis.

A Figura 6.15 apresenta o comportamento da utilização dos recursos do processador do nodo *storage*. Ou seja, é nesta máquina onde aconteceram todas as operações MySQL da funcionalidade de geração de relatório de descontos da aplicação ADX. A funcionalidade em questão, como mencionado anteriormente, apresenta apenas a operação de consulta (*select*) nas bases de dados da aplicação. Este fato evidenciou o maior percentual de uso de processador entre os quatro servidores. Entretanto, ficou menos tempo sendo utilizado durante o período de teste. Isto aconteceu devido a distribuição da carga entre os nodos servidores reais e o nodo *storage*. Enquanto os nodos servidores reais processavam as partes do código que não necessitavam de operações na base de dados, o processador do nodo *storage* ficou ocioso. Assim que os nodos servidores reais começaram a necessitar de resultados de operações MySQL, essas foram direcionadas para o nodo *stogare*, e assim começou o início das atividades no

processador.

Pode-se começar a visualizar, através do comportamento dos processadores, que o motivo que levou a configuração do ambiente de *cluster* com nodo *storage* ter melhor desempenho de tempo de resposta foi devido a uma melhor utilização e otimização dos recursos disponíveis para atender a demanda do teste.

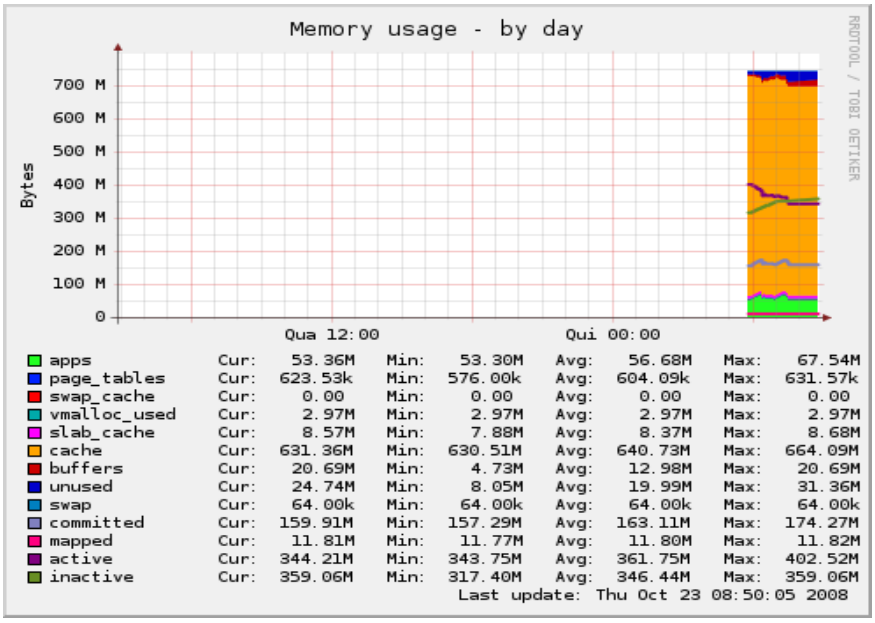


Figura 6.16 RAM do servidor 1, Teste 4, *storage*

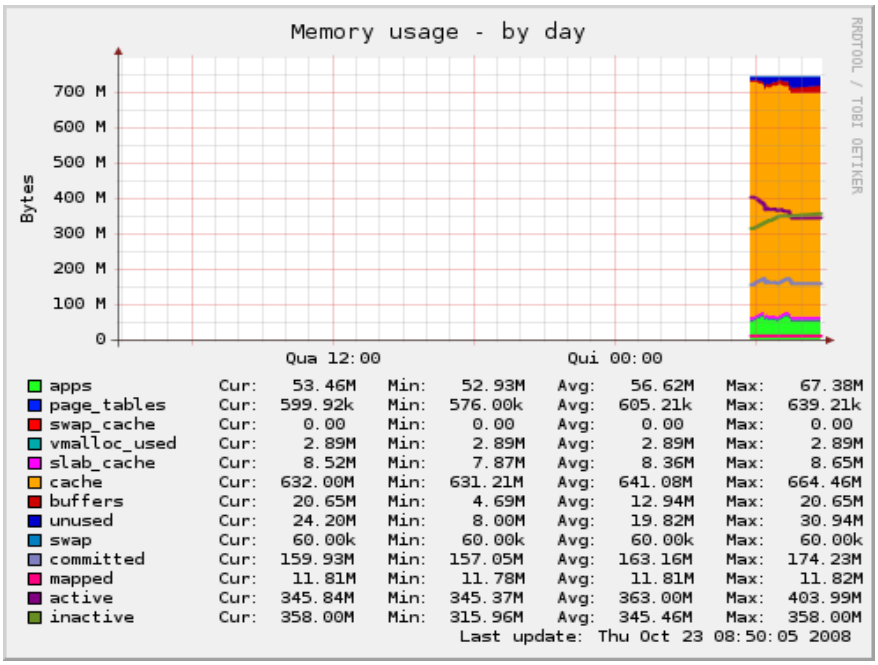


Figura 6.17 RAM do servidor 2, Teste 4, *storage*

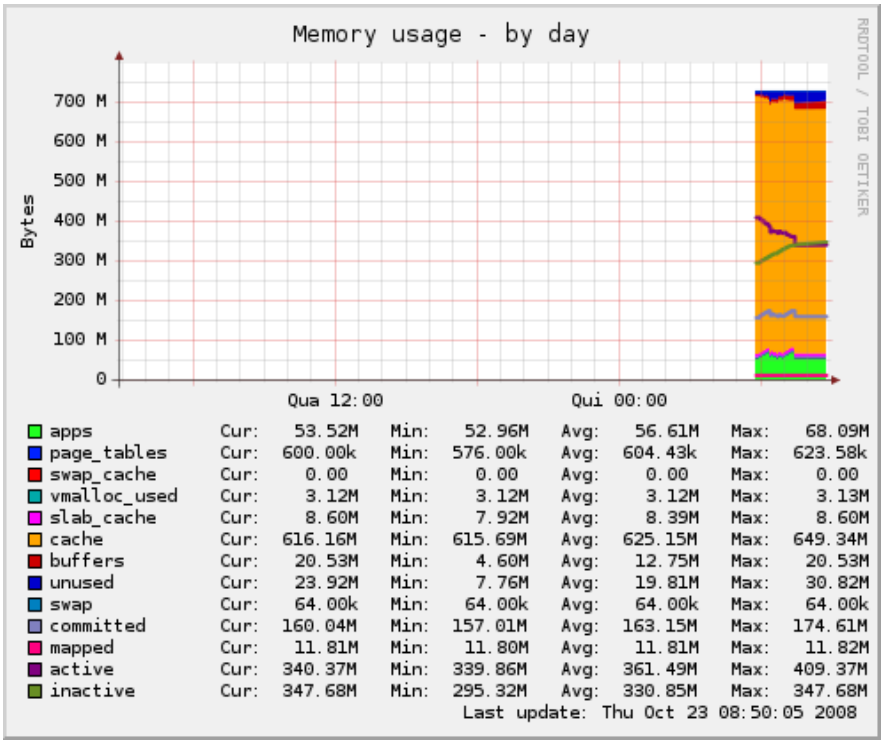


Figura 6.18 RAM do servidor 3, Teste 4, *storage*

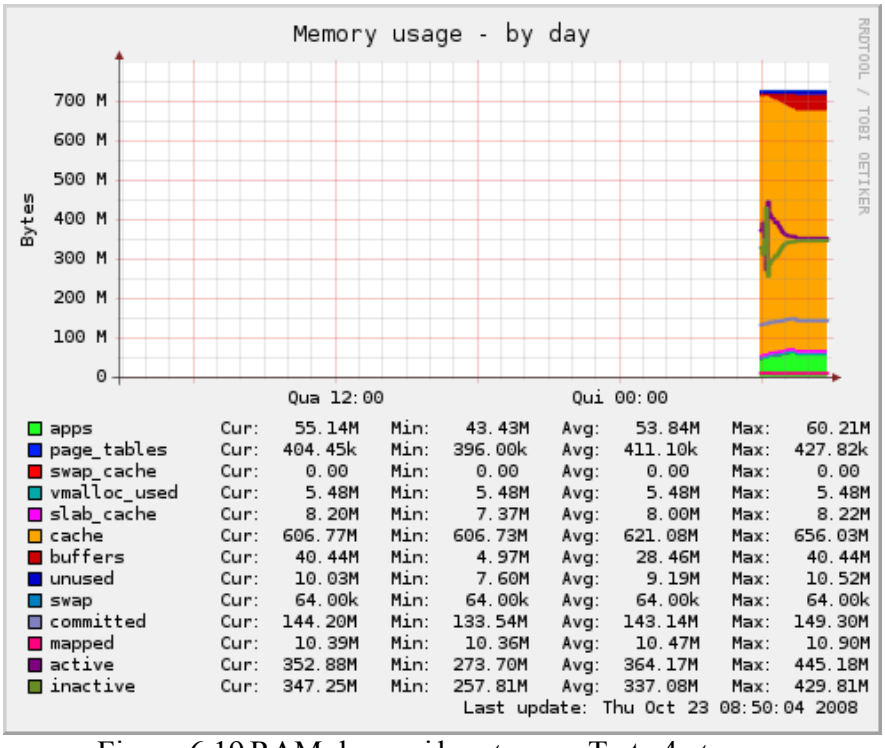


Figura 6.19 RAM do servidor *storage*, Teste 4 *storage*.

As Figuras 6.16, 6.17 e 6.18 se referem ao comportamento de consumo de memória principal dos nodos servidores reais. Pode-se observar que os recursos foram consumidos quase que na totalidade para todos os nodos. Na memória do servidor 1, da Figura 6.16, a quantidade de espaço ocioso foi de 19,99 Mbytes, em média. No servidor 2, da Figura 6.17, foi de 19,82 Mbytes. E foi verificado, na Figura 6.18, que a parte ociosa da memória do servidor real 3 foi de 19,81 Mbytes. Como o tamanho da memória de todas as máquinas envolvidas no teste é de 768 MBytes, pode-se concluir que em média 80,16% dos recursos da memória de cada nodo servidor real foram utilizados para auxiliar no processamento da demanda de trabalho evidenciado pela execução repetitiva da funcionalidade de geração de relatório de desconto da aplicação ADX.

A Figura 6.19 apresenta o consumo dos recursos disponíveis do nodo *storage* durante as duas horas de teste. Pode-se observar que a diferença é mínima, os recursos são utilizados, também, quase que na totalidade. A parte ociosa é de 9,19 MBytes, o que representa 90,81% dos recursos disponíveis utilizados para atender a demanda de operações MySQL oriundas dos servidores reais e pertinentes a funcionalidade de geração de relatório de descontos da aplicação ADX.

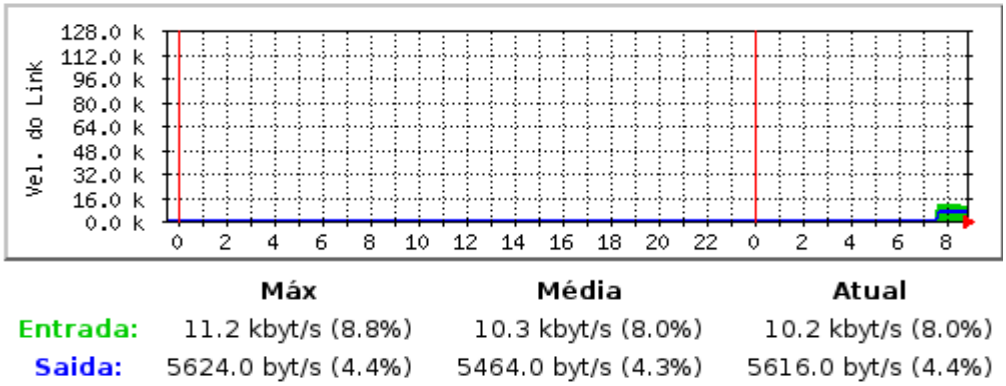


Figura 6.20 Rede do servidor 1, Teste 4, *storage*

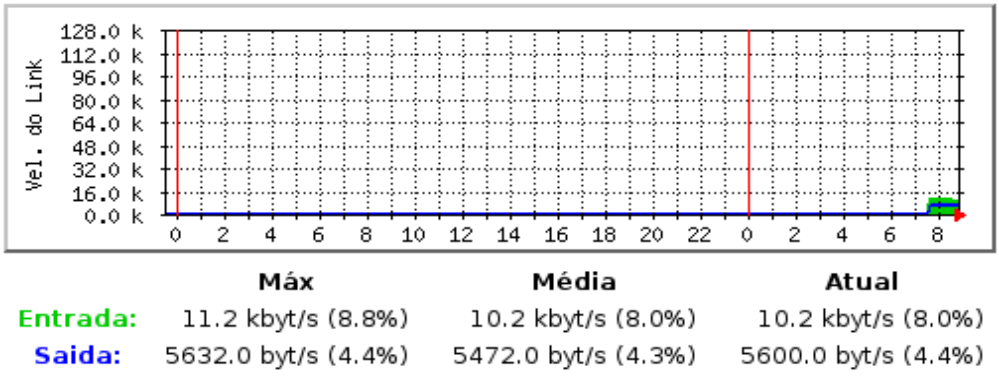
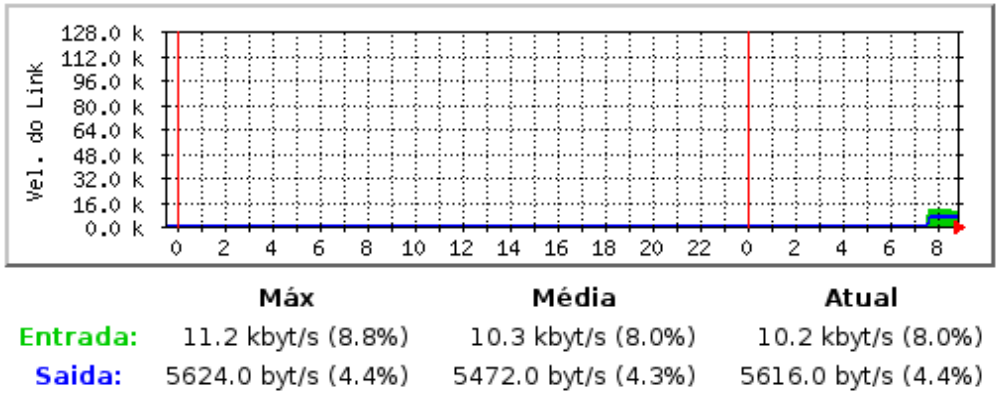
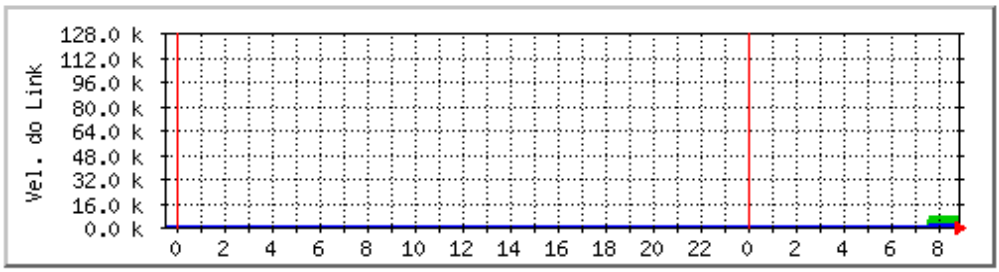


