UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

RÉGIS EDUARDO WEIZENMANN GREGOL

RECURSOS DE ESCALABILIDADE E ALTA DISPONIBILIDADE PARA APLICAÇÕES WEB

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

RÉGIS EDUARDO WEIZENMANN GREGOL

RECURSOS DE ESCALABILIDADE E ALTA DISPONIBILIDADE PARA APLICAÇÕES WEB

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Me. Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Recursos de escalabilidade e alta disponibilidade para aplicações Web

Por

Régis Eduardo Weizenmann Gregol

Este Trabalho de Diplomação (TD) foi apresentado às 09:10 h do dia 25 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram argüidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Me. Fernando Schütz
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Me. Paulo Lopes De Menezes
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Nelson Miguel Betzek
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

A folha de aprovação assinada encontra-se na Coordenação do Curso.

AGRADECIMENTOS

Aos meus pais e familiares que estiveram sempre dispostos a me ajudar e sempre me encorajaram para seguir em frente.

Aos meus professores, que contribuíram com os ensinamentos dos quais sempre lembrarei. Se hoje tenho a fonte do saber e a motivação para aprender, devo muito a eles.

Ao meu professor orientador e amigo Fernando Schütz, o qual teve importante participação na realização de muitos dos meus projetos, me incentivando e motivando sempre.

Ao meu veterano e amigo Luiz Eduardo Kowaslki que compartilhou as idéias sobre esse assunto e sempre esteve disposto a me ajudar.

Ao meu supervisor de estágio e amigo Diorgenes Felipe Grzesiuk que me ajudou na configuração dos servidores.

Aos meus amigos Leandro Augusto de Carvalho e Victor Téo que se disponibilizaram em vários momentos para ler e dar opiniões sobre a fundamentação desse trabalho.

“O resultado deve ser raro, os ingredientes têm de ser simples.”
Paulo Leminski

RESUMO

GREGOL, W. Régis Eduardo. Recursos de escalabilidade e alta disponibilidade para aplicações *Web*. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

Com a expansão da tecnologia e o crescente acesso às aplicações existentes na Internet, torna-se um desafio mantê-las disponíveis o tempo todo.

Este trabalho tem como finalidade demonstrar na teoria e na prática o uso de recursos para escalar uma aplicação *Web* e obter uma alta disponibilidade da mesma. Tais recursos podem ser aplicados diretamente na aplicação, por exemplo, com a utilização do Ajax ou também podem ser aplicados na arquitetura de servidores que irão disponibilizar a aplicação, podendo utilizar conceitos de escalabilidade vertical, clusterização e escalabilidade horizontal. Além disso, demonstrar técnicas para aumentar o desempenho da aplicação *Web*.

Palavras-chave: Clusterização. Balanço de Carga. Desempenho.

ABSTRACT

GREGOL, W. Régis Eduardo. Recursos de escalabilidade e alta disponibilidade para aplicações *Web*. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

With the expansion of the technology and increasing access to existing applications on the Internet, it becomes a challenge to keep them available all the time.

This work aims to demonstrate in theory and in practice the use of resources to scale a Web application and get the same high availability. Such features can be applied directly to the application as the use of Ajax but can also be applied to server architecture that will provide the application and can use concepts of vertical scalability, clustering, and horizontal scalability. Furthermore, demonstrate techniques to increase Web application performance.

Palavras-chave: Clustering. Load Balance. Performance.

LISTA DE FIGURAS

LISTA DE SIGLAS

AJAX	Asynchronous Javascript And Xml
FTP	File Transfer Protocol
GLP	General Public License
GNU	GNU is Not Unix
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
JDBC	Java Database Connectivity
LDAP	Lightweight Directory Access Protocol
ROR	Ruby On Rails
RVM	Ruby Virtual Machine
SGBD	Sistema Gerenciador de Banco de Dados
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO

As informações nunca estiveram tão próximas das pessoas após a popularização da Internet, que é fonte de conhecimento, entretenimento, e integração social. Ela se tornou um grande ambiente de desenvolvimento de soluções.

> O número de brasileiros com mais de dois anos de idade que acessaram a Internet a partir de suas casas ou trabalho atingiu a marca de 43,2 milhões de pessoas em março. Isso representa um crescimento de 4,4% em comparação com mês de fevereiro e um aumento de 13,9% em relação ao mesmo mês de 2010. (VEJA DIGITAL, 2011).

O crescimento no acesso à Rede motivou a criação de muitos sites e aplicativos atraentes, tais como redes sociais como o *Facebook*, *Google+*, *Twitter* e muitas outras existentes. Segundo dados da consultoria *comScore*, o *Facebook* teve mais de 141 milhões de visitantes no mês de junho. Em todo o mundo ele possui mais de 500 milhões de usuários cadastrados (G1, 2010).

Com tantos acessos a aplicações disponíveis na Internet, é indispensável que os aplicativos sejam capazes de escalar uma grande demanda de usuários, acessos e transações que possam ocorrer. Sendo assim, os desenvolvedores devem se preocupar para que seus sistemas tenham escalabilidade.

A escalabilidade é uma característica que indica a habilidade de um aplicativo estar preparado para crescer e manipular uma grande porção de trabalho (Gomes, 2010).

Ela pode ser considerada um requisito não funcional de arquitetura de *software* e dependendo da necessidade de um cliente, e acesso de usuários, pode se ter uma aplicação escalável em alta ou baixa disponibilidade (Almeida, 2008).

Aliado a escalabilidade, uma aplicação *Web* deve ter um bom desempenho, pois de nada adianta escalar uma aplicação se a mesma não tiver um desempenho que atenda às necessidades do usuário.

1.1 OBJETIVO GERAL

Desenvolver, como estudo experimental, um sistema *Web* de *microblogging* com o intuito de demonstrar técnicas que podem ser utilizadas visando a escalabilidade em um projeto on-line.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Desenvolver um referencial teórico sobre escalabilidade, tecnologias *Web* e desempenho de sistemas on-line;
- Analisar e projetar o estudo experimental;
- Desenvolver uma aplicação *Web* que servirá como estudo experimental, implementando nesta os mecanismos de escalabilidade;
- Testar na prática a escalabilidade da aplicação após a utilização dos recursos de otimização;
- Apresentar os resultados obtidos.

1.3 JUSTIFICATIVA

Na era da informação, onde as tecnologias surgem freqüentemente e de maneira muito rápida, o avanço das aplicações existentes na “nuvem” vem se expandindo (Silva, 2006).

Segundo Steppat (2009) “Nas grandes aplicações *Web* é cada vez mais comum a quantidade de informações ser enorme, e ainda temos uma certeza:

amanhã teremos mais dados para armazenar. Como lidar com isso de maneira eficiente?”

Independente dos recursos tecnológicos a serem utilizados no desenvolvimento de uma aplicação *Web*, é necessário prever que uma aplicação pode crescer.

Por volta do ano de 2008, gerou-se uma discussão sobre o *microblogging twitter* que era fenômeno na Internet. Ele parecia estar sofrendo altos índices de popularidade e teria crescido demasiadamente, com isso, a aplicação ficava indisponível para o acesso de usuários. (Galves, 2008).

Segundo Noronha (2010) “Sistemas modulares e escaláveis oferecem os benefícios da maior agilidade e a habilidade de implantar os recursos necessários quando novas oportunidades aparecem e as condições são alteradas.”

Para atender uma demanda crescente de uso e de informações faz-se necessário a adoção de certos recursos que visam propiciar uma maior escalabilidade da aplicação. Tais recursos podem ser utilizados de diversas maneiras, como físicos, com aumento de *hardware* de um servidor para atender a demanda, ou podem ser lógicos, como a clusterização da aplicação ou uma reestruturação de código para melhor desempenho. Com isso, pode se obter uma aceitação maior de uma aplicação perante aos usuários.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho possui cinco capítulos, sendo que o primeiro trata sobre a contextualização do tema abordado e também define os objetivos e a justificativa do projeto.

O segundo busca fornecer um conceito teórico sobre os recursos que foram utilizados no desenvolvimento do mesmo, como escalabilidade vertical, clusterização, escalabilidade horizontal.

O capítulo seguinte demonstra um estudo experimental, onde foi verificado na prática as soluções para utilizar recursos de escalabilidade na aplicação desenvolvida.

No quarto capítulo é feita a conclusão final sobre o trabalho e as sugestões para trabalhos futuros.

Por fim, o último capítulo contém as referências bibliográficas que foram utilizadas para embasamento teórico do projeto.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo tem como objetivo fornecer um referencial teórico sobre os recursos utilizados para o desenvolvimento do estudo experimental. Os conceitos abordados são: escalabilidade vertical, escalabilidade horizontal, balanço de carga, clusterização, entre outros recursos para escalar aplicações *Web*.

2.1 ESCALABILIDADE EM APLICAÇÕES WEB

Em 1990 a *World Wide Web* (a *Web*) foi construída sobre uma infra-estrutura física da Internet, ela mudou radicalmente a disponibilidade de informação, possibilitando a disseminação da informação de forma digital (Levene, 2010). De lá pra cá, o rápido avanço tecnológico fez com que fosse indispensável à utilização de aplicações *Web* não só para buscar informações, mas também para a realização de tarefas online, onde muitas pessoas preferem acessar aplicativos na “nuvem” do que instalarem em seus computadores (Lingham, 2007).