Figura 6.21 Rede do servidor 2, Teste 4, *storage*



	Máx	Média	Atual
Entrada:	11.2 kbyt/s (8.8%)	10.3 kbyt/s (8.0%)	10.2 kbyt/s (8.0%)
Saida:	5624.0 byt/s (4.4%)	5472.0 byt/s (4.3%)	5616.0 byt/s (4.4%)

Figura 6.22 Rede do servidor 3, Teste 4, *storage*



	Máx	Média	Atual
Entrada:	7648.0 byt/s (6.0%)	6800.0 byt/s (5.3%)	6680.0 byt/s (5.2%)
Saida:	576.0 byt/s (0.5%)	544.0 byt/s (0.4%)	576.0 byt/s (0.5%)

Figura 6.23 Rede do nodor storage, Teste 4, *storage*

As Figuras 6.20, 6.21 e 6.22 sintetizam as informações de tráfego de rede dos nodos servidores reais 1, 2 e 3 respectivamente. O tráfego teve um comportamento estável e permaneceu idêntico em todos os nodos servidores reais. O consumo de vazão de rede mostra o comportamento de 10,3 Kbytes/s para entrada e 5464,0 Bytes/s para saída de dados durante as duas horas de execução. Comparando o resultado do ambiente de *cluster* com nodo *storage* com o ambiente centralizado, pode-se perceber que o tráfego no *cluster* com *storage* foi menor para cada nodo comparado a máquina centralizada, mostrando assim, que esta última configuração contribuiu para o melhor desempenho dos tempos de respostas.

A Figura 6.23 apresenta o consumo de rede da interface do servidor *storage*. O tráfego foi 6800.0 Bytes/s para entrada e 544,0 Bytes/s para saída de dados durante as duas horas de execução. Pode-se perceber que o tráfego diminui quando comparamos com o comportamento dos nodos servidores reais. Isto acontece porque na interface do nodo servidor *storage* trafega apenas as solicitações de operações MySQL. O tráfego de

saída é ainda menor, pois o resultado da execução da funcionalidade de geração de relatório de desconto da aplicação ADX cria um arquivo no disco local onde a aplicação se encontra implantada, contendo a relação dos alunos que possuem desconto e qual é esse percentual. Ou seja, o tráfego de saída é mínimo.

O consumo de vazão de rede, apresentado na Figura 6.10, mostra o comportamento do ambiente centralizado em média de 16,4 Kbytes/s para entrada e 0,908 Kbytes/s para saída de dados. Já o somatório do tráfego de dados do ambiente de *cluster* com nodo *storage* apresenta em média 31,48 Kbytes/s para entrada e 2,184 Kbytes/s para saída de dados. Pode-se concluir que, quando se comparam esses dois ambientes, o tráfego, em média, é menor no ambiente centralizado, tanto para entrada quanto para saída de dados. Entretanto, quando se compara o tráfego dos dados do ambiente centralizado com o tráfego dos servidores reais e *storage*, individualmente, o ambiente de *cluster* tem uma menor utilização da largura de banda por interface de dados, otimizando, assim, o uso das interfaces para futuras alocações da largura de banda.

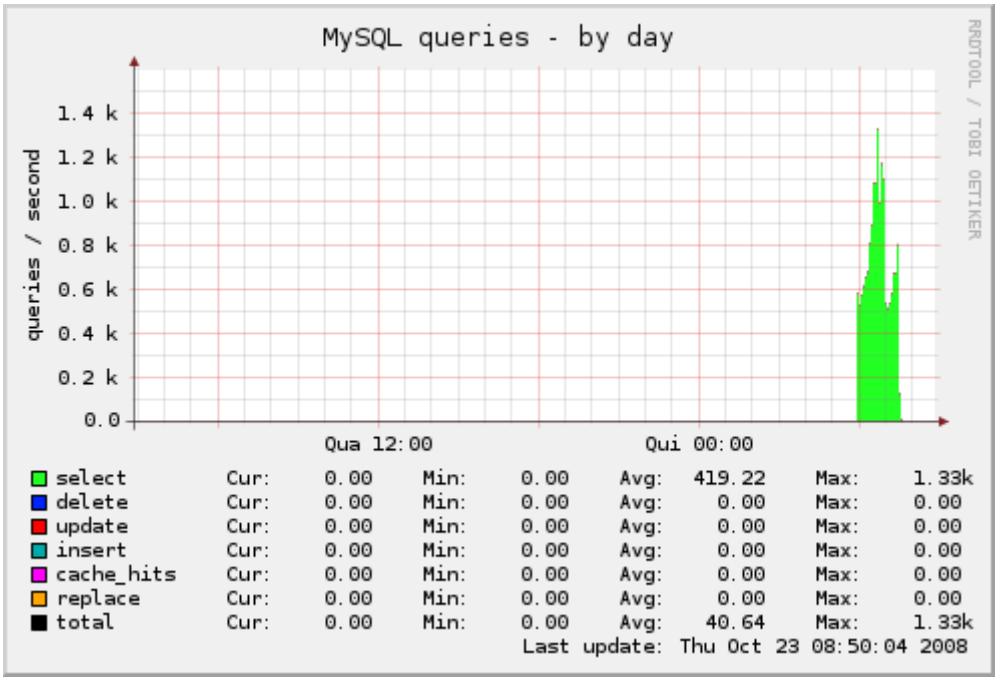


Figura 6.24 Consultas do servidor *storage*, Teste 4

A Figura 6.24 apresenta o comportamento de operações MySQL no servidor *storage*. A taxa média foi de 419,22 consultas (*selects*) para atender a demanda de execução dos testes. Comparando este valor com o valor de 632,42 consultas por segundo, da Figura 5.23, percebe-se que houve uma redução do número médio de consultas da ordem de 33%. Esta redução contribuiu para uma diminuição de

processamento e consumo de recursos do processador, o que evidenciou melhor desempenho da execução da bateria de testes 4 para o ambiente de *cluster* configurado com nodo *storage,* quando comparado ao ambiente centralizado.

c-) *Cluster* com armazenamento replicado e atualização periódica

Uma outra possível topologia para configuração do ambiente de *cluster* de alto desempenho e alta disponibilidade é arranjar a estrutura de tal forma que exista uma instância do banco de dados da aplicação ADX em cada nodo servidor real. Ou seja, eliminar o nodo *storage*, para que o processamento das operações MySQL aconteça nos próprios servidores reais.

Este fato evidencia melhor otimização do armazenamento do banco de dados da aplicação, pois este não estará mais posicionado em apenas um nodo. Dessa forma, não existirá mais um ponto único de falha para o banco de dados. Na verdade o banco estará distribuído e replicado entre os nodos servidores reais. Um outro benefício esperado é a diminuição do tráfego de dados no ambiente computacional. Esta topologia é apresentada na Figura 5.3.

Para geração de registro de informações sobre o resultado das execuções da funcionalidade, também foi criado um *script* de arquivo texto para armazenar o resultado para cada repetição da execução da rotina do ADX avaliada no ambiente com banco de dados replicado no nodos servidores reais. Seis execuções da funcionalidade foram iniciadas, simultaneamente, através do comando lynx 192.168.0.254. O IP 192.168.0.254, é o endereço virtual do ambiente de *cluster*. O funcionamento consiste na chegada das requisições ao nodo balanceador de carga e este distribuir essas requisições para os nodos servidores reais. A formatação destes dados podem ser visualizadas na Tabela 5 a seguir. O Anexo IV apresenta os arquivos *scripts,* separados por testes, gerados no momento das repetições da execução da funcionalidade de geração de relatórios de descontos do ADX.

Tabela 4: Tempos de execução - Teste 4 (replicado)

Testes	Tempo em horas
1	0,637
2	0,635
3	0,647
4	0,637
5	0,636
6	0,636
Média	**0,638**

Comparando os dados de tempo de execução dos testes da Tabela 3 (ambiente centralizado), com os dados da Tabela 4 (ambiente de *cluster* com nodo *storage*) e com os dados da Tabela 5 (ambiente de *cluster* com banco de dados do ADX replicado nos nodos servidores reais), pode-se perceber que a média dos tempos de execução no ambiente com banco de dados replicado é menor. Ou seja, enquanto o ambiente centralizado gastou 1,943 horas para a repetição de seis vezes da execução da funcionalidade de geração de relatório de desconto e o ambiente de *cluster* com nodo *storage* gastou 1,626 horas em média, o ambiente de *cluster* com banco de dados replicado gastou apenas 0,638 horas ou aproximadamente 38,173 minutos.

Pode-se afirmar que a execução da bateria de teste 4 no ambiente de *cluster* com banco de dados replicado tem melhor desempenho, da ordem de 67,14%, que o ambiente centralizado. Comparando com o ambiente de *cluster* com nodo *storage*, a melhoria de desempenho é da ordem de 60,76% para a mesma demanda de trabalho executada da aplicação ADX.

Analisando o comportamento do ambiente, constata-se que este resultado acontece devido aos nodos servidores reais executarem todo processamento lógico da funcionalidade, inclusive os que se referem a operações de banco de dados. Esta configuração permite uma otimização ainda maior dos recursos computacionais envolvidos, quando comparada com a configuração do ambiente de *cluster* com nodo storage, pois há um balanceamento de carga também para as operações de banco de dados. No caso da funcionalidade de geração de relatório de descontos da aplicação ADX, as operações de consultas (*selects*) são dividas também pelos nodos servidores reais. Este fato provocou melhor desempenho no que se refere ao tempo de execução da funcionalidade, quando compara-se os três ambientes.

O acompanhamento do comportamento do balanceamento de carga constatou que

cada nodo servidor real executou exatamente duas requisições, como no ambiente de *cluster* com nodo *storage*. Entretanto os gráficos, que vem logo em seguida, de operação de banco de dados, demonstram a divisão da carga de consultas (selects) também entre os nodos servidores reais.

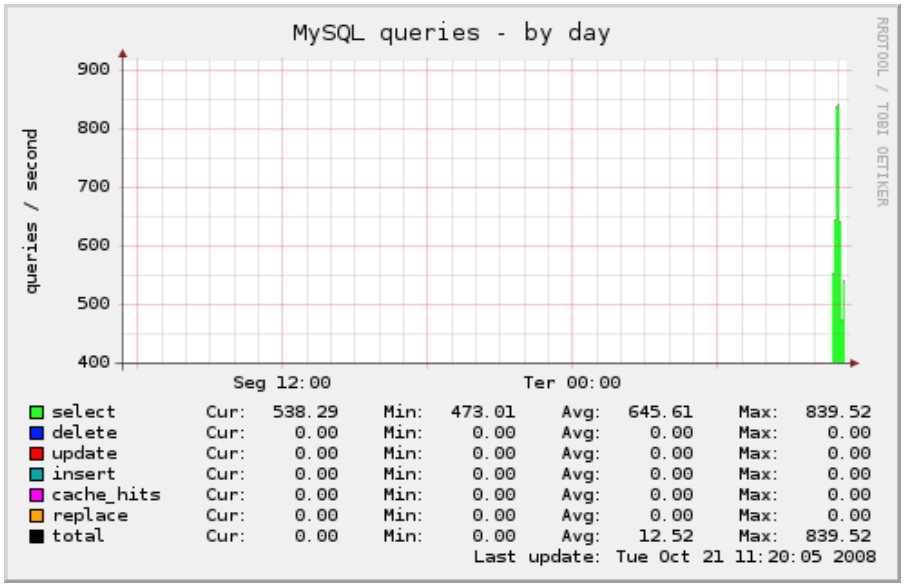


Figura 6.25 Consultas do servidor real 1, Teste 4

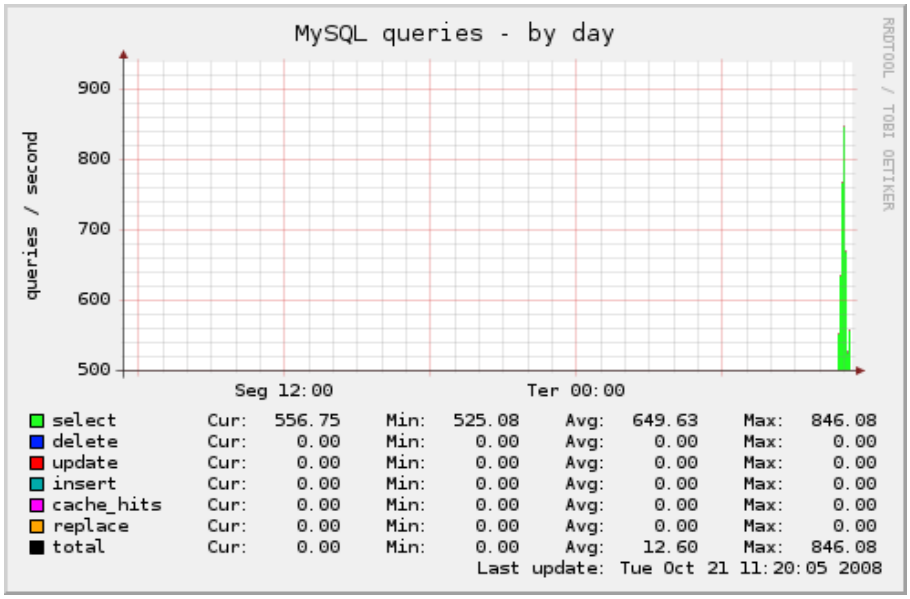


Figura 6.26 Consultas servidor real 2, Teste 4

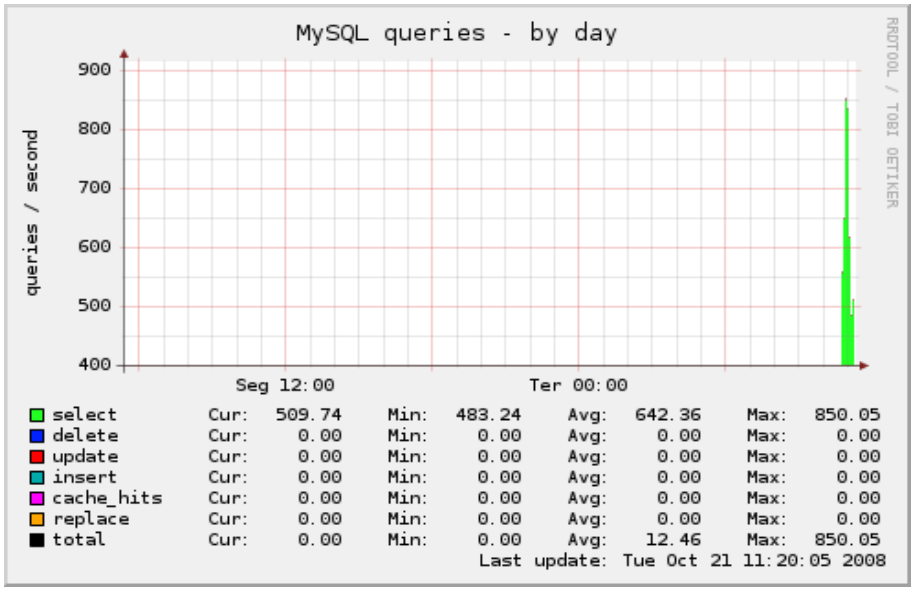


Figura 6.27 Consultas servidor real 3, Teste 4

Visualizando a Figura 6.11 (atividade de operações de consultas do ambiente centralizado), a Figura 6.24 (atividade de operações de consultas do ambiente com nodo *storage*) e comparando com as Figuras 6.25, 6.26 e 6.27 do ambiente de *cluster* com banco de dados replicado, que apresenta as atividades de operações de consultas dos nodos servidores reais 1, 2 e 3 respectivamente, pode-se concluir que é menor o nível de atividades de operações de consultas nos nodos servidores reais do ambiente de *cluster* com banco de dados replicado.

Este fato acontece devido à distribuição da carga ser dividida igualmente entre os nodos servidores reais e conseqüentemente, cada nodo servidor real executou apenas as operações de consultas provenientes de duas requisições de execução da funcionalidade de geração de relatório de descontos da aplicação ADX, diferente do ambiente centralizado e ambiente de *cluster* com nodo *storage* que executaram todas as operações de consultas provenientes das seis execuçõesda funcionalidade em uma mesma máquina.

Entretanto, o número médio de consultas não difere drasticamente quando comparamos os três ambientes. Os nodos servidores reais apresentaram média de 645,86 consultas por segundo para execução de duas vezes da funcionalidade em questão. Enquanto o ambiente centralizado teve média de 632,42 e o ambiente de *cluster* com nodo storage teve média de 419,22 para executar a funcionalidade seis vezes. Este fato se dá devido as características da lógica aplicada ao código da funcionalidade. O que importa neste teste é que ficou comprovado que através da distribuição da carga, tanto da lógica básica da aplicação quanto da lógica de operações com banco de dados, tem-se desempenho melhor de uma aplicação que exige alto nível de desempenho quando

compara-se com ambientes centralizados.

Observe a seguir o comportamento dos processadores, memórias e vazão de rede. Todos estes recursos foram utilizados de forma otimizada, o que resultou no menor tempo de resposta verificado na Tabela 5.

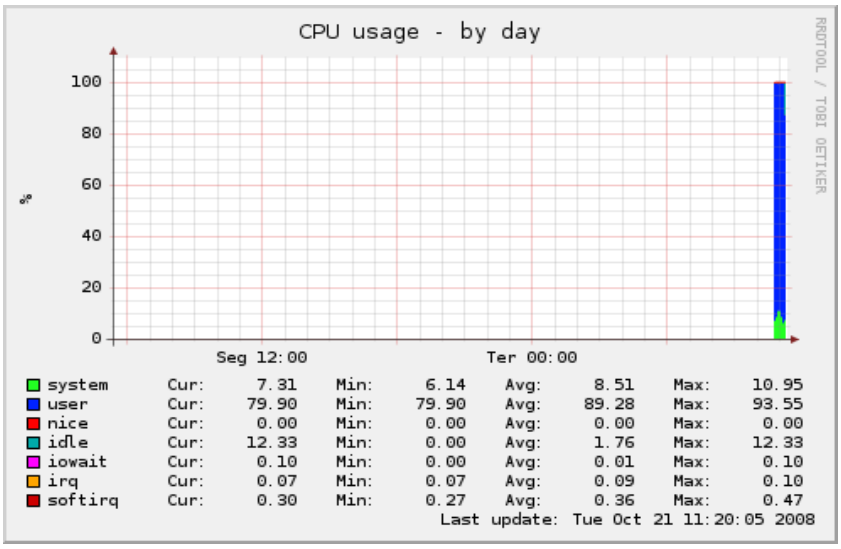


Figura 6.28 CPU do servidor real 1, Teste 4

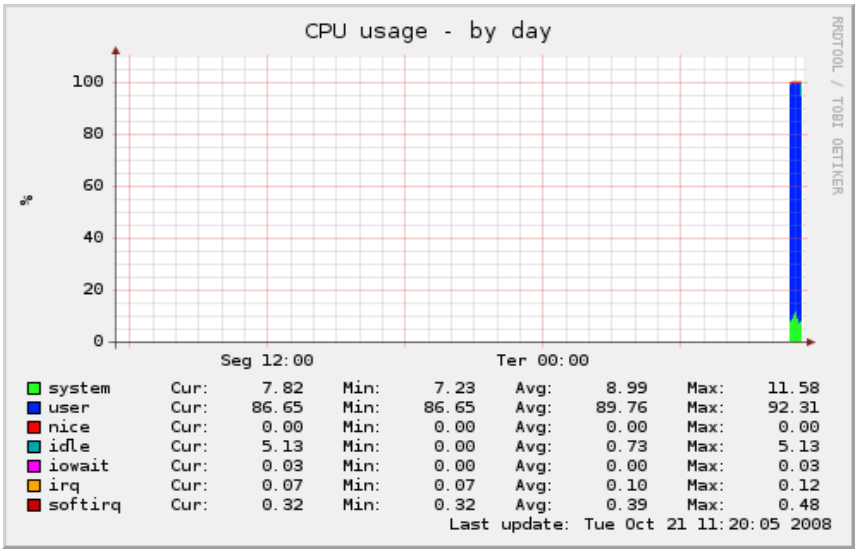


Figura 6.29 CPU do servidor real 2, Teste 4

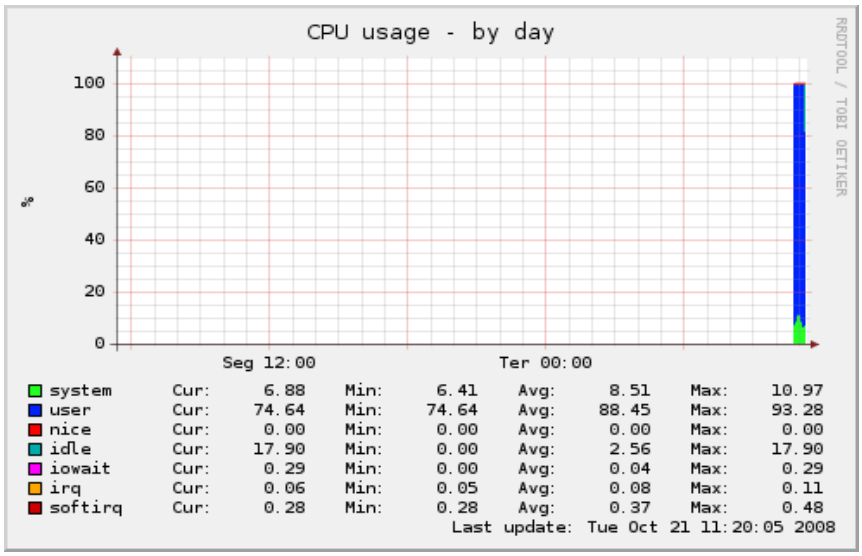


Figura 6.30 CPU do servidor real 3, Teste 4

A partir das Figuras 6.28, 6.29 e 6.30 dos nodos servidores reais do ambiente de *cluster* com banco de dados replicado, pode-se constatar que apesar da atividade de utilização dos recursos do processador ser intensa, em média 97,83% do recursos para os três servidores reais, a carga de trabalho é dividida entre os três servidores, o que resulta em menor tempo de ocupação dos recursos e conseqüentemente resultando em tempo de resposta menor ou melhor desempenho quando comparado com os outros ambientes.