Com um grande número de acessos à *Web*, torna-se indispensável desenvolver soluções eficientes e que estejam disponíveis mesmo com aumento de acessos simultaneamente em um sistema. Nesse contexto, se encaixa a escalabilidade, uma aplicação *Web* deve estar bem projetada para suprir à demanda de acessos em grande fluxo.

A escalabilidade pode ser definida como “a facilidade com que um sistema ou componente pode ser modificado para atender a área de um problema”. Ela não está contida em velocidade de um sistema. Desempenho e escalabilidade são conceitos diferentes, um sistema pode ter um alto desempenho e não escalar, conseqüentemente, pode se ocorrer o inverso. Também não restringe quanto à linguagem de programação utilizada para o desenvolvimento de um sistema. (Henderson, 2006).

Projetar aplicações escaláveis, pode depender de vários fatores. Ao incorporar limites físicos pode-se considerar:

- Escalabilidade horizontal;
- Escalabilidade vertical.

2.1.1 Escalabilidade Horizontal

Entende-se por escalabilidade horizontal (*scale out*) adicionar nós a uma arquitetura de sistema. Para Ferreira (2010) “adicionar um novo servidor e um sistema de *software* que permita a distribuição do trabalho entre múltiplas máquinas”.

A Figura 1 demonstra um exemplo:

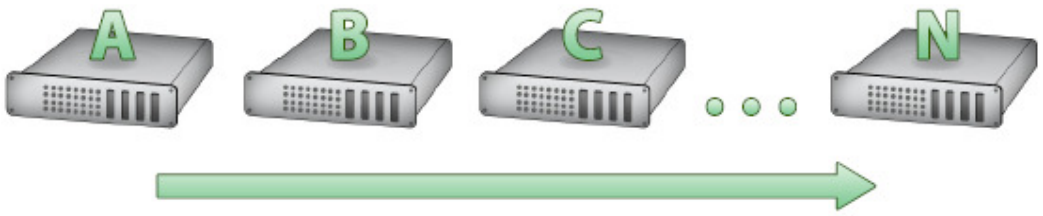

Figura 1 - Esquema de Escalabilidade Horizontal.
Fonte: GoGrid (2010, p.06).

Escalar horizontalmente é acoplar várias máquinas individuais compartilhando o mesmo recurso e fazendo o mesmo trabalho, servindo a mesma aplicação. (GoGrid, 2010).

2.1.2 Escalabilidade Vertical

Entende-se por escalabilidade vertical (*scale up*) aumentar o processamento do *hardware* de um servidor apenas, adicionando mais recursos como memória ou um disco rígido mais rápido para atender uma demanda crescente de requisições e armazenamento em uma aplicação. (Ferreira, 2010).

A Figura 2 ilustra o esquema de escalabilidade vertical.

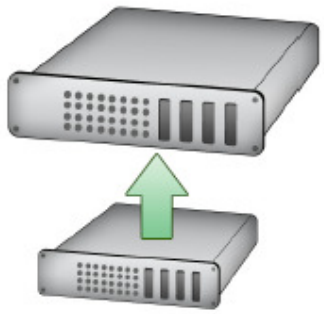

Figura 2 - Esquema de Escalabilidade Vertical.
Fonte: GoGrid (2010, p.06).

Por outro lado, não necessariamente deve-se adicionar recursos de *hardware* ao servidor, pode-se também substituir o servidor antigo por outro mais potente, mais novo. (GoGrid, 2010).

2.1.3 Balanço De Carga e Alta Disponibilidade

O balanceamento de carga pode ser definido como a divisão de uma quantidade de trabalho entre dois ou mais servidores. Também pode ser definido como processo de distribuição de solicitações de serviços em um grupo de servidores. (Natário, 2011).

A distribuição de tarefas consiste em aumentar a escalabilidade adicionando mais servidores em uma arquitetura de rede. Também melhora o desempenho de processamento, pois em uma estrutura balanceada de carga pode-se direcionar as solicitações de serviço para servidores que estão com pouca carga de trabalho. Com isso, se obtém uma aplicação com alta disponibilidade. (Natário, 2011).

A Figura 3 demonstra um exemplo de balanceamento de carga:

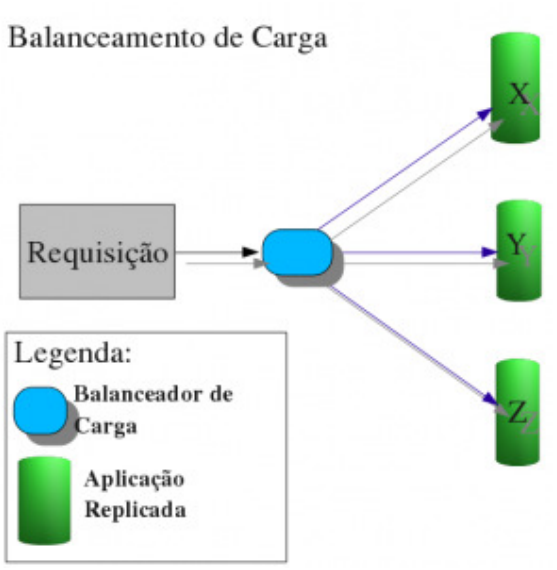

Figura 3 - Balanceamento de carga.
Fonte: SourceForge (2010).

O balanceamento de carga tem ligação direta com a disponibilidade de uma aplicação.

> **Disponibilidade**: É a capacidade do sistema em responder às requisições feitas pelos clientes (usuários, outros sistemas, monitoramento) conforme as regras que são estabelecidas para cada sistema. Ou seja, é capacidade do serviço estar disponível quando requisitado, e com capacidade de resistência a falhas. Podemos medir/classificar a disponibilidade em baixa, média e alta, e o que determinará os valores para esta classificação serão os requisitos do próprio sistema. Exemplificando, um sistema WEB, pode ser classificado como de Alta-Disponibilidade e com os valores de disponibilidade em 24X7 (vinte e quatro horas por dia e sete dias por semana) devendo estar "on-line" e respondendo às requisições continuamente e sem interrupção durante todo o seu ciclo vida dentro dos parâmetros definidos como limites e tolerância pelo sistema. (SOURCEFORGE, 2010).

Utilizando balanceadores de carga, pode-se ter aplicações já desenvolvidas trabalhando de forma isolada e paralela disponibilizando o conteúdo da aplicação com mais garantia. (SourceForge, 2010).

2.1.4 Clusterização

Para entender os conceitos de clusterização, deve-se entender o conceito básico de *Cluster*.

Segundo Alecrim (2004) "*Cluster* pode ser definido como um sistema onde dois ou mais computadores trabalham de maneira conjunta para realizar processamento pesado." Ou seja, os computadores trabalham como se fossem um.

A Figura 4 demonstra um esboço de clusterização de aplicações *Web*.

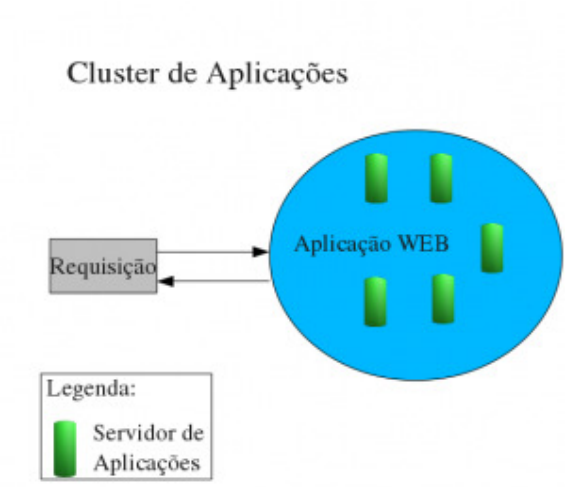

Figura 4 - Cluster de aplicações *Web*.
Fonte: SourceForge (2010).

A clusterização de aplicações pode ser considerada uma evolução do balanceamento de carga, pois permite o balanceamento e a integração da aplicação replicada. Ela consiste em montar um *cluster* com vários servidores de aplicação, podendo estar no mesmo equipamento. Os servidores podem compartilhar alguns

recursos, porém cada servidor atenderá suas próprias requisições. (SourceForge, 2010).

2.2 A LINGUAGEM RUBY

Ruby é uma linguagem balanceada entre programação funcional e imperativa, inspirada em outras linguagens de programação como *Perl*, *Smalltalk*, *Eiffel*, *Ada* e *Lisp*. Foi criada por *Yukihiro Matsumoto* ou simplismente "Matz" em 1995 no Japão. (Lang, 2011).

Segundo Oliveira (2005) "*Ruby* é uma linguagem de *script* interpretada para programação orientada a objetos de um modo fácil e rápido". Ela é *open source*, o que significa que tem seu código fonte aberto, e possibilita que programadores possam desenvolver melhorias e outras funcionalidades para a linguagem, é também totalmente gratuita e segue a licença MIT[1].

Em *Ruby* tudo é objeto, isso a diferencia de muitas outras linguagens que possuem tipos primitivos como números. Também não é necessária a declaração de variáveis, pois ela usa convenções simples de nomenclatura para designar o escopo de variáveis. Suas variáveis podem ser declaradas como:

- **Global**: Utiliza-se o caractere cifrão, exemplo: "$var";
- **Local**: Pode-se definir uma variável local utilizando o seu nome, exemplo: "var";
- **Instância**: Uma variável de instância pode ser declarada com a utilização do caractere arroba, exemplo: "@var".