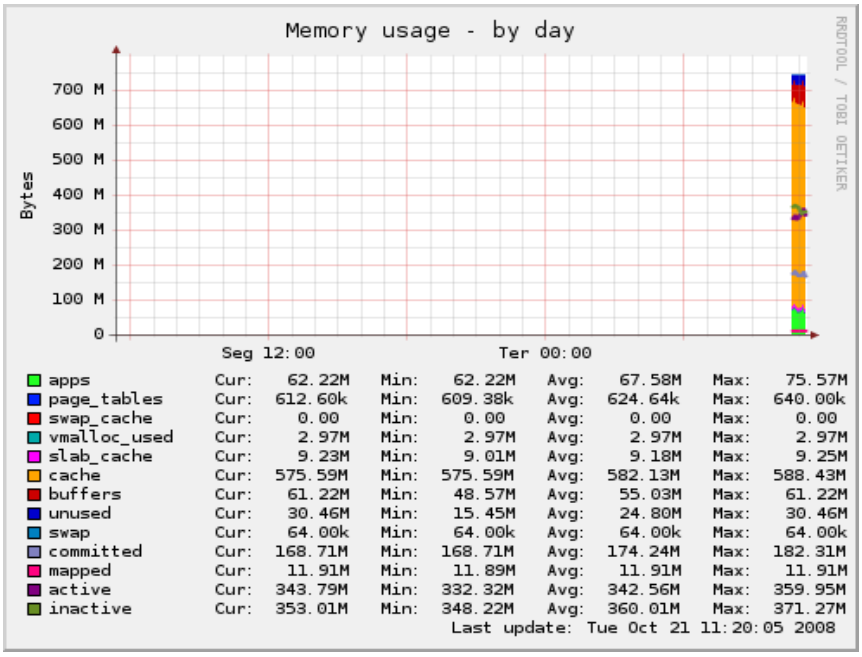


Figura 6.31 RAM do servidor real 1, Teste 4

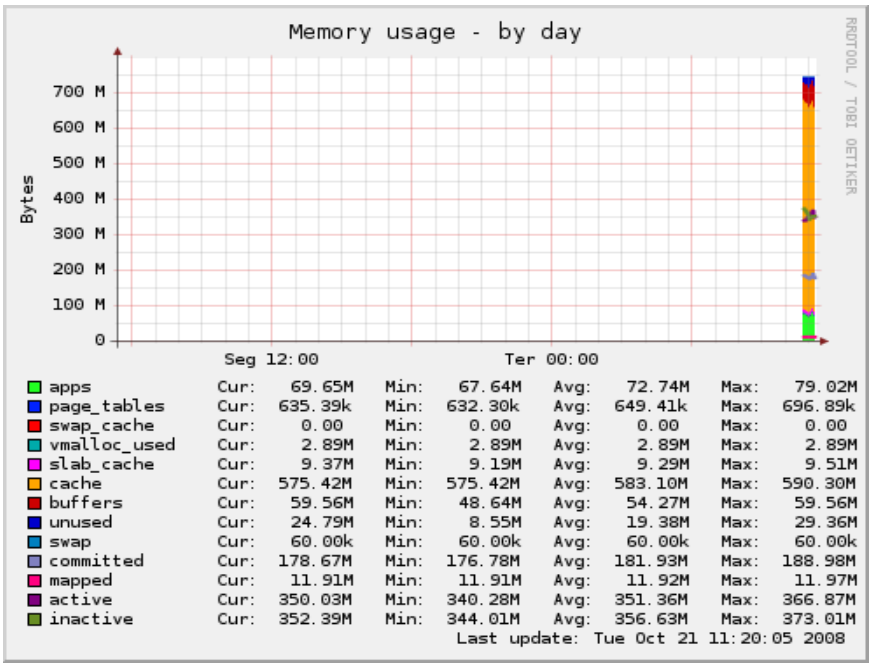


Figura 6.32 RAM do servidor real 2, Teste 4

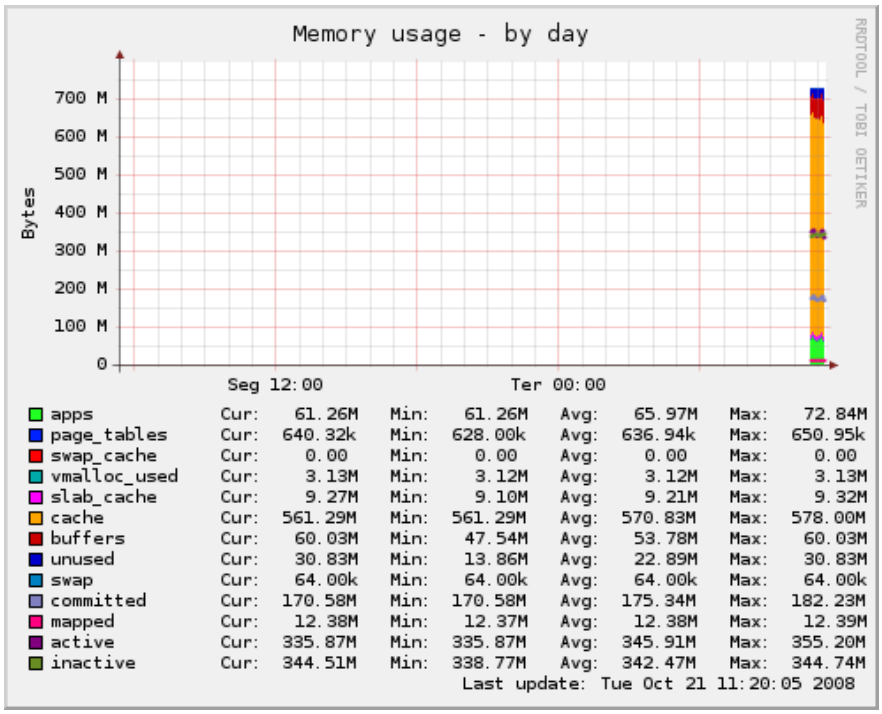


Figura 6.33 RAM do servidor real 3, Teste 4

As Figuras 6.31, 6.32 e 6.33 apresentam a utilização dos recursos de memória principal dos nodos servidores reais do ambiente de *cluster* com banco de dados replicado. O servidor real 1 usou em média, para a demanda da aplicação ADX, 67,58

Mbytes, o servidor real 2 usou 72,74 Mbytes e o servidor real 3 utilizou 65,97 Mybtes, resultando assim numa média de 68,76 Mbytes. Este valor é maior que o do ambiente de *cluster* com banco de dados compartilhado, que em média foi 55,93 Mbytes para todos os servidores, inclusive o de banco de dados. Este fato ocorreu devido a estrutura do *cluster* com banco de dados replicado permitir que os nodos servidores reais executem toda a lógica da aplicação ADX, incluindo as de operações com o banco de dados.

O gasto de memória para a demanda da aplicação ADX no ambiente centralizado foi de 84,74 Mbytes, um pouco maior que no ambiente de banco de dados replicado. Isso se justifica pois não existe nenhum tipo de redundância física de memória para compartilhar a alocação dos recursos demandada pela aplicação.

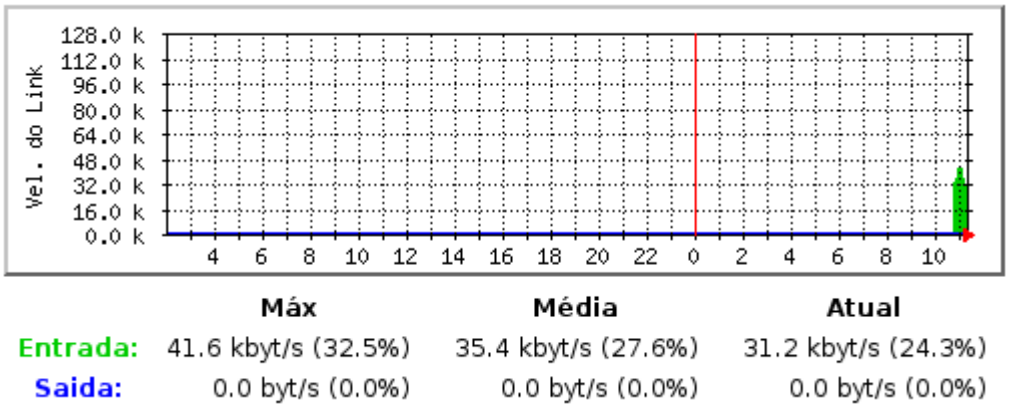


Figura 6.34 Rede do servidor real 1, Teste 4

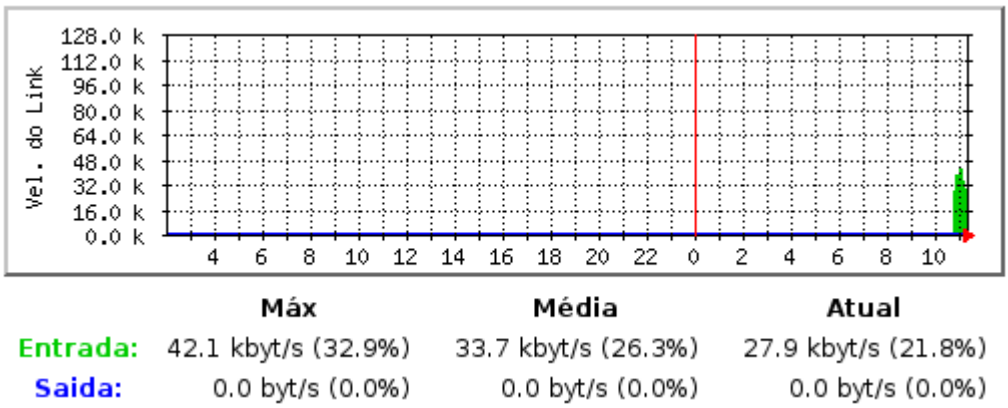


Figura 6.35 Rede do servidor real 2, Teste 4

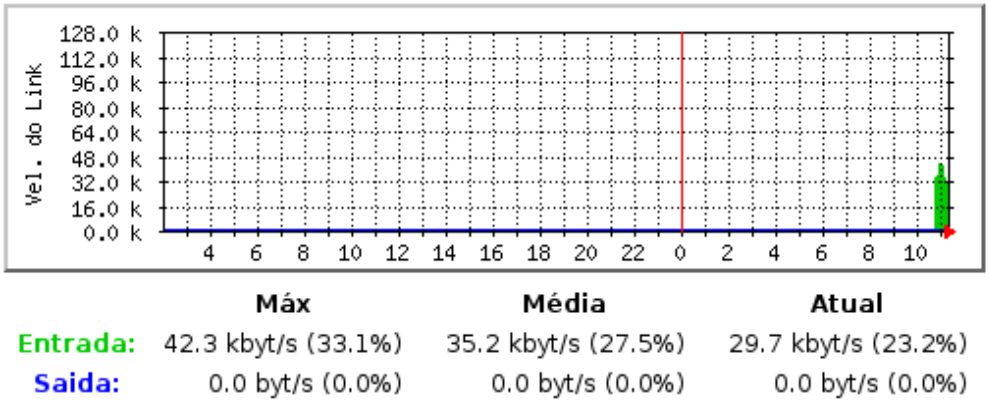


Figura 6.36 Rede do servidor real 3, Teste 4

Já as Figuras 6.34, 6.35 e 6.36 apresentam a utilização da largura de banda dos nodos servidores reais do ambiente de *cluster* com banco de dados replicado. A taxa de entrada de dados foi de 35,4 Kbytes/s e de 0 Kbytes/s para a saída de dados do nodo servidor real 1. Para o nodo servidor real 2 a taxa de entrada foi de 33,7 Kbytes/s e de 0 Kbytes/s para a saída de dados. O nodo servidor real 3 apresentou taxa de 35,2 Kbytes/s e de 0 Kbytes/s para a saída.

Pode-se observar que a utilização da largura de banda, através do tráfego de entrada e saída de dados, é menor e num período reduzido de tempo, quando compara-se as taxas relacionadas no monitoramento do ambiente centralizado e ambiente de *cluster* com nodo *storage*. Um fato interessante neste teste é que os três nodos servidores reais apresentaram uma taxa de 0 Kbytes/s para saída de dados pelas interfaces de rede. Isto aconteceu devido a característica da aplicação, que define que o resultado da operação da funcionalidade de geração de relatório de desconto deve ficar armazenado no disco onde se encontra a implantação da aplicação ADX, que neste caso está nos próprios servidores reais.

6.2.5 Teste 5 – Alta Disponibilidade para o Balanceador

A quinta bateria de teste deste estudo de caso consiste na demonstração da alta disponibilidade do ambiente de computação com *cluster* através da redundância do nodo servidor balanceador de carga apresentada na arquitetura do capítulo 4. Pretende-se comprovar que através de técnicas de checagens periódicas de estados de serviços e recursos por meio de temporizadores, assim como proposto na solução adotada, é possível obter um ambiente altamente disponível.

A principal definição para este teste foi a simulação da falha. Optamos por realizar um tipo de falha crítica, do tipo física, na qual não há possibilidades do sistema voltar imediatamente ou sem intervenção humana. A falha definida foi a da perda total de funcionamento da interface de rede do nodo servidor balanceador de carga principal. Foi simulado um erro físico, como a perda de energia, simplesmente desligando a interface de rede do nodo balanceador de carga principal no momento de atendimento das requisições enviadas a aplicação ADX. O objetivo desta ação é analisar o comportamento do sistema computacional mediante uma falha grave e desenvolver comentários.

Uma questão importante sobre a ferramenta de *benchmark* HTTPERF utilizada nesta bateria de testes se refere as respostas das requisições. O HTTPERF simula uma situação ideal de uma rede de comunicação de dados, onde o *timeout* para as respostas dos pedidos de requisições é sempre do tamanho máximo, de tal forma que todas as requisições obtenham suas respostas, também chamadas de *acks*.

Para este teste, utilizamos o HTTPERF com as configurações padrões, ou seja, com o tamanho de *timeout* ideal para esperar por todas as respostas das requisições. O resultado encontrado foi que, para todas as dez repetições do teste de mil requisições, nenhuma requisição era perdida. Esse mesmo teste foi repetido inúmeras vezes, e em apenas um teste houve uma perda, deixando de atender a apenas uma solicitação da aplicação.

O retorno da execução do HTTPERF demonstrou que as mil requisições enviadas a aplicação crítica foram respondidas, mesmo com a presença de uma falha grave. Entretanto o tempo de resposta para a conclusão de todas as requisições foi alto. Foram gastos em média 21,875 s para finalizar cada teste. Se compararmos este tempo com uma execução típica, percebemos que é extremamente alto, pois, um tempo médio de duração de teste para mil requisições foi encontrado na ordem de 0.801 s.

O tempo de duração de teste foi alto devido a ação de troca de servidores. Assim que o *heartbeat* detectou que havia uma falha no nodo servidor balanceador de carga principal, as ações para substituir o nodo principal, em estado de falha, pelo nodo *backup*, começaram. A consequência deste fato foi um tempo de duração de teste maior, pois a ferramenta HTTPERF foi obrigada a esperar pelas respostas das mil requisições enviadas a aplicação ADX.

O nodo servidor balanceador de carga principal conseguiu distribuir para os nodos servidores reais, em média, 384 requisições até o momento da falha. A partir deste momento inicia-se o processo de transição do comando do balanceamento da carga para o nodo *backup*.

O nodo balanceador de carga *backup* distribuiu, em média, as outras 616 requisições que o servidor principal não conseguiu atender devido a falha. Este fato, comprova que o ambiente atinge o objetivo de garantir maior disponibilidade para aplicação ADX.

Entretanto, em uma situação real de rede, para este tipo de situação, de receber uma rajada de mil requisições de uma só vez, erros podem acontecer, durante o procedimento de *failover* apresentado. Devido ao *timeout*, podemos ter um tempo de execução grande e perder algumas requisições devido a descartes de pacotes para desalojamento de recursos em consequência de ter excedido o *timeout.*

Devido a esta situação, foi desenvolvido o mesmo teste simulando o *timeout* máximo de uma rede local, que gira em torno de 0,5 s [BRADEN 1992]. Ou seja, com esta definição, o HTTPERF esperará as respostas para os pedidos enviados no máximo 0,5 s, simulando assim situações reais de rede.

O resultado do monitoramento da execução do HTTPERF apresenta que foi gasto um tempo de 34.136 s para atender as mil requisições destinadas a aplicação ADX. Destas, 932 tiveram respostas, ou seja, foram atendidas. Um total de 68 requisições foram perdidas devido à falha que aconteceu com o nodo servidor balanceador de carga principal.

Pode-se observar que o tempo de duração de teste foi maior que o da situação anterior que não configurava tempo de *timeout.* Este fato foi crucial para que acontecessem os descartes de pacotes das requisições que extrapolaram o tempo de espera.

O nodo servidor balanceador de carga principal conseguiu distribuir 155 requisições para os nodos servidores reais até o momento da falha. A partir deste momento inicia-se o processo de transição do comando do balanceamento da carga para o nodo *backup.*

O ambiente de *cluster* modelado a partir da arquitetura conseguiu atender a 778 requisições destinadas a aplicação ADX através do nodo balanceador de carga *backup.* Esta bateria de teste foi repetida dez vezes. E todos os testes apresentaram o mesmo comportamento. A seguir é apresentada a Tabela 6 que exibe os dados de tempo de duração de teste, o número de requisições efetuadas, o número de requisições atendidas e um cálculo de média e percentuais para análises do comportamento de disponibilidade da arquitetura.

Tabela 5: Testes de disponibilidade para o balanceador

Testes	Respostas	% perda	Tempo de Exec. (s)
1	932	6,8	34,136
2	935	6,4	32,669
3	933	6,7	33,413
4	932	6,8	34,789
5	935	6,5	32,299
6	933	6,7	34,299
7	933	6,7	34,306
8	936	6,4	32,821
9	934	6,6	33,814
10	934	6,6	33,809
Médias	**933,7**	**6,63**	**33,736**

A Tabela 6 apresenta os principais dados das dez repetições do teste de disponibilidade para o esquema de balanceamento de carga. Pode-se observar que a média de perda dos teste foi de 6,63%. Ou seja, mesmo diante de uma falha, o ambiente computacional conseguiu atender em média 93,37% das requisições enviadas a aplicação ADX configurada no *cluster* de alto desempenho e alta disponibilidade.

Outros testes, com a mesma finalidade, foram desenvolvidos com número de requisições maiores. Entretanto, o que conseguimos observar é que o percentual de perda continuava próximo a 6,63%.

A conclusão principal que pode-se retirar da bateria de testes 5 é que mesmo mediante uma falha grave do nodo servidor balanceador de carga é possível dar continuidade no atendimento a demanda de requisições à aplicação crítica, atendendo assim ao objetivo de alta disponibilidade definido neste trabalho.

6.2.6 Teste 6 – Alta Disponibilidade para os servidores reais

O desenvolvimento deste último teste também teve como objetivo demonstrar a alta disponibilidade da aplicação ADX. Entretanto, o que é avaliado neste teste é o comportamento do ambiente computacional quando acontece uma falha com um dos nodos servidores reais. Este teste consiste em comprovar que a arquitetura de *cluster* reuni as técnicas de desempenho e disponibilidade para construção de um ambiente

computacional robusto, aproveitando ao máximo a redundância dos equipamentos envolvidos.

A estrutura original da metodologia adotada para construção do *cluster* teve a definição de três nodos servidores reais para a realização do processamento da demanda de requisições destinadas a aplicação ADX. O interesse foi comprovar que mesmo com a perda de um dos três nodos servidores reais o ambiente computacional não deixaria de funcionar e ainda tentaria atender a demanda de requisições destinadas a aplicação crítica.

Uma outra definição para a concretização do teste em questão foi determinar que no momento que o ambiente computacional recebe a rajada de mil requisições, o nodo servidor real 1 do *cluster* teria a perda de funcionamento da interface de rede, simulando, assim um erro físico. Este teste também simulou a situação real de rede com *timeout* definido em 0,5 s para a ferramenta HTTPERF.

O resultado da execução do HTTPERF apresentou um gasto de tempo de 15.840 s para atender as mil requisições destinadas a aplicação ADX. Destas, 997 tiveram respostas, ou seja foram atendidas, sendo apenas 3 requisições perdidas devido a falha que aconteceu com o nodo servidor real 1.

Pode-se observar que o tempo de duração do teste foi maior que o de situações anteriores, onde não havia falhas no ambiente, que em média foram 0.801 s. Entretanto, quando comparamos este tempo de duração com o teste 5, de 34.136 s, onde a falha está no nodo servidor balanceador de carga principal e não no nodo servidor real 1, percebe-se que o valor é praticamente a metade, pois na falha do nodo servidor balanceador de carga principal há um tempo maior de processamento para transferir o comando da distribuição da carga para o nodo servidor balanceador de carga backup.

Através do comportamento do *cluster*, foi possível perceber que o nodo servidor real 1 atendeu 176 requisições antes da sua parada. Os outros dois nodos servidores reais 1 e 2 atenderam 413 e 411 respectivamente, totalizando assim as mil requisições destinadas a aplicação ADX. De acordo com o resultado do HTTPERF sabemos que três dessas requisições não foram atendidas.

A conclusão que pode-se ter deste teste se refere principalmente a constatação da robustez do ambiente. Ou seja, mesmo diante da perda de um nodo servidor real que auxilia no processamento da carga de trabalho, a aplicação não para de funcionar. Houve uma perda, devido ao descarte de três requisições, entretanto, a aplicação não parou de funcionar e o sistema ainda atendeu 997 requisições.

Este teste foi repetido dez vezes. A seguir é apresentada a Tabela 7 que mostra os

dados de tempo de duração de teste, o número de requisições efetuadas, o número de requisições atendidas e um cálculo de média e percentuais para análises do comportamento de disponibilidade da arquitetura.

Tabela 6: Testes de disponibilidade para os nodos reais

Testes	Respostas	% perda	Tempo de Exec. (s)
1	977	2,3	15,840
2	977	2,3	15,822
3	977	2,3	15,823
4	977	2,3	15,826
5	977	2,3	15,813
6	977	2,3	15,830
7	977	2,3	15,808
8	977	2,3	15,821
9	977	2,3	15,852
10	977	2,3	15,818
Médias	**977**	**2,3**	**15,825**

A Tabela 6 apresenta os principais dados das dez repetições do teste de disponibilidade do ambiente computacional com foco nos nodos servidores reais. Pode-se observar que a média de perda dos teste foi de 2,3 %. Ou seja, mesmo diante de uma falha, neste caso a perda de um dos nodos servidores reais, o ambiente computacional conseguiu atender em média 97,7% das requisições enviadas a aplicação ADX configurada no *cluster* de alto desempenho e alta disponibilidade.

Um outro aspecto importante deste teste é a perda de desempenho devido a perda do nodo servidor real 1. Na análise do registro de comportamento do servidor balanceador de carga, percebe-se que os nodos servidores reais 2 e 3 executaram mais requisições devido a falha no servidor real 1. Para comprovar a perda de desempenho, o teste foi executado novamente com o *cluster* configurado apenas com dois nodos servidores reais. As definições foram as mesmas: rajada de mil requisições por minuto, por duas horas, totalizando 120.000 requisições. O objetivo foi analisar o comportamento do processador, memória principal e vazão de rede e comparar com o *cluster* original com três nodos.

Tanto o processador quanto a memória principal tiveram um comportamento semelhante ao ambiente de *cluster* com três nodos servidores reais. Ou seja, devido a características da aplicação o consumo foi mínimo e se manteve praticamente idêntico. O

nodo servidor real 2 apresentou um consumo de 0,65% dos recursos do processador e o nodo servidor real 3 foi de 0,67% como mostram as Figuras 6.37 e 6.38 respectivamente. Comparando estes resultados com o resultado do *cluster* com três nodos servidores reais, pode-se constatar um leve aumento, pois, como apresentado na Figura 6.4, os processadores alcançaram média de 0,56% de consumo dos recursos do computador

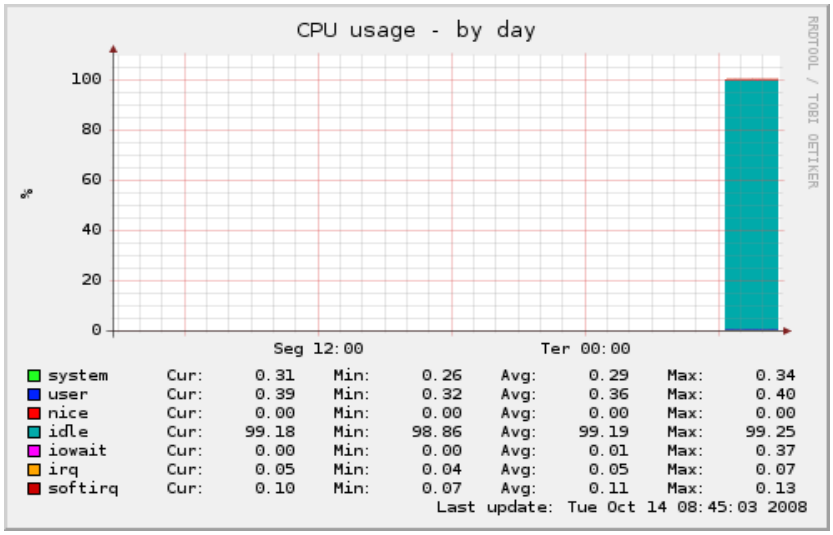


Figura 6.37 CPU do servidor real 2, Teste 6

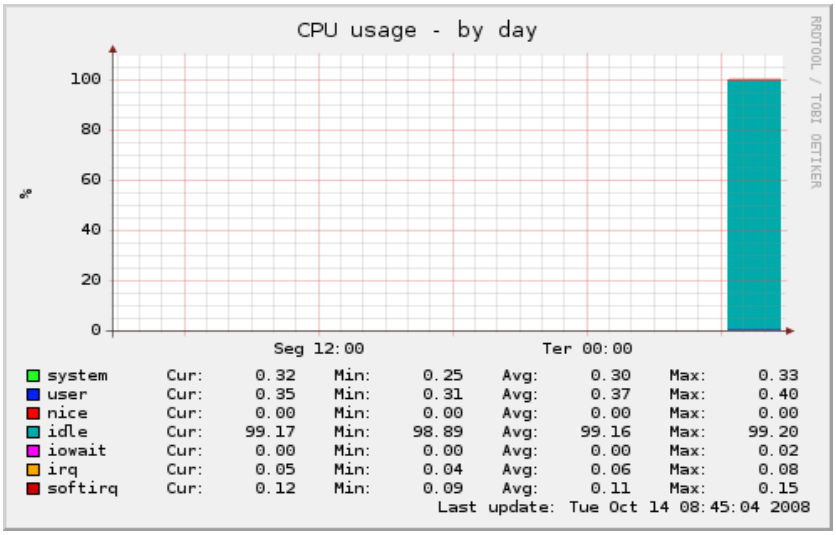


Figura 6.38 CPU do servidor real 3, Teste 6

Quando compara-se o consumo de memória principal dos dois ambientes, *cluster* com dois nodos servidores reais e *cluster* com três nodos, constata-se que no ambiente com dois nodos o consumo dos recursos foi maior. Este fato se deu porque os dois nodos precisaram absorver as requisições enviadas para o nodo servidor real 1 com falha, o que evidenciou o crescimento do número de requisições em cada nodo e conseqüentemente maior alocamento de recursos de memória principal .

A Figura 6.39 apresenta o consumo de memória para o nodo servidor real 2 e a Figura 6.40 o consumo do nodo servidor real 3.

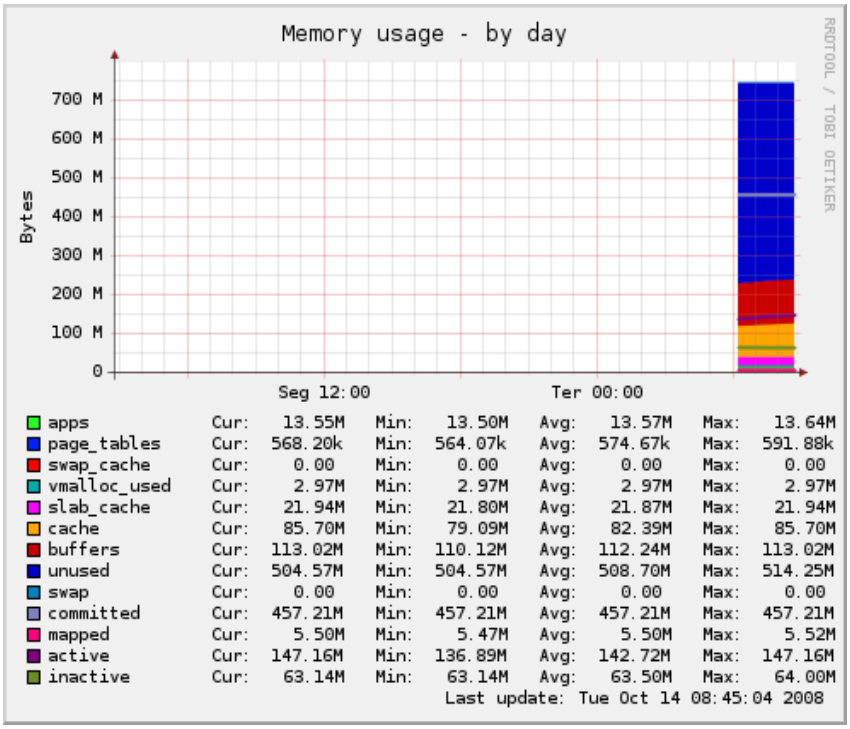


Figura 6.39 RAM do servidor real 2, Teste 6

Elas mostram que as memórias dos nodos servidores reais 1 e 2 ficaram ociosas em 508,7 MBytes. Isto significa dizer que foram consumidos 33,76% dos recursos disponíveis da memória principal pelos dois nodos servidores reais.