Para atribuir valores a uma variável utiliza-se o caractere que representa igual. Exemplo: "var = 50".

[1] Licença que concede permissão de título gratuito, para que qualquer pessoa obtenha uma cópia deste *software* e arquivos de documentação associados a ele. (OSI, 2011).

2.3 O FRAMEWORK RUBY ON RAILS

O *Rails* foi desenvolvido em julho de 2004, com o objetivo de permitir desenvolvimento ágil, com poucas linhas de código, tendo uma alta produtividade, possibilitando gerar bons resultados em uma aplicação *Web*.

Segundo Hansson e Thomas (2008), "O *Ruby On Rails* é um *framework* que torna mais fácil desenvolver, instalar e manter aplicativos *Web*".

Rails é também chamado de *meta-framework* porque surgiu de uma união de alguns *frameworks*. Dentre eles:

- **Active Record**: Em *Rails*, pode ser considerado um *framework* que contém mapeamento objeto-relacional, entre a aplicação e o banco de dados;
- **Action Pack**: *framework* que visa gerar as *Actions Views* (onde vão os códigos HTML, XML, *Javascripts*, e outros), e os *Actions Controllers* (responsável pelo controle de regras de negócio);
- **Action Mailer**: *framework* responsável por disponibilizar serviço de entrega e recebimento de emails, capaz de realizar diversas operações apenas com chamadas de entregas de correspondência;
- **Active Support**: *framework* que contém coleções de diversas classes e extensões de bibliotecas, consideradas úteis para uma aplicação em *Ruby On Rails*;
- **Active WebServices**: *framework* que provê uma maneira de publicar APIs que se comuniquem com o *Rails*.

O *Ruby On Rails*, segue o princípio *DRY* (*Don´t Repeat Yourself*) que significa "não se repita". Este conceito visa aumentar a produtividade do desenvolvedor, fazendo com que não seja preciso repetir códigos em sua aplicação.

2.4 AJAX ON RAILS

Segundo Niederauer (2007) “A palavra Ajax vem da expressão Asynchronous *JavaScript* and XML. É o uso sistemático de *JavaScript* e XML (entre outras tecnologias) para tornar o navegador mais interativo com o usuário, utilizando-se solicitações assícronas de informações.” O autor ainda completa que “podemos utilizar o Ajax para fazer uma solicitação ao servidor *Web* sem que seja necessário recarregar a página que estamos acessando.”

De modo tradicional (sem Ajax) uma requisição HTTP a uma página da *Web* acontece de forma direta entre cliente e servidor. A Figura 5 esboça um exemplo de uma requisição tradicional.

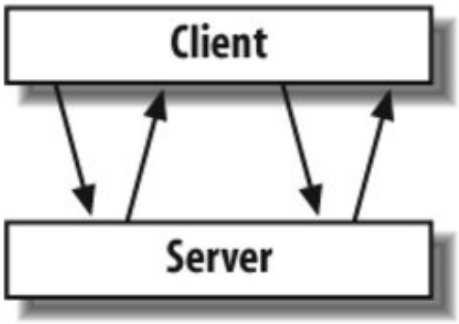

Figura 5 - Requisição cliente-servidor modelo tradicional.
Fonte: Raymond (2006, p.02).

Diferentemente do modelo tradicional, suponha-se que um cliente requisite a ação de um formulário em uma página *Web*. Ele não precisa esperar que a página seja recarregada, pois o modelo Ajax se encarrega de enviar somente a requisição do formulário ao invés da página completa. O tempo de espera do usuário é praticamente zero (Raymond, 2006).

A Figura 6 demonstra um exemplo de requisição com Ajax.

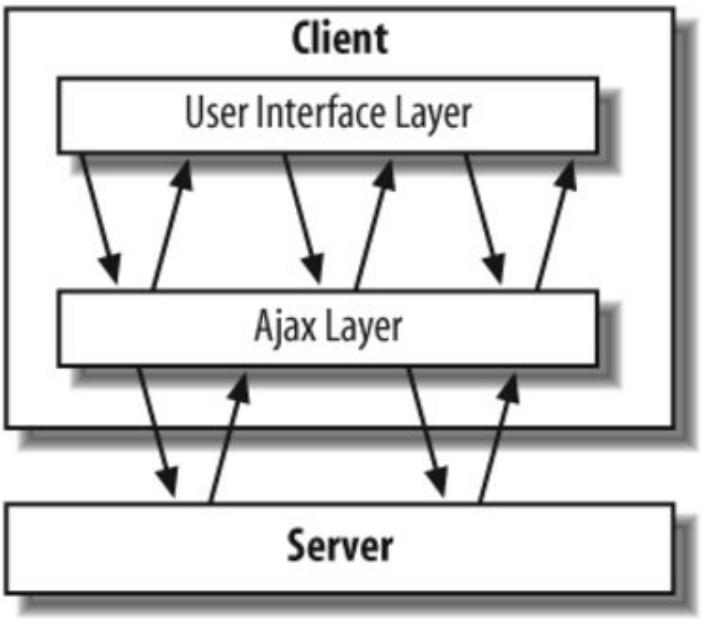

Figura 6 - Requisição cliente-servidor modelo Ajax.
Fonte: Raymond (2006, p.03).

Pode se observar na Figura 6 que quando uma ação é disparada pelo cliente ao servidor ela é primeiramente entregue à camada de Ajax. Essa se responsabiliza por interagir com o servidor buscando somente a resposta à solicitação.

2.5 MYSQL

Segundo Buyens (2002) “MySQL é um *software* de banco de dados que suporta a linguagem de consulta de banco de dados chamada SQL. A SQL é um padrão de comunicação com banco de dados.” Também pode ser considerado um servidor de banco de dados, contém um conjunto de APIs robusto, pois suporta várias linguagens de programação como C, *C++*, *Eiffel*, *Java*, *Perl*, *PHP*, *Python*, *Ruby* e muitas outras. (Buyens, 2002).

Para Muto (2004) “O banco de dados MySQL pode ser considerado um dos projetos de *softwares* de origem mais quente desde o Linux”. Pois compete seriamente com os maiores sistemas de banco de dados existentes no mercado.

> O banco de dados MySQL se tornou a base de dados mundial de código aberto mais popular por causa de seu alto desempenho, alta confiabilidade e facilidade de uso. É também o banco de dados de escolha para uma nova geração de aplicações construídas na pilha LAMP (Linux, Apache, MySQL, PHP / Perl / Python.) Muitas das grandes organizações e de crescimento mais rápido do mundo, incluindo Facebook, Google, Adobe, Alcatel Lucent e Zappos confiar em MySQL para economizar tempo e dinheiro ao optimizar os seus sites de alto volume Web, sistemas críticos aos negócios e pacotes de software. (MYSQL, 2011).

O MySQL é um *software open source* fornecido pela licença GPL[2], encontra-se na versão 5.5.16 até o presente momento. Além da versão *free*, o MySQL possui licença comercial, onde é possível obter suporte diferenciado dos desenvolvedores.

2.6 SERVIDORES DE APLICAÇÃO

Pode se dizer que um servidor de aplicação *Web* é um *software* projetado para fornecer uma infraestrutura de serviços para a execução de aplicações distribuídas, ou seja, a sua vantagem em relação ao modelo cliente-servidor é oferecer serviços já implementados aos desenvolvedores de aplicações, fazendo com que esses se preocupem mais com a codificação e a lógica de negócio de seus aplicativos do que com aspectos de infraestrutura. (Application Servers, 2003).

De certo modo, utilizar esses serviços diminui a complexibilidade no desenvolvimento de uma aplicação, pois eles controlam o fluxo de dados, implementam recursos de performance e gerenciam a segurança. Os servidores de aplicação utilizam a arquitetura de n-camadas (cliente, servidor de aplicação e banco de dados) o que permite aproveitar melhor as características desses componentes. A camada de *Front-End* pode ser considerada o *browser* que é disposto para apresentação e validação da aplicação. A segunda camada é a disposição da aplicação executada no servidor. A última camada representa o servidor de banco de dados. (Application Servers, 2003).

A Figura 7 apresenta um exemplo de arquitetura de servidor de aplicação.

[2] Licença Pública Geral é utilizada pela maioria dos programas que não são parte do Projeto GNU. (GNU, 2008).