O percentual de utilização de consumo de memória no *cluster* com três nodos servidores reais não excedeu a 21,14% como pode ser observado na Figura 6.5. Estes dados comprovam que o consumo da memória principal foi maior no teste com apenas dois nodos servidores reais. Além disso, pode-se perceber que o tempo ocupado dos recursos de memória primária é maior no *cluster* com dois nodos servidores reais.

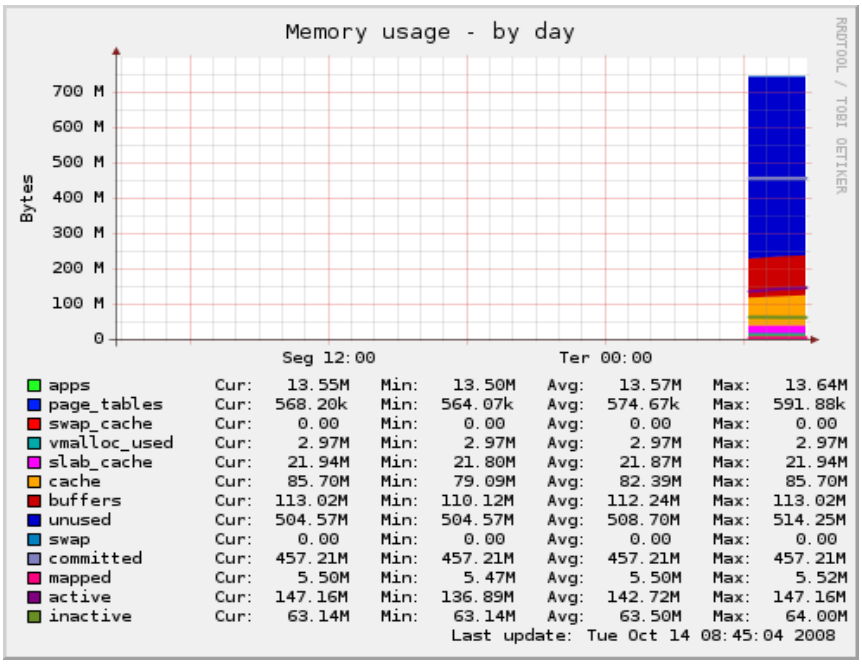
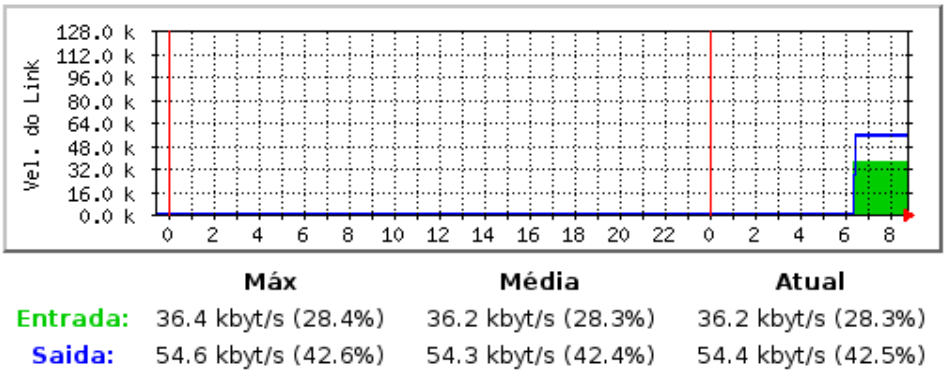


Figura 6.40 RAM do servidor real 3, Teste 6

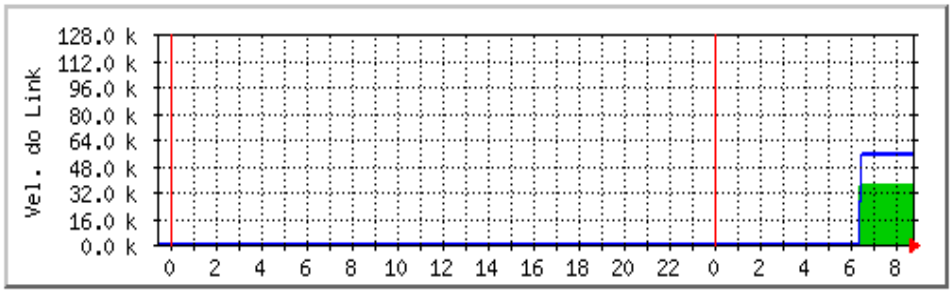
Quanto à vazão de rede, fica claro que o número de dados trafegados pelas interfaces de redes dos dois nodos servidores reais aumentou, quando comparamos com o *cluster* com três nodos servidores reais. A Figura 6.41 apresenta que o nodo servidor real 2 obteve taxas de 36,2 Kbytes de dados de entrada e 54,3 Kbytes de dados de saída trafegados por sua interface. A Figura 6.42 mostra que o nodo servidor real 3 obteve taxas de 36,1 Kbytes para entrada e 54,2 Kbytes para a saída.

No *cluster* com três nodos servidores reais pode-se verificar que o máximo de dados trafegado pelas interfaces de rede foram 27,5 Kbytes para entrada e 40,0 Kbytes para saída, como pode ser visto na Figura 6.6. A comparação destes dados comprova o aumento do tráfego de dados no *cluster* com dois nodos servidores reais, o que evidencia que a perda do nodo servidor real 1 resultou em perda de desempenho dos três recursos analisados no que se refere a vazão de rede.



	Máx	Média	Atual
Entrada:	36.4 kbyt/s (28.4%)	36.2 kbyt/s (28.3%)	36.2 kbyt/s (28.3%)
Saida:	54.6 kbyt/s (42.6%)	54.3 kbyt/s (42.4%)	54.4 kbyt/s (42.5%)

Figura 6.41 RAM do servidor real 2, Teste 6



	Máx	Média	Atual
Entrada:	36.3 kbyt/s (28.4%)	36.1 kbyt/s (28.2%)	36.2 kbyt/s (28.3%)
Saida:	54.6 kbyt/s (42.6%)	54.2 kbyt/s (42.4%)	54.4 kbyt/s (42.5%)

Figura 6.42 RAM do servidor real 3, Teste 6

Pode-se concluir que diante da falha de um dos nodos servidores reais, a aplicação ADX continuou oferecendo respostas às requisições, porém com a perda de desempenho devido a ausência de um dos nodos de processamento da carga enviada a aplicação ADX. Entretanto para este caso, a diferença não foi grande. Enquanto o tempo de duração de um teste de mil requisições no *cluster* com três nodos foi em média 0,77 s, o tempo de mil requisições no *cluster* com dois nodos gastava em média 0,80 s.

7. CONSIDERAÇÕES FINAIS

De acordo com a contextualização teórica deste trabalho pode-se concluir que sistemas computacionais que executam aplicações críticas necessitam da combinação das capacidades de disponibilidade, escalabilidade, desempenho e gerenciabilidade.

Um dos objetivos deste trabalho foi inserir todas estas capacidades na arquitetura de *cluster* de alto desempenho e alta disponibilidade apresentada, no intuito de comprovar que a realização de planos estratégicos para melhoria de desempenho de aplicações que rodam em circunstâncias críticas é possível, viável e necessário.

Profissionais da computação devem encarar seriamente os problemas ocasionados por falhas não tratadas nos sistemas informatizados. Tolerância a falhas compreende muitas das técnicas que permitem aumentar a qualidade de sistemas de computação. Apesar da tolerância a falhas não garantir comportamento correto na presença de todo e qualquer tipo de erro e apesar das técnicas empregadas envolverem algum grau de redundância e, portanto, tornarem os sistemas maiores e mais caros, ela permite alcançar a confiabilidade e a disponibilidade desejadas para os sistemas computadorizados.

7.1 Conclusões e Contribuições

Através dos testes realizados no estudo de caso apresentado foi possível constatar que a estrutura utilizada de *clusters* de alta disponibilidade e alto desempenho possibilitou melhor desempenho, no que se refere a otimização dos recursos computacionais envolvidos, escalabilidade e alta disponibilidade. A solução apresentada melhorou o tempo de resposta das tarefas da aplicação crítica em relação aos modelos tradicionais centralizados. Além disso possibilitaram também:

- analisar os parâmetros gerais da aplicação crítica, afim de compreender a influência que possuem no sistema global;
- visualizar e compreender o funcionamento do mecanismo de balanceamento de carga, que teve como objetivo distribuir de maneira eficiente e equilibrada as tarefas de processamento entre os processadores que o sistema computacional possuía;
- utilizar softwares de gerência de redes e sistemas computacionais para realizar medidas de desempenho, tais como:
    - avaliar o ganho no tempo de resposta comparado com a execução em sistemas computacionais críticos funcionando em modelos tradicionais;

- avaliar a disponibilidade do sistema de computação e sua tolerância a falhas mediante a problemas críticos, como a perda de um processador ou interface de rede.

A principal contribuição deste trabalho para a área de gerência de redes e serviços é a constatação de que é possível e totalmente viável o desenvolvimento de um ambiente profissional e robusto utilizando mecanismos de gerenciamento de serviços de sistemas computacionais críticos, através da filosofia de sistemas livres e de código fonte aberto, tais como,GNU/Linux, *Linux Virtual Server Project* e *Heartbeat*. A solução apresentada neste trabalho evidencia custo financeiro zero com sistemas de softwares para implantação de todo o ambiente. Ou seja, é uma alternativa eficiente e barata, para *clusters* de computação de alto desempenho e de alta disponibilidade.

7.2 Trabalhos Futuros

Em síntese, este trabalho apresentou uma forma de utilizar *clusters* de computação como infraestrutura para aplicações críticas no intuito de melhorar desempenho e disponibilidade de dados e demais recursos envolvidos no ambiente. O carater experimental possibilitou constatar na prática a eficiência da proposta. Dentro desta linha, visualiza-se outras ações e possibilidades para enriquecer o presente trabalho. As principais ações são:

- Alterar política de escalonamento do LVS: a política de escalonamento utilizado foi a Round-Robin, devido ao custo de implantação. Esta se limita a criar uma espera circular para as requisições que chegam ao nodo balanceador de carga, através do endereço virtual do *cluster*. Uma possibilidade de um resultado significativo é o teste de outras políticas de balanceamento de carga e a comparação dos resultados;
- Propor modificações em políticas de escalonamento do LVS: como LVS é um projeto livre, de código fonte aberto indica-se verificar o funcionamento das políticas de escalonamento e propor melhorias;
- Propor novas políticas de escalonamento para o LVS: o objetivo para esta ação ou possibilidade seria analisar o funcionamento e a implementação do LVS e propor novas políticas de escalonamento de tarefas. Entretanto, considera-se que o objetivo deva ser bem específico, pois existem várias políticas de escalonamento clássicas já implementadas para o LVS;

- Alterar aplicação crítica: como boa parte do desenvolvimento de novas aplicações estão sendo concentradas no ambiente da Internet, este trabalho analisou um sistema de informação que utiliza o protocolo HTTP para executar suas funcionalidades. Uma outra possibilidade de trabalho futuro seria utilizar a mesma estrutura apresentada, juntando HA e HP, para analisar outro tipo de aplicação, como por exemplo, sistemas legados executado em rede via *socket* de comunicação de dados;
- Utilizar hardware com diferentes desempenho: neste trabalho, todos os hardwares utilizados foram de mesma arquitetura de computadores e mesma configuração. Seria interessante colocar equipamentos de diferentes configurações para analisar o comportamento nas mesmas circunstâncias analisadas;
- Utilizar outros sistemas de *clusters* que se adaptem à estrutura proposta neste trabalho: o LVS foi utilizado neste trabalho devido a licença livre e a adaptação completa a estrutura proposta de HA e HP. Um experimento interessante para a estrutura proposta seria procurar por outro sistema de computação de *cluster*, livre ou proprietário, que se encaixa nas diretrizes da proposta. O objetivo seria desenvolver os mesmos teste realizados neste trabalho para comparar o desempenho dos sistemas de *clusters*.

REFERÊNCIAS BIBLIOGRÁFICAS

ALTEON, Networs; The Next Step in Server Load Balancing. San Jose, Califórina, 1999. Disponível em http://www.olivercom.com/pdf/solutions/slb.pdf. Último acesso em 28 fevereiro 2008.

ANDERSON, T.; LEE, P. A. Fault tolerance -principles and practice. Englewood Cliffs, Prentice-Hall, 1981.

AVIZIENIS, A.; Design of fault-tolerant computers, proceedings of Fall Joint Computer Conf., vol 31, Thompson Books, 1976.

AVIZIENIS, A.; LAPRIE, J.; RANDELL, C.; LANDWEHR, C. E. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Sec. Comput., 1(1):11–33. 2004.

BARAK, A.; Cluster and Multi-Cluster Grid Management. 2008. Disponível em http://www.mosix.org/. Último acesso em 02 de junho 2008.

BARROSO, L.; DEAN, J.; HOELZLE, U.; Web Search for a Planet: The Google Cluster Architecture. IEEE MICRO, 2003.

Beowulf.org; What makes a cluster a Beowulf. 2007. Disponível em http://www.beowulf.org/overview/index.html. Último acesso em 02 de junho 2008.