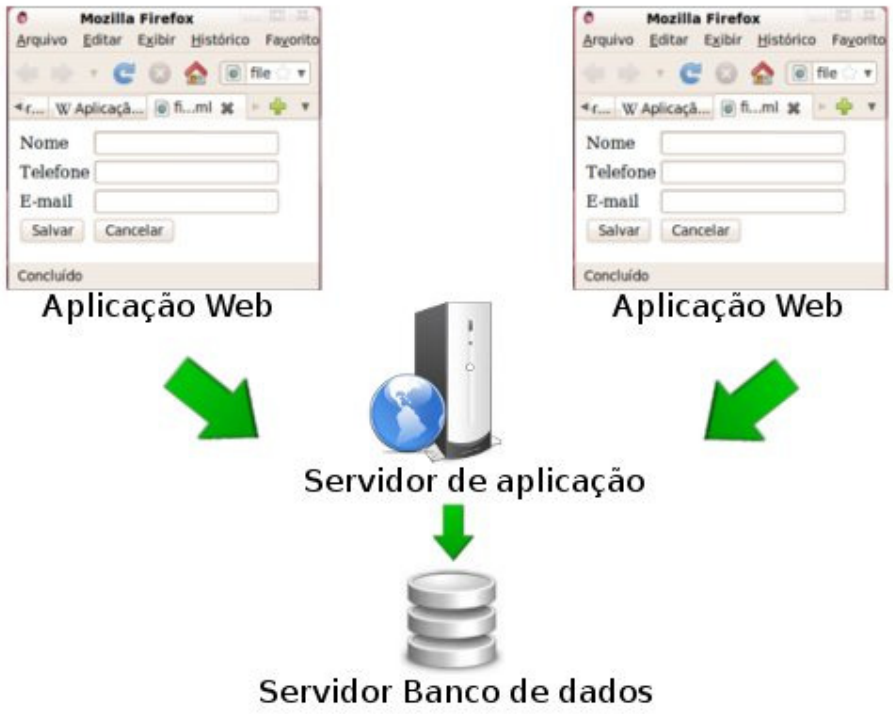

Figura 7 - Arquitetura de servidor de aplicação.
Fonte: JavaEE: Conceitos Básicos (2010).

Alguns servidores de aplicação possibilitam o compartilhamento dos seus recursos permitindo:

- Balanceamento de Carga: distribuir as requisições dos clientes;
- Tolerância a Falhas: políticas de recuperação e clusterização;
- Gerenciamento de Componentes: tempo de sessão;
- Gerenciamento de Transações: integridade de transação com banco de dados;
- Segurança: impedir a invasão na aplicação.

Os servidores de aplicação são capazes de servir em multiplataforma, ou seja, a maioria deles pode rodar aplicações em qualquer sistema operacional, como Windows ou Linux. (Application Servers, 2003).

2.7 WEBRICK

O *Webrick* pode ser considerado um servidor de aplicações *Web*, escrito na linguagem *Ruby* em que sua classe abstrata disponibiliza uma maneira simples de atender as requisições HTTP. (Segment7, 2011).

Desenvolvido pelo japonês Takahashi Masayoshi Gotou Yuuzou juntamente com desenvolvedores *Rails*, faz parte da biblioteca 1.8, e também da biblioteca 1.9 do *Ruby* (Santoso, 2004).

A última atualização do *webrick* foi realizada em maio de 2010 de acordo com o *ruby-doc.org*.

2.8 MONGREL

O servidor *mongrel* é uma pequena biblioteca escrita em *Ruby* que implementa um servidor HTTP muito rápida para aplicações *Web*. Foi destinado para iniciar uma aplicação atrás de um servidor *Web* mais completo como Apache ou *Nginx*. (Fauna, 2008).

Sua instalação pode ser feita através de *Gems* (bibliotecas) do *Ruby*, rodando o comando "*gem install mongrel*" em um terminal.

Para iniciar uma aplicação *Web Rails* utilizando o *mongrel* como servidor, deve-se utilizar o comando "*mongrel_rails start*" na raiz da aplicação. O *mongrel* utiliza a porta 3000 para rodar aplicações *Rails*.

2.9 APACHE

O Apache HTTP *Server Project*, foi desenvolvido com o objetivo de manter um servidor *Web* que fosse livre (*open-source*), seguro, eficiente, rápido, flexível e que estivesse em sincronia com os padrões HTTP. (Apache, 2011).

O apache teve seu inicio na década de 90, em fevereiro de 1995 criou-se o Apache *Group*, contando com oito desenvolvedores que visavam melhorar o projeto. Começou-se então apoiar a comunidade de usuários do Apache projetando uma nova arquitetura de servidor que fosse modular e extensível. Sua primeira versão em dezembro de 1995. O servidor Apache teve um grande sucesso no seu início, contando com a participação de usuários que contribuíram com idéias, código e documentação para o projeto. Em 1999, Apache *Group* formou a Apache *Software Fundation* para fornecer suporte do servidor Apache. A fundação colocou o *software* como base para desenvolvimento futuro, e expandiu o número de projetos *open source* pela fundação. Atualmente a Apache conta com um grupo fiél de colaboradores. (Apache, 2011).

O Apache é um dos servidores *Web* mais popular no mundo desde 1996 com a explosão da Internet. Até o presente momento encontra-se na versão 2.2.21 lançada em 13 de setembro de 2011. (Apache, 2011).

2.10 APACHE JMETER

O JMeter é uma aplicação *desktop* desenvolvida na linguagem Java e de código fonte aberto. Foi criado pela Apache *Software Foundation*, projetado para testar o comportamento e o desempenho de uma aplicação *Web*, ele simula cargas de dados em um servidor de aplicações, podendo realizar testes nos protocolos HTTP, JDBC, FTP, LDAP e MAIL. Ele pode ser considerado um monitor de teste, pois gera gráficos dinâmicos ao decorrer de um teste em uma aplicação (Jakarta, 2011).

O JMeter possuí alguns elementos básicos, dentre eles:

- Plano de teste: descreve uma série de passos que o JMeter irá executar quando for iniciado;

- Área de trabalho: os elementos filhos não serão executados, ou seja, são utilizados apenas como área de armazenamento temporário para os testes, não aparecendo ao usuário.

A Figura 8 mostra um *screenshot* da tela inicial do Apache JMeter:

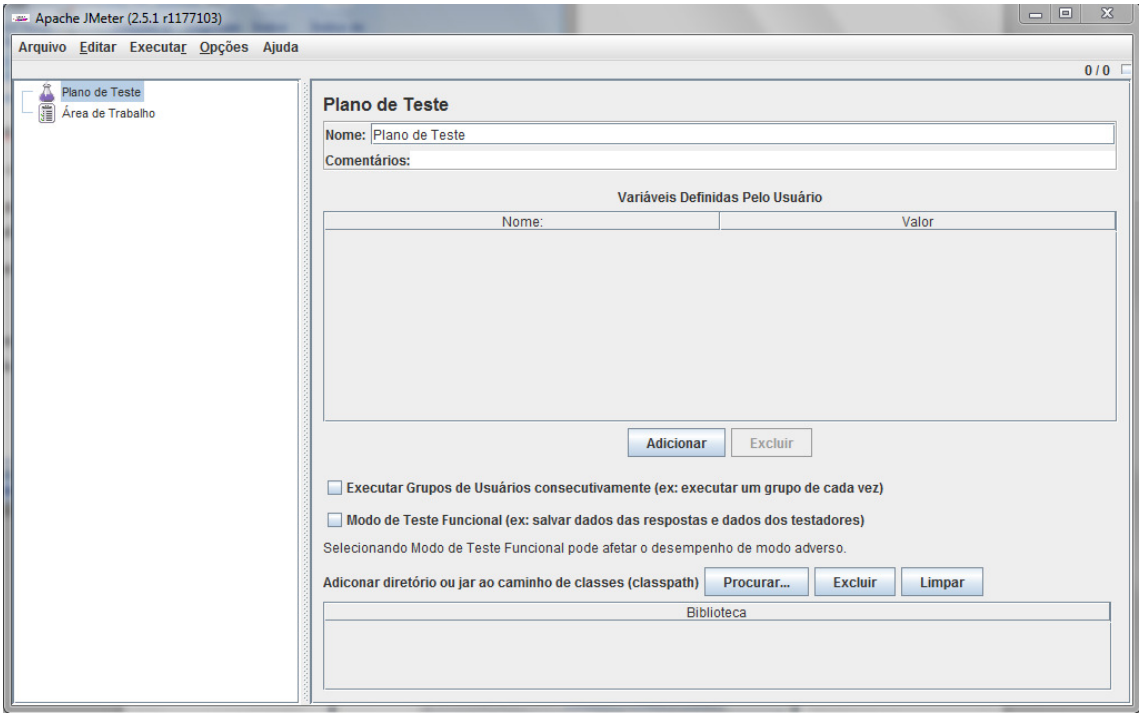

Figura 8 - Tela Inicial do Apache JMeter.

Segundo Freitas (2008) “Através dessa ferramenta é possível simular, por exemplo, diversos usuários acessando simultaneamente uma página. Como resultado, o JMeter gera relatórios e gráficos, facilitando a compreensão sobre como a aplicação reagiu.” A autora ainda completa que o JMeter possui os seguintes elementos:

- Grupo de usuários: onde se define o número de usuários concorrentes, ou seja, o número de vezes que o teste deverá se repetir e o tempo total da execução;

- Testador: contém os elementos responsáveis pelas requisições dos serviços HTTP, *Request*, FTP *Request*, JDBC *Request*);

- Ouvinte: contém os elementos necessários para criar relatórios de testes;

- Temporizador: com esse elemento é possível contabilizar pausas entre os grupos de usuários, pois o JMeter executa todos os grupos de usuário em sequência e sem pausa, simulando uma sobrecarga no servidor;

- Asserções: podem ser adicionadas aos grupos de usuários asserções para verificar se a resposta enviada pelo servidor é a esperada, permitindo o uso de expressões regulares;

- Elemento de configuração: contém os elementos responsáveis por armazenar configurações padrões para os grupos de usuários e outros elementos de controle;

- Controlador Lógico: contêm elementos responsáveis pela lógica dos testes. Dentre eles:

    - Controlador de uma única vez: componentes aninhados são executados apenas uma vez;

    - Controlador de intercalação: executa um grupo de usuários alternadamente;

    - Controlador de iteração: executa repetidamente os componentes conforme o número de vezes especificado;

    - Controlador Simples: usado para agrupar requisições.