BRADEN, R.; Extending TCP for Transactions - Concepts. University of South California, Information Sciences Institute. Disponível em HTTP://www.rfc-archive.org/getrfc.PHP?rfc=1379. último acesso em 17/12/2008.

BUYYA, R.; High Performance Cluster Computing: Architectures and Systems. Volume 1, Prentice Hall, New Jersey, 1999.

COMPAQ Computer Corp.; INTEL Corp.; MICROSOFT Corp. Virtual Interface Architecture Specifica tion. Disponível em http://www.viarch.org, Version 1.0, 1997. Último acesso em 12 de outubro de 2008.

DANTAS, Mário; Computação Distribuída de Alto Desempenho. Axcel Books. 2005.

DOCTUM; instituto Doctum de Educação e Tecnologia. Disponível em http://www.doctum.com.br. 2008. Último acesso em 05/06/2008.

FUJIWARA, E. Code Design for Dependable Systems: Theory and Practical Applications. John Wiley & Sons. Inc., 2006.

HORMAN, S.; Ultra Monkey: Load Balacing and High Availability Solution. Disponível em http://www.ultramonkey.org. 2008. último acesso em 06 julho de 2008.

KOPPER, K.; The Linux Enterprise Cluster Build a Highly Available Cluster with Commodity Hardware and Free Software. No Starch Pr. 2005.

KRETCHMAR, J. M.; Open Source Network Administration. New Jersey: Prentice Hall, 2003.

KUROSE, J.; ROSS, W. K. Redes de Computadores e a Internet – uma abordagem top-down - 3 ª Edição. Pearson. 2005.

LAPRIE, J. C; Dependability: von concepts to limits. In: Proceedings of the IFIP International Workshop on Dependable Computing and its Applications. DCIA 98, Johannesburg, South Africa, January 12-14, 1998. p.108-126.

LINUX-HA project; The basic goal of the High Availability Linux project. Disponível em http://www.linux-ha.org. 2008. Último Acesso em 02 de junho de 2008

LOZANO, F.; O que é o Software Livre. Free Software Foundation, 2008. Disponível em http://www.gnu.org/philosophy/free-sw.pt-br.html. Último acesso em Acessado em 19 de novembro de 2008.

MENASCÉ, D. A.; ALMEIDA, V. A. F.; DOWDY, L. W. Capacity planning for WEB peformance: metrics, models & methods. New Jersey: Prentice Hall, 1998.

MOSBERGE, D.; JIN, T.; HTTPERF: A tool for Meassuring WEB Server Performance. Performance Evaluation Review, Volume 26, Number 3. 1998.

NEIL, S.; Safety-Critical Computer Systems. Addison Wesley, 1996.

NEMETH, E.; SNYDER, G.; HEIN, T. Manual Completo do Linux: Guia doAdministrador; Edição 1 ; Makron Books; 2004.

PANKAJ, J. Fault Tolerance in Distributed Systems, Prentice Hall, 1994.

PEREIRA, N. A.; Serviços de Pertinência para Clusters de Alta Disponibilidade, Dissertação de Mestrado em Ciência da Computação, USP. 2004.

POHL, G.; RENNER, M. Munin: Graphisches Netzwerk – und System-Monitoring. Brosch. Open Source Press. 2008.

RED HAT Copyright ©; Red Hat Cluster Suite. 2008. Disponível em http://www.redhat.com/cluster_suite/. 2008. Último acesso em 02 junho de 2008

RISTA, L. C. G.; Uma abordagem de monitoração wireless em um ambiente de cluster com alta disponibilidade. Dissertação de Mestrado, Departamento de Ciência da Computação, UFSC, 2005.

SAUVÉ, J. p.; NICOLLETTI, S. P.; VIGOLVINO, L. R. Melhores Práticas para a Gerência de Redes de Computadores - 1 ª Edição, Campus, 2003.

SAYDAM T.; Magedanz, T. From Networks and Network Management Into Service and Service Management. Guest Editorial, Journal of Network and Systems Management, Vol.4, No.4, 1996

STALLINGS, W.; SNMP, SNMPv2, and CMIP - The Practical Guide to Network-Management Standards. Addison Wesley, 1993.

TANENBAUM, A. S.; Distributed Operating Systems. Primeira Edição, Prentice Hall, 1995.

TANENBAUM, A. S.; Distributed Systems: Principles and Paradigms. Segunda Edição, Prentice Hall, 2007.

TANENBAUM, A. S.; Redes de Computadores. Quarta Edição, Campus. 2003.

TEODORO, G.; TAVARES, T.; et. al. Load Balancing on Stateful Clustered WEB Servers. UFMG. 2004.

WEBER, T; JANSCH-PÔRTO, I; WEBER, R. Fundamentos de tolerância a falhas. Vitória: SBC/UFES, 1990.

WEBER, T.; Um roteiro para exploração dos conceitos básicos de tolerância a falhas. UFRGS, 2002.

ANEXO I – Detalhes da implementação do *cluster*

Instalação e configuração do servidor responsável pelo balanceamento de carga

Pacote Ipvsadm: é usado para criar, manter e inspecionar o servidor virtual, através da tabela do kernel GNU/Linux onde se encontra a lista dos direcionamentos. Com os repositórios oficiais da distribuição escolhida para o estudo de caso, basta em um terminal do sistema com privilégios de super usuário inserir o seguinte comando:

 # apt-get install ipvsadm

Pacote Ultramonkey: implementa os servidores virtuais com GNU/Linux. O procedimento para instalação também é simples. Com o repositório oficial do projeto ultramonkey, HTTP://www.ultramonkey.org/download/3/ selecionado, basta em um terminal do sistema com privilégios de super usuário inserir o seguinte comando:

#apt-get install ultramonkey

Este procedimento irá instalar o Linux Virtual Server com a técnica roteamento direto pronto para configuração. Outros programas que são instalados e prontos para configuração de acordo com arquitetura do capitulo 4 são o *heartbeat* e o *ldirector* com discutido na seção de desenvolvimento.

O primeiro passo é habilitar o parâmetro do kernel chamado *Packet Forwarding* no nodo balanceardor de carga. Isto é necessário para que o ambiente seja capaz de rotear o tráfego para os nodos servidores reais. Para realização desta tarefa é necessário alterar a seguinte linha no arquivo /etc/sysctl.conf para:

 # Enables packet forwardingnet.ipv4.ip_forward = 1

Em seguida, é necessário realizar a configuração do *heartbeat* Os arquivos listados abaixo encontram-se no diretório de configuração do *heartbeat* (/etc/ha.d/). Estes arquivos devem ser idênticos em ambos os nodos balanceadores de carga, primário e secundário (*backup*).

O arquivo ha.cf é responsável por enumerar os nodos do *cluster*, a topologia do

mesmo e a configuração das funcionalidades que são ativadas. A seguir é apresentada uma seqüência do arquivo, comas principais configurações comentadas:

- # /etc/ha.d/ha.cf : linha indicativa do caminho do arquivo;
- logfacility local0: utiliza mensagens para exploração de LOGs. A diretiva local0 é utilizada nos sistemas GNU/Linux;
- bcast eth0: a diretiva ou variável bcast é utilizada para configurar as interfaces de rede;
- mcast eth0 225.0.0.1 694 1 0: configura o caminho de comunicação multcast indicado;
- auto_failback on: determina que, quando o nodo primário voltar com 100% de operacionalidade, ele assumirá novamente o comando do ambiente computacional;
- node lb01: define o nome do nodo balanceador de carga primário. Observação: este nome deve ser exatamente a saída do comando uname -n do balanceador de carga primário;
- node lb02: define o nome do nodo balanceador de carga secundário. Observação: este nome deve ser exatamente a saída do comando uname -n do balanceador de carga secundário;
- respawn hacluster /usr/lib/heartbeat/ipfail: especifica um programa para ser acompanhado durante sua execução. Caso este programa tenha outra saída que o código 100, o programa será automaticamente reiniciado. O programa ipfail indicado na linha de configuração tenta prover detecção de falhas de rede, e então reage, direcionando o *cluster* para os recursos de *failover* quando necessário. Para isso, o ipfail usa a mensagem ICMP através de ping ou grupos de pings. Desde que ambos os nodos possam se comunicar entre si, o ipfail pode detectar com confiabilidade quando um dos seus nodos tem se tornado instável, e compensa;
- apiauth ipfail gid=haclient uid=hacluster: é uma diretiva de autorização. Especifica quais usuários ou grupos de usuários têm permissão de utilizar o programa ipfail. Neste caso apenas usuários de sistema do *cluster* podem ter acesso ao programa.

Um outro arquivo utilizado para implementar o *heartbeat* é o haresources. Ele é responsável por especificar os serviços do *cluster* e quem são os proprietários dos recursos envolvidos no cluster. A seguir é apresentada uma seqüência do arquivo, com as

principais configurações comentadas:

- # /etc/ha.d/haresources: linha indicativa do caminho do arquivo;
- lb01: nome do nodo balanceador de carga primário. Saída do comando uname -n do nodo balanceador de carga primário;
- ldirectord::ldirectord.cf: ativa o *daemon* para monitorar e administrar os servidores virtuais para balanceamento de carga do *cluster* de acordo com a arquitetura do capítulo 4 configurada no arquivo ldirectord.cf;
- LVSSyncDaemonSwap::master: diretiva utilizada para indicar o processo de *failback*. Ou seja, ativar o nodo primário com balanceador de carga do *cluster* novamente, assim que o mesmo estiver 100% operacional;
- Ipaddr2::192.168.0.254/24/eth0/192.168.0.255: diretiva utilizada para configurar serviços de rede. O endereço escolhido para ser o IP virtual é 192.168.0.254. Ou seja, todos os nodos dos servidores reais serão identificado por este IP, dando a impressão de uma única máquina;

O arquivo authkeys também faz parte dos arquivos importantes para implementação do *heartbeat*. Ele contém as informações do *heartbeat* usadas para autenticar os membros do *cluster*. A seguir é apresentada uma seqüência do arquivo, com as principais configurações comentadas:

- # /etc/ha.d/authkeys : linha indicativa do caminho do arquivo;
- auth 3: indica o uso do número de chave 3 para a assinatura dos pacotes. Trara-se de um valor padrão para as configurações básicas de *heartbeat*;
- 3 md5 somerandomstring: md5 é o método de assinatura da chave e somerandomstring é a chave secreta compartilhada, para utilização nas assinaturas dos pacotes. Esta chave foi a padrão do arquivo instalado. Deve ser a mesma nos nodos primário e secundário.

Por fim, mas não menos importante, o arquivo ldirectord.cf fecha a lista dos principais arquivos de configurações do LVS utilizando o projeto Ultramonkey. O objetivo deste arquivo é a declaração das principais diretivas de configuração da arquitetura do capítulo 4. Na verdade o ldiretord, é um *daemon* escrito por Jacob Rief [KOPPER 2008] que tem como função monitorar os serviços dos servidores reais, realizando requisições e verificações de mensagens esperadas pelo servidor virtual. Para

servidores HTTP, como o estudo de caso em questão, é realizada uma requisição de URL conhecido e verifica se a resposta contém uma seqüência esperada.

A seguir é apresentada uma seqüência do arquivo, com as principais configurações comentadas:

- # /etc/ha.d/ldirectord.cf : linha indicativa do caminho do arquivo;
- checktimeout=10: 10 segundos para timeout, caso verificação seja negativa;
- checkinterval=2: 2 segundos para intervalo entre uma verificação e a próxima;
- autoreload=no: define se o ldirector pode ou não verificar continuamente por modificações de configuração no arquivo;
- logfile="local0": log ativado;
- quiescent=yes: quanto o servidor virtual ou um dos servidores reais estiverem fora de funcionamento, o peso será fixado em zero, o que significa que não serão aceites novas requisições;
- virtual=192.168.0.254:80: define um serviço virtual através de um endereço IP (ou host) e a porta (ou serviço);
- real=192.168.0.200:80 gate: servidor real 1. define um serviço verdadeiro através de um endereço IP (ou host) e a porta (ou serviço). O parâmetro gate é o método de encaminhamento dos pacotes;
- real=192.168.0.201:80 gate: servidor real 2;
- real=192.168.0.202:80 gate: servidor real 3;
- fallback=127.0.0.1:80 gate: o servidor virtual é redirecionado se todos os servidores reais falharem.
- service=HTTP: aplicação;
- request="ldirector.html": objeto da aplicação a ser consultado de acordo com a temporização definida;
- receive="Test Page": parâmetro utilizado para comparar o resultado solicitado, caso afirmativo o servidor real é declarado “vivo” ;
- scheduler=rr: diretiva utilizada para definir a política de escalonamento do balanceamento de carga. Foi inicialmente definida a política rr (Round-Robin), que trata as solicitações HTTP dos clientes como uma fila de espera circular, distribuindo de forma linear as requisoções para os servidores reais;
- protocol=tcp: protocolo da camada de transporte utilizado para transitar com os pacotes;
- checktype=negotiate: tipo de verificação para desempenho. O parâmetro negotiate

indica que envia uma solicitação e recebe uma string correspondente.