3 MATERIAIS E MÉTODOS

Para este trabalho foi desenvolvida uma aplicação *Web* para testes, que se baseia em uma pequena rede social, mais especificamente um *microblogging* on-line, onde um usuário poderia se cadastrar, efetuar a autenticação para entrada no sistema (*login)* e interagir com o sistema postando mensagens sobre o que esta pensando ou o que está fazendo.

Também utilizou-se do *software* de testes JMeter para realizar os testes de escalabilidade na aplicação simulando quantidades de usuários simultâneos acessando a aplicação.

Para o desenvolvimento dos testes e desenvolvimento dos recursos de escalabilidade na aplicação on-line, foi realizado uma ampla pesquisa bibliográfica, além de consultas na Internet em sites especializados no assunto a fim de dar subsídios suficientes à implementação dos recursos utilizados no estudo.

A pesquisa das tecnologias e linguagens foi de natureza aplicada e com objetivo exploratório, sendo a que mais se aplicava, pois segundo Silva e Menezes (2001), a pesquisa aplicada “objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos”, e a pesquisa exploratória “visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses, envolve levantamento bibliográfico”.

Para demonstrar as configurações e a implementação dos recursos de escalabilidade em uma aplicação on-line vão ser descritas as ferramentas utilizadas e as configurações realizadas para o desenvolvimento de testes na aplicação.

3.1 REQUISITOS DE ESCALABILIDADE NA APLICAÇÃO

A aplicação *Web* desenvolvida tem a necessidade de se manter on-line e atender as requisições mesmo com grandes quantidades de acessos dos usuários:

- Otimizar a aplicação, diminuindo requisições de *refresh* na página principal da aplicação através da implementação de recursos como Ajax;
- Otimizar o desempenho da aplicação aumentando a rapidez de navegação obtendo assim a satisfação dos usuários;
- Atender a um número crescente de visitas simultâneas de usuários e se manter disponível para satisfação dos usuários.

3.2 GEDIT

Para desenvolvimento da aplicação *Web* utilizou-se como IDE de desenvolvimento o Gedit, um aplicativo editor de texto que é disponibilizado com as distribuições GNOME do Linux sua licença de *software* GLP *General Public License* liberado pela GNU. (Gnome, 2011).

O Gedit possui um sistema de *plugins* flexível que pode ser usado para adicionar novos recursos dinamicamente. Utilizou-se *plugins* para aplicações *Ruby on Rails*. Com a utilização desses *plugins*, o Gedit passa a ser denominado Gmate pela semelhança com o TextMate (editor de texto padrão do MacOS).

3.3 AJAX NA APLICAÇÃO

Para deixar a aplicação de *microblogging* mais rápida e mais escalável, utilizou-se técnicas de programação com Ajax. A Figura 9 demonstra a configuração desse recurso utilizado na aplicação.

- A linha 1 demonstra que será inserido um texto no topo da página HTML de posts;

- A linha 2 demonstra a utilização de um efeito visual de *highlight* que consiste em realçar a cor de fundo de um elemento dando destaque para quando o usuário postar alguma informação.

```
1  page.insert_html :top, :posts, :partial => @post
2  page[@post].visual_effect :highlight
```

Figura 9 - Trecho de código implementação do Ajax.

3.4 ARQUITETURA DE HARDWARE E SOFTWARE

Para servir a aplicação *Web* desenvolvida, utilizou-se as seguintes arquiteturas de *hardware* e *software*.

3.4.1 Primeiro Servidor

O primeiro servidor foi composto pelas seguintes configurações:

- Servidor HP *ProLiant* ML 330, processador *Intel Xeon* 2.8 Ghz, 2 Hds Samsung 40Gb com 7200 rpm em *Raid* 0, 2 pentes de memória de 1Gb Kingston 800mhz.
- Sistema Operacional CentOS 6;
- Ambiente instalado com RVM (*Ruby Virtual Machine*), *Ruby* versão 1.8.7, *Rails* versão 2.3.2.

3.4.2 Segundo Servidor

O segundo servidor foi composto pelas seguintes configurações:

- Servidor HP *ProLiant* ML 330, processador *Intel Xeon* 2.8 Ghz, 2 Hds Samsung 80Gb com 15000 rpm em *Raid* 0, 2 pentes de memória de 1Gb Kingston 800mhz.
- Sistema Operacional CentOS 6;
- Ambiente instalado com RVM (*Ruby Virtual Machine*), *Ruby* versão 1.8.7, *Rails* versão 2.3.2.

3.4.3 Computador De Teste

O computador utilizado para efetuar os testes no servidor contém as seguintes configurações:

- Dell *OptiPlex* 755, processador *Intel* 2.2 Ghz, 1 Hd Samsung de 160Gb de 7200 rpm, 4 Gb de memória *MarkVision* 667Mhz;
- Sistema Operacional Ubuntu 9.10;
- Ambiente instalado com JMeter (*software* usado para os testes).

4 RESULTADOS E DISCUSSÕES

Os testes executados pretendem fornecer um referencial de dados representativos para entendimento dos recursos ou componentes que influenciam na escalabilidade de uma aplicação *Web*.

Utilizou-se o *software* JMeter no computador de testes, onde no Plano de Teste foi adicionado o componente de Grupos de Usuários (usuários virtuais), nesse componente adicionou-se um Testador o componente de Requisição HTTP.

Para demonstrar o resultado dessas requisições utilizou-se duas formas de visualização (Jakarta, 2011):

- Relatório de sumário: cria uma linha de tabela e expõe dados do teste, dentre eles:
  - Rótulo: o rótulo da amostra;
  - # Amostras: o número de usuários virtuais (Grupo de usuário) definido;
  - Média: A média de tempo decorrido para as amostras das requisições em segundos;
  - Mín: O menor tempo decorrido para as amostras das requisições em segundos;
  - Max: O mais longo tempo decorrido para as amostras das requisições em segundos;
  - Desvio Padrão: o desvio padrão do tempo decorrido das requisições;
  - % de Erro: Porcentagem de pedidos com erros;
  - Vazão: O rendimento das requisições na aplicação, que é medido em segundos/minutos/horas.
  - KB/s: O fluxo de dados por segundo medido em *kilobytes*;

- Média de *Bytes*: Tamanho médio de resposta das amostras.

• Gráfico de resultados: gera um gráfico simples, que traça amostras ponto a ponto e exibe na tela pontos de:

- Dados: Traça valores de dados reais;
- Média: Traça a média dos dados;
- Mediana: Traça a mediana dos dados;
- Desvio: Traça o desvio padrão dos dados;
- Vazão: Traça o número de amostras por unidade/tempo.

A Figura 10 demonstra como ficaram dispostos os componentes no JMeter.

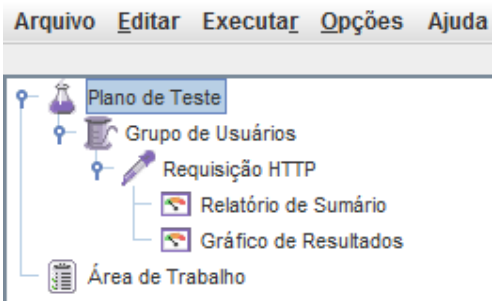

Figura 10 - Componentes Plano de Teste JMeter.

Para determinar a escalabilidade em uma aplicação *Web*, devem-se executar testes de carga que possibilitem determinar o comportamento da aplicação em condições normais e em altos picos de carga. É recomendável começar um teste com um pequeno número de usuários virtuais e depois incrementar mais usuários (Campos, 2011).

Para a realização dos testes definiu-se três grupos de usuários:

• 400 usuários: definido como população de amostra baixa;

- 25000 usuários: definido como população de amostra média;
- 50000 usuários: definido como população de amostra alta.

4.1 PRIMEIRO TESTE

No primeiro teste utilizou-se o primeiro servidor com suas configurações básicas iniciais descritas no item 3.5.1 desse trabalho e o servidor *webrick* como servidor de aplicação.

4.1.1 400 Usuários

As Figuras 11 e 12 demonstram os resultados através do Relatório de Sumário e o Gráfico de Resultados do teste com o Grupo de 400 usuários virtuais.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	400	9464	87	21005	7719,97	27,25%	18,1/sec	33,41	1887,9
TOTAL	400	9464	87	21005	7719,97	27,25%	18,1/sec	33,41	1887,9

Figura 11 - Relatório de Sumário 400 usuários.

Pode-se perceber na Figura 11, na vazão um rendimento de 18,1/segundos por requisição, também uma porcentagem de erro de quase 30% no atendimento das requisições.

Também pôde-se observar no Gráfico de Resultados (Figura 12) a disponibilidade na vazão que representou um resultado de 1.087.252/minutos. Apenas com esse número, não se pode chegar a uma conclusão e afirmar se esse resultado foi bom ou ruim, pois esse resultado é do primeiro teste.

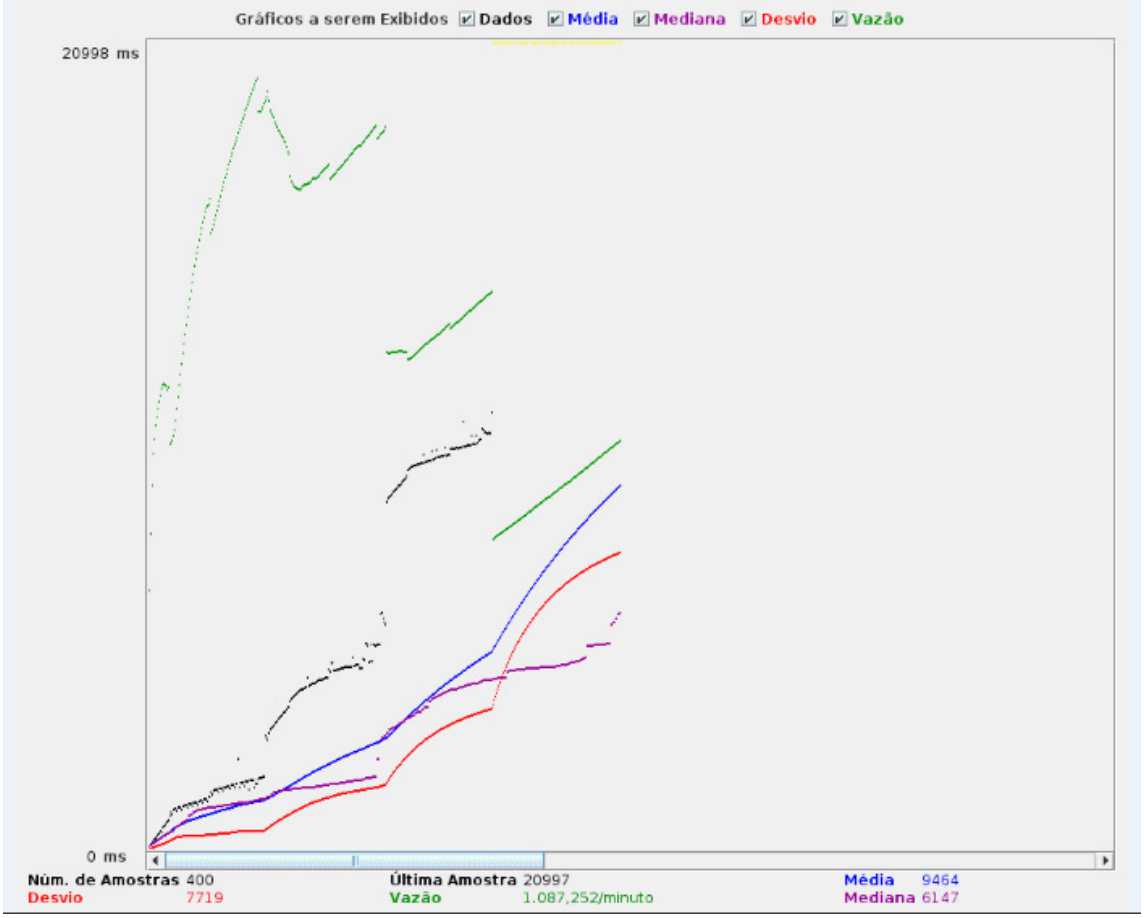

Figura 12 - Gráfico de Resultados 400 usuários.

Após realizado os testes com o grupo de 400 usuários virtuais. Aumentou-se a população amostra para 25000 usuários virtuais.

4.1.2 25000 Usuários

As Figuras 13 e 14 demonstram os resultados através do Relatório de Sumário e o Gráfico de Resultados.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	25000	1873	0	111818	8445,58	93,51%	188,1/sec	246,07	1339,5
TOTAL	25000	1873	0	111818	8445,58	93,51%	188,1/sec	246,07	1339,5

Figura 13 - Relatório de Sumário 25000 usuários.

Pode-se perceber uma vazão de 188,1/segundos por requisição, uma porção bem grande em relação à população amostra baixa (400 usuários). Também teve um aumento significativo na porcentagem de erro no atendimento das requisições que foi aproximadamente 67%.

No Gráfico de Resultados Figura 14, pôde-se perceber uma alta vazão, porém seguido das linhas de desvios de requisição e a média dos dados.

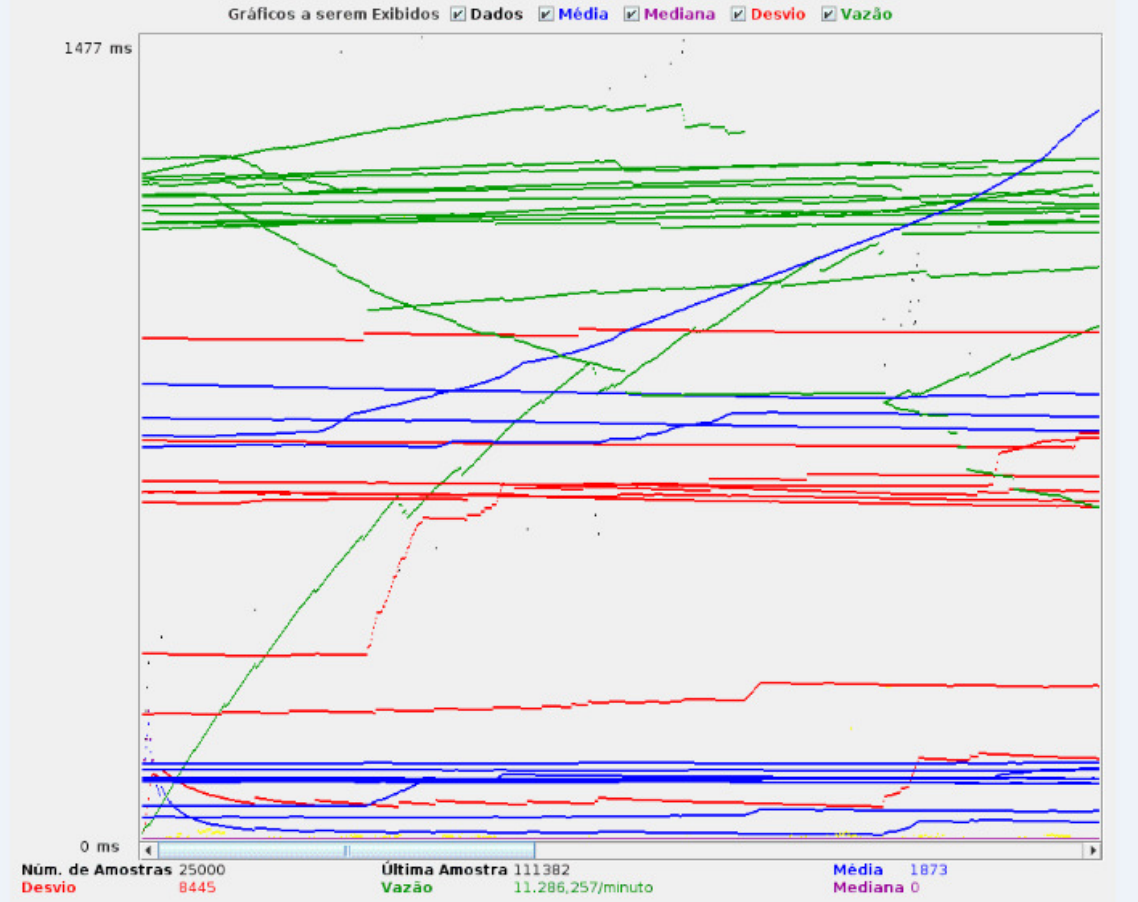

Figura 14 - Gráfico de Resultados 25000 usuários.

Após realizados os testes com um grupo de 25000 usuários, utilizou-se os 50000 usuários virtuais (população amostra alta).

4.1.3 50000 Usuários

As Figuras 15 e 16 demonstram os resultados através do Relatório de Sumário e o Gráfico de Resultados.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	50000	1127	0	69277	4642,63	96,10%	719,6/sec	928,84	1321,8
TOTAL	50000	1127	0	69277	4642,63	96,10%	719,6/sec	928,84	1321,8

Figura 15 - Relatório de Sumário 50000 usuários.

No Relatório de sumário pôde-se perceber um aumento considerável de vazão em relação à população amostra média (25000 usuários), pois na amostra média a disponibilidade da vazão é de 188,1 segundos, e para população amostra alta (50000 usuários) esse número subiu para 719,6 segundos, verificou-se que o numero de usuários virtuais dobrou e a vazão praticamente quadriplicou. Porém ouve um aumento de 3% na porcentagem de erros no atendimento das requisições.

A Figura 16 demonstra os resultados do Gráfico de Resultados.