Configuração dos servidores reais

O primeiro passo na configuração dos servidores reais é personalizar o arquivo /etc/sysctl.conf. Este arquivo é usado para modificar o kernel do GNU/Linux em tempo de execução. Isto será necessário para definir o fluxo dos pacotes transitados entre o ambiente do *cluster* e os clientes da aplicação. As seguintes ações de configuração são necessárias:

- habilitar configurações da tabela ARP através da opção arp_ignore: esta opção controla os critérios que determinam como os pacotes serão processados:
  - net.ipv4.conf.all.arp_ignore = 1
  - net.ipv4.conf.eth0.arp_ignore = 1
  - estas configurações definem que, quando um pedido de nível arp é recebido pela interface de rede eth0, a resposta acontecerá apenas se esse endereço é configurado em eth0. Em particular, não há resposta se o endereço é configurado em lo(loopback). Todas as interfaces envolvidas, como eth1, se estiverem sendo utilizadas, devem ser configuradas: net.ipv4.conf.eth1.arp_ignore = 1;
- habilitar configurações da tabela ARP através da opção arp_announce: Esta opção controla os endereços IPs das fontes (clientes do *cluster*) que podem ser colocados no cabeçalho de solicitação dos pedidos ARP. Quando o nodo gera a requisição são oferecidos vários endereços para atender a requisição. Ou seja, enumera os níveis permitidos.
  - net.ipv4.conf.all.arp_announce =2
  - net.ipv4.conf.eth0.arp_announce =2
  - com essas definições, ao se fazer um pedido ARP enviado através da interface eth0, será sempre utilizado o endereço que esta associado a mesma como o endereço de origem de solicitação ARP. Todas as interfaces envolvidas, como eth1, se estiverem sendo utilizadas, devem ser configuradas: net.ipv4.conf.eth1.arp_announce =2.

Após as configurações do arquivo /etc/sysctl.conf, é necessário configurar nos servidores uma interface loopback virtual para responder pelo ip virtual. O arquivo

/etc/network/interfaces deve ser editado e personalizado com as seguintes configurações:

auto lo:0 # definição do ip virtual para lo
iface lo:0 inet static
address 192.168.0.254
netmask 255.255.255.255
pre-up sysctl -p > /dev/null

Para finalizar, foi criado o arquivo ldirector.html. Este arquivo é requerido pelos nodo balanceador de carga repetidas vezes. O objetivo é verificar se os nodos servidores reais ainda estão executando corretamente. Nesta seção foi apresentada em linhas gerais, os principais procedimento para instalação e configuração do ambiente de *cluster* de alta disponibilidade e alto desempenho, de acordo com a arquitetura do capítulo 4 e com a metodologia da solução adotada na Figura 5.1.

Validação do funcionamento do *cluster*

Um teste preliminar de validação, muito importante, foi a constatação que os balanceadores de carga, primário e secundário, estão totalmente operacionais e prontos para uso com eficiência. Através do uso do comando ip addr sh eth0, executado em modo terminal com status e privilégio de super usuário do sistema operacional, podemos obter um extrato com as principais informações relacionadas as funcionalidades do balanceador de carga. A seguir é apresentado o extrato do balanceador de carga primário e secundário:

- balanceador de carga primário:

2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
link/ether 00:16:3e:40:18:e5 brd ff:ff:ff:ff:ff:ff
inet 192.168.0.1/24 brd 192.168.0.255 scope global eth0
inet 192.168.0.254/24 brd 192.168.0.255 scope global secondary eth0

Pode-se verificar que a última linha apresenta que o IP virtual do *cluster* está ativo na interface de rede eth0 e pronto para uso.

- balanceador de carga secundário:

2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
link/ether 00:16:3e:50:e3:3a brd ff:ff:ff:ff:ff:ff
inet 192.168.0.2/24 brd 192.168.0.255 scope global eth0

Já para o balanceador de carga secundário, o extrato apresenta as principais configurações, entretanto, demonstra que está em espera, pois, não apresenta a posse do IP virtual do cluster implementado pelo Linux Virtual Server.

Foi verificado, também, o estado do *daemon* que ativa o funcionamento do Linux Virtual Server, o *ldirector*. Foi executado o comando *directord ldirectord.cf* em modo terminal com *status* e privilégios de super usuário do sistema operacional no nodo balanceador de carga primário.

O resultado obtido foi: *ldirectord for /etc/ha.d/ldirectord.cf is running with pid: 2697,* exibindo que o processo que ativa o direcionador está rodando e alocado no processador no nodo em questão. Foi realizado a execução do mesmo comando nas mesmas condições no nodo balanceador de carga secundário, também chamado nodo *backup* do balanceador de carga e o resultado foi: *ldirectord is stopped for /etc/ha.d/ldirectord.cf*, que indicando que o processo está parado.

O nodo backup do balanceador de carga está com a configuração em perfeito estado, entretanto em *standby*, aguardando o comando para entrar em funcionamento.

ANEXO II – Execuções do Teste 4 com o ambiente Centralizado

Neste *script* de execução pode-se observar o tempo gasto para processar seis repetições da funcionalidade de geração de relatório de descontos da aplicação ADX no ambiente de teste Centralizado.

Teste 1

Script iniciado em Seg 20 Out 2008 13:41:06 BRT
LAB066:~/relatorios# date
Seg Out 20 13:41:29 BRT 2008
LAB066:~/relatorios# time lynx 192.168.0.200
real 117m39.823s
user 0m10.481s
sys 0m1.568s
LAB066:~/relatorios# date
Seg Out 20 15:39:20 BRT 2008
LAB066:~/relatorios# exit
Script concluído em Seg 20 Out 2008 15:39:23 BRT

Teste 2

Script iniciado em Seg 20 Out 2008 13:39:43 BRT
LAB066:~/relatorios# date
Seg Out 20 13:41:44 BRT 2008
LAB066:~/relatorios# lynx 192.168.0.200
real 116m56.440s
user 0m10.657s
sys 0m1.764s
LAB066:~/relatorios# date
Seg Out 20 15:38:50 BRT 2008
LAB066:~/relatorios# exit
Script concluído em Seg 20 Out 2008 15:38:52 BRT

Teste 3

Script iniciado em Seg 20 Out 2008 13:39:51 BRT
LAB066:~/relatorios# date
Seg Out 20 13:41:56 BRT 2008
LAB066:~/relatorios# time lynx 192.168.0.200
real 115m40.716s
user 0m10.661s
sys 0m1.780s
LAB066:~/relatorios# date
Seg Out 20 15:37:48 BRT 2008
LAB066:~/relatorios# exit
Script concluído em Seg 20 Out 2008 15:37:53 BRT

Teste 4

Script iniciado em Seg 20 Out 2008 13:39:57 BRT
LAB066:~/relatorios# date
Seg Out 20 13:42:14 BRT 2008
LAB066:~/relatorios# time lynx 192.168.0.200
real 116m15.321s
user 0m10.829s
sys 0m1.692s
LAB066:~/relatorios# date
Seg Out 20 15:38:39 BRT 2008
LAB066:~/relatorios# exit
Script concluído em Seg 20 Out 2008 15:38:41 BRT

Teste 5

Script iniciado em Seg 20 Out 2008 13:40:03 BRT
LAB066:~/relatorios# date
Seg Out 20 13:42:22 BRT 2008
LAB066:~/relatorios# time lynx 192.168.0.200
real 116m29.789s

user 0m10.501s

sys 0m1.700s

LAB066:~/relatorios# date

Seg Out 20 15:39:00 BRT 2008

LAB066:~/relatorios# exit

Script concluído em Seg 20 Out 2008 15:39:02 BRT

Teste 6

Script iniciado em Seg 20 Out 2008 13:40:19 BRT

LAB066:~/relatorios# date

Seg Out 20 13:42:32 BRT 2008

LAB066:~/relatorios# time lynx 192.168.0.200

real 115m26.342s

user 0m10.589s

sys 0m1.744s

LAB066:~/relatorios# date

Seg Out 20 15:38:07 BRT 2008

LAB066:~/relatorios# exit

Script concluído em Seg 20 Out 2008 15:38:09 BRT

ANEXO III – Execuções do Teste 4 com o ambiente *storage*

Neste *script* de execução pode-se observar o tempo gasto para processar seis repetições da funcionalidade de geração de relatório de descontos da aplicação ADX no ambiente de *cluster* com nodo *storage* para armazenamento da base de dados da aplicação.

Teste 1

Script iniciado em Qui 23 Out 2008 08:03:21 BRT
LAB066:~/storage# date
Qui Out 23 08:03:54 BRT 2008
LAB066:~/storage# time lynx 192.168.0.254
real 96m51.878s
user 0m15.121s
sys 0m2.036s
LAB066:~/storage# date
Qui Out 23 09:40:54 BRT 2008
LAB066:~/storage# exit
Script concluído em Qui 23 Out 2008 09:40:56 BRT

Teste 2

Script iniciado em Qui 23 Out 2008 08:03:27 BRT
LAB066:~# date
Qui Out 23 08:04:03 BRT 2008
LAB066:~# time lynx 192.168.0.254
real 96m50.250s
user 0m14.641s
sys 0m2.400s
LAB066:~# date
Qui Out 23 09:41:01 BRT 2008
LAB066:~# exit
Script concluído em Qui 23 Out 2008 09:41:04 BRT

Teste 3

Script iniciado em Qui 23 Out 2008 08:03:34 BRT
LAB066:~/storage# date
Qui Out 23 08:04:11 BRT 2008
LAB066:~/storage# time lynx 192.168.0.254
real 96m26.693s
user 0m14.601s
sys 0m2.056s
LAB066:~/storage# date
Qui Out 23 09:40:45 BRT 2008
LAB066:~/storage# exit
Script concluído em Qui 23 Out 2008 15:22:40 BRT

Teste 4

Script iniciado em Qui 23 Out 2008 08:03:40 BRT
LAB066:~/storage# date
Qui Out 23 08:04:19 BRT 2008
LAB066:~/storage# time lynx 192.168.0.254
real 96m42.745s
user 0m14.713s
sys 0m2.156s
LAB066:~/storage# date
Qui Out 23 09:41:09 BRT 2008
LAB066:~/storage# exit
Script concluído em Qui 23 Out 2008 09:41:11 BRT

Teste 5

Script iniciado em Qui 23 Out 2008 08:03:45 BRT
LAB066:~/storage# date
Qui Out 23 08:04:27 BRT 2008
LAB066:~/storage# time lynx 192.168.0.254
real 96m40.602s

user 0m14.825s

sys 0m2.080s

LAB066:~/storage# date

Qui Out 23 09:41:15 BRT 2008

LAB066:~/storage# exit

Script concluído em Qui 23 Out 2008 09:41:17 BRT

Teste 6

Script iniciado em Qui 23 Out 2008 08:03:51 BRT

LAB066:~/storage# date

Qui Out 23 08:04:35 BRT 2008

LAB066:~/storage# time lynx 192.168.0.254

real 96m38.786s

user 0m14.709s

sys 0m2.192s

LAB066:~/storage# date

Qui Out 23 09:41:21 BRT 2008

LAB066:~/storage# exit

Script concluído em Qui 23 Out 2008 09:41:23 BRT

ANEXO IV – Execuções do Teste 4 com o ambiente replicado

Neste *script* de execução pode-se observar o tempo gasto para processar seis repetições da funcionalidade de geração de relatório de descontos da aplicação ADX no ambiente de teste com a base de dados da aplicação replicada nos três nodos servidores reais.

Teste 1

Script iniciado em Ter 21 Out 2008 12:49:06 BRT
Ter Out 21 12:50:47 BRT 2008
LAB066:~/relat_MySQL_distribuido# time lynx 192.168.0.254
real 38m1.187s
user 0m11.365s
sys 0m1.744s
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 13:28:59 BRT 2008
LAB066:~/relat_MySQL_distribuido# exit
Script concluído em Ter 21 Out 2008 13:29:01 BRT

Teste 2

Script iniciado em Ter 21 Out 2008 12:50:12 BRT
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 12:51:00 BRT 2008
LAB066:~/relat_MySQL_distribuido# time lynx 192.168.0.254
real 37m54.203s
user 0m11.501s
sys 0m1.852s
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 13:29:05 BRT 2008
LAB066:~/relat_MySQL_distribuido# exit
Script concluído em Ter 21 Out 2008 13:29:07 BRT

Teste 3

Script iniciado em Ter 21 Out 2008 12:49:52 BRT
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 12:51:13 BRT 2008
LAB066:~/relat_MySQL_distribuido# time lynx 192.168.0.254
real 38m37.950s
user 0m10.693s
sys 0m1.700s
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 13:30:01 BRT 2008
LAB066:~/relat_MySQL_distribuido# exit
Script concluído em Ter 21 Out 2008 13:30:04 BRT

Teste 4

Script iniciado em Ter 21 Out 2008 12:49:46 BRT
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 12:51:24 BRT 2008
LAB066:~/relat_MySQL_distribuido# time linx 192.168.0.254
real 38m1.298s
user 0m11.345s
sys 0m1.784s
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 13:29:42 BRT 2008
LAB066:~/relat_MySQL_distribuido# exit
Script concluído em Ter 21 Out 2008 13:29:43 BRT

Teste 5

Script iniciado em Ter 21 Out 2008 12:50:26 BRT
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 12:51:41 BRT 2008
LAB066:~/relat_MySQL_distribuido# time lynx 192.168.0.254
real 37m59.499s
user 0m11.409s

sys 0m1.724s
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 13:29:47 BRT 2008
LAB066:~/relat_MySQL_distribuido# exit

Script concluído em Ter 21 Out 2008 13:29:49 BRT

Teste 6

Script iniciado em Ter 21 Out 2008 12:50:35 BRT
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 12:51:49 BRT 2008
LAB066:~/relat_MySQL_distribuido# time lynx 192.168.0.254
real 37m57.612s
user 0m10.937s
sys 0m1.868s
LAB066:~/relat_MySQL_distribuido# date
Ter Out 21 13:29:53 BRT 2008
LAB066:~/relat_MySQL_distribuido# exit
Script concluído em Ter 21 Out 2008 13:29:56 BRT