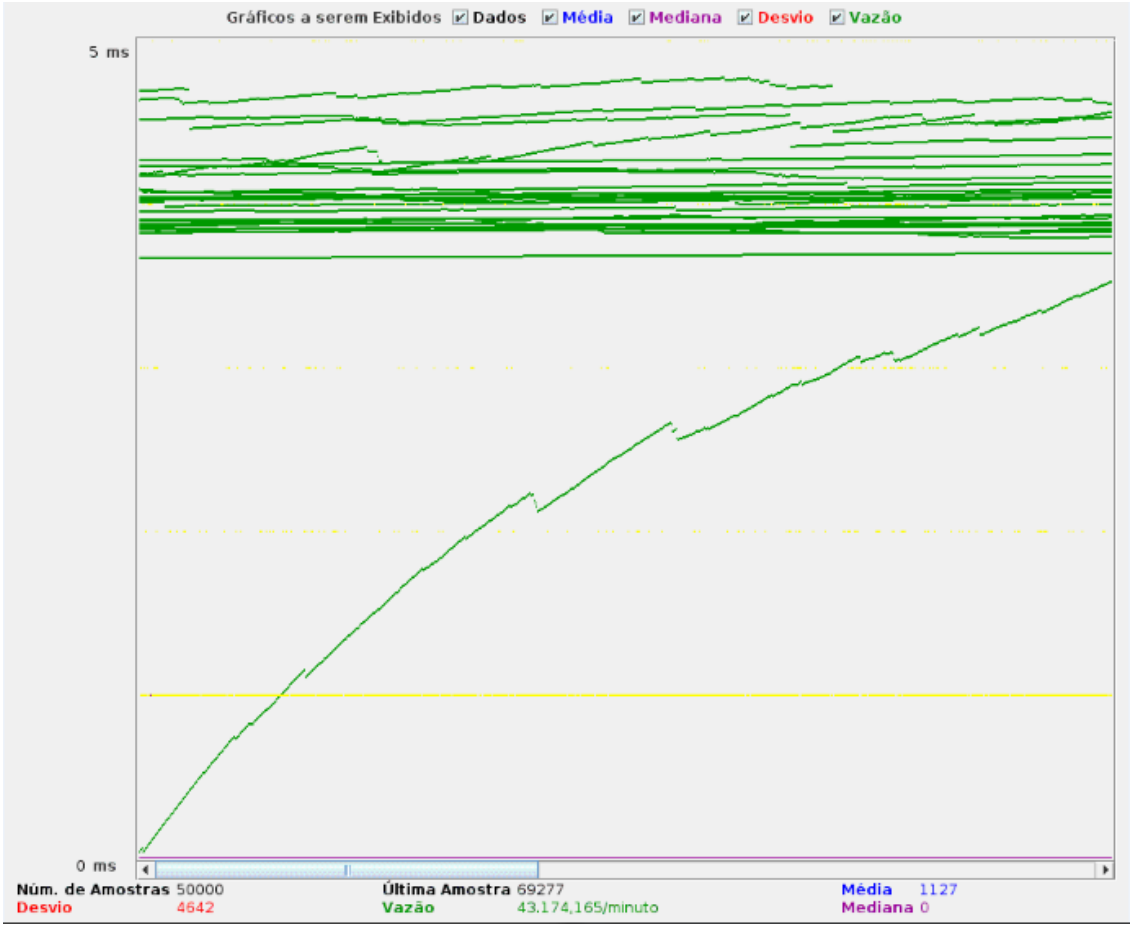

Figura 16 - Gráfico de Resultados 50000 usuários.

No Gráfico de Resultados (Figura 16) pode se perceber como a linha de vazão teve um crescimento significativo e se manteve disponível na maior parte do tempo.

4.2 SEGUNDO TESTE

No segundo teste utilizou-se o conceito de **Escalabilidade Vertical**, e foram adicionados mais recursos ao primeiro servidor.

Aumentou-se a capacidade de memória RAM do primeiro servidor de 2GB para 4GB de memória. Após isso realizou-se novamente os testes.

4.2.1 400 Usuários

As Figuras 17 e 18 demonstram os resultados através do Relatório de Sumário e o Gráfico de Resultados do teste com o grupo de 400 usuários virtuais.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	400	7680	190	21005	6035,23	12,50%	17,8/sec	33,98	1955,6
TOTAL	400	7680	190	21005	6035,23	12,50%	17,8/sec	33,98	1955,6

Figura 17 - Relatório de Sumário 400 usuários.

Pode-se perceber em relação ao primeiro teste que a porcentagem de erro diminuiu de 27,25% para 12,50%. A vazão apresenta-se um pouco abaixo comparada ao primeiro teste que era de 18,1 segundos baixou para 17,8 segundos. Isso pode ser justificar se levado em consideração a oscilação da rede da Intranet a qual foram executados os testes.

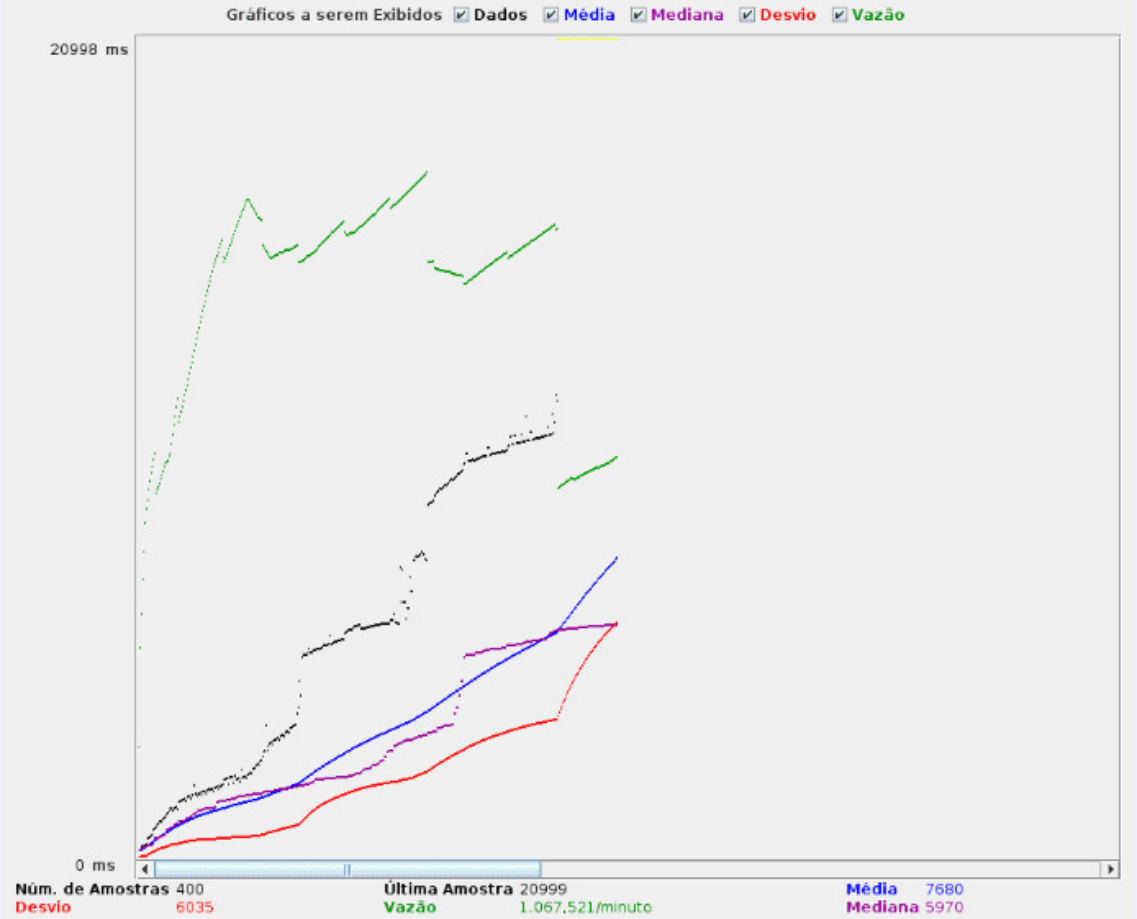

Figura 18 - Gráfico de Resultados 400 usuários.

Também pôde-se observar no Gráfico de Resultados (Figura 18) a disponibilidade na vazão que representou um resultado de 1.067.521/minutos. Comparado ao primeiro teste com 400 usuários obteve uma vazão um pouco abaixo.

4.2.2 25000 Usuários

As Figuras 19 e 20 demonstram os resultados através do Relatório de Sumário e o Gráfico de Resultados.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de By...
Requisição HTTP	25000	1683	0	114598	7969,61	94,14%	217,4/sec	283,88	1337,1
TOTAL	25000	1683	0	114598	7969,61	94,14%	217,4/sec	283,88	1337,1

Figura 19 - Relatório de Sumário 25000 usuários.

Pode-se perceber na vazão o resultado de 217,4 segundos de atendimento das requisições, um aumento de 29,3 segundos na vazão comparado ao primeiro teste que foi de 188,1 segundos.

O Gráfico de Resultados (Figura 20) esboçou os seguintes resultados.

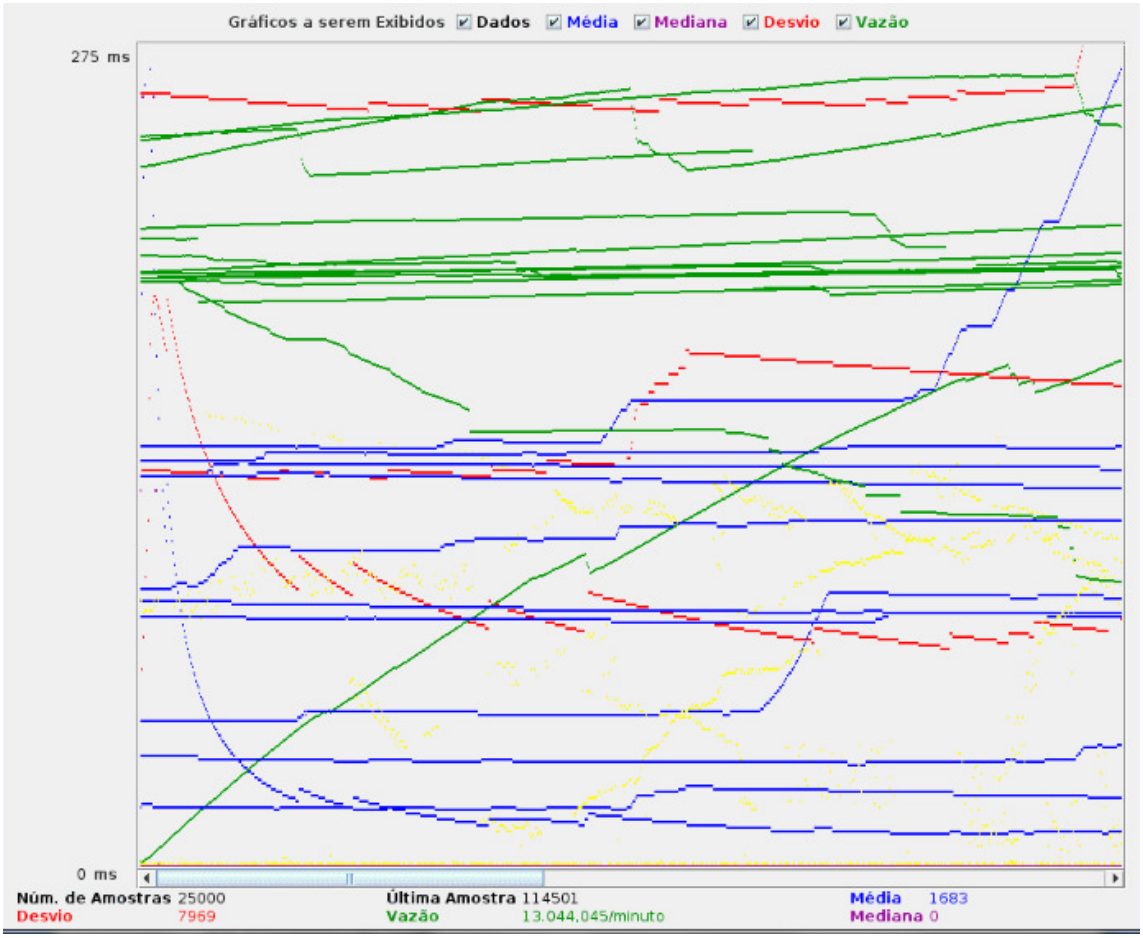

Figura 20 - Gráfico de Resultados 25000 usuários.

Na Figura 20 é possível perceber um aumento na vazão no teste com a população de amostra média (25000 usuários) que resultou em 13.044.045/minutos. Comparada ao primeiro teste esse número era de 11.286.237/minutos. Porém, também pode se perceber várias linhas de desvio (linhas vermelhas) no gráfico e linhas da média (linhas azuis).

4.2.3 50000 Usuários

A Figura 21 demonstra o resultado de 50000 usuários virtuais no Relatório de Sumário.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de By...
Requisição HTTP	50000	789	0	57079	4058,45	97,38%	875,5/sec	1120,59	1310,7
TOTAL	50000	789	0	57079	4058,45	97,38%	875,5/sec	1120,59	1310,7

Figura 21 - Relatório de Sumário 50000 usuários.

Pode-se perceber um aumento significativo da vazão que foi de 875,5 segundos nesse teste. Comparado ao primeiro teste que foi de 719,6 segundos, ouve um aumento de 155,9 segundos no atendimento das requisições com o mesmo número de usuários virtuais. Isso pode ser justificado novamente pela oscilação do tráfego de dados na rede Intranet onde foram executados os testes.

A Figura 22 demonstra o Gráfico de Resultados, onde se pode perceber que a aplicação se manteve disponível a maior parte do tempo.

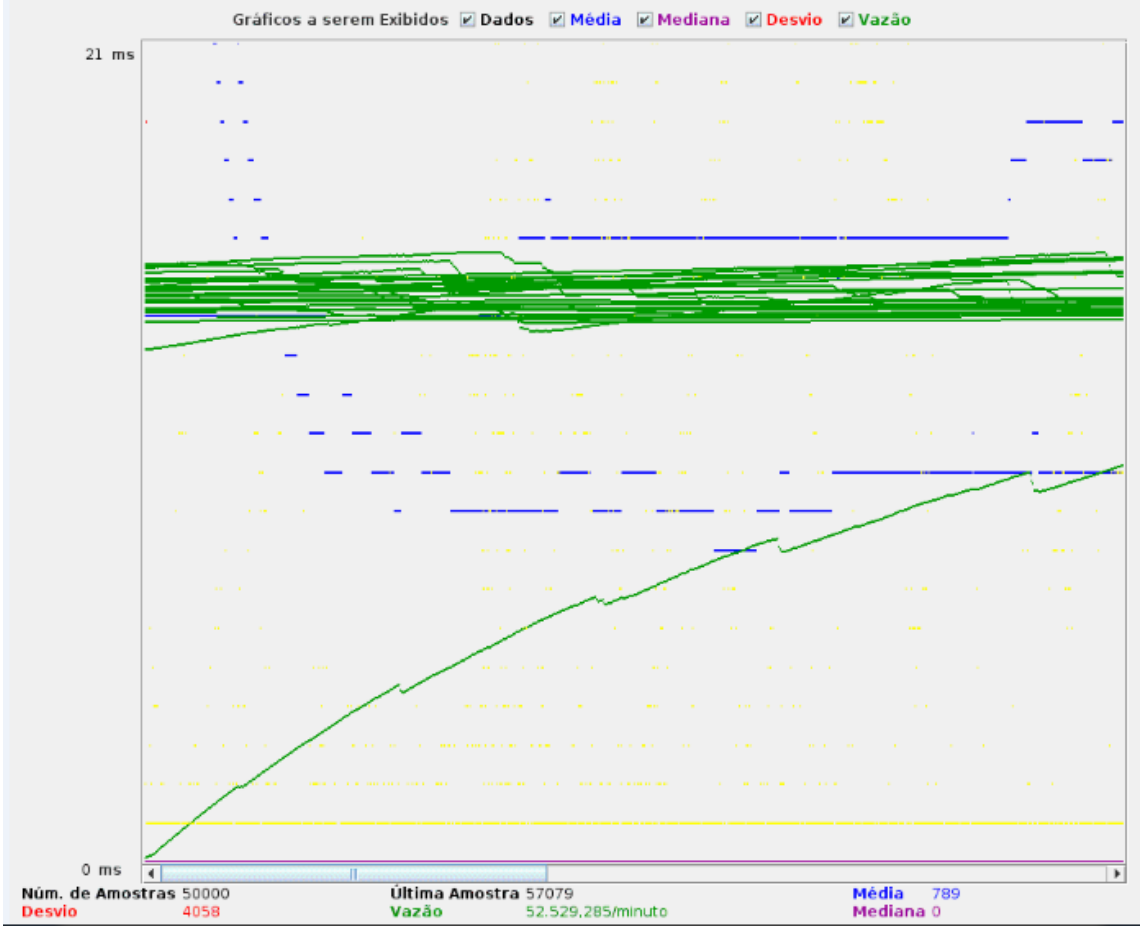

Figura 22 - Gráfico de Resultados 50000 usuários.

Nos testes realizados no primeiro servidor, pôde-se notar que a **Escalabilidade Vertical** embora pouco significativa (nesse caso), devido aos baixos recursos disponíveis para elaborar os testes, conseguiu demonstrar as diferenças positivas em relação ao servidor com a configuração básica inicial.

4.3 TERCEIRO TESTE

No terceiro teste utilizou-se o conceito de Clusterização. Para isso, necessariamente o primeiro servidor foi configurado com:

- Módulo Apache *Proxy Balancer*: Esse módulo utiliza o serviço de *mod_proxy* do Apache. Ele fornece suporte de balanceamento de carga para HTTP, FTP e protocolos apj13. (Apache Module, 2011);

- *Mongrel* e *Mongrel Cluster*: *Mongrel* utilizado como servidor de aplicação, e *Mongrel Cluster* (*gem* ou biblioteca do *Ruby*) utilizado para servir múltiplos servidores. (Kyusik, 2009).

- *Phusion Passenger*: *Gem* projetada para questões de performance, estabilidade e segurança, necessária para utilizar em conjunto com servidores Apache, pois facilita o *deployment* de aplicações *Web* escritas em *Ruby*. (Phusion, 2010).

A Figura 23 demonstra como foi feita a primeira etapa de configuração para Clusterização no *httpd.conf* do Apache.

- Na linha 1 demonstra a configuração do Apache *Proxy Balancer*, com o *mongrel_cluster*;

- Das linhas 2 até 9 configurou-se o IP do primeiro servidor e as portas em que cada *Cluster* iria executar.

```
1  <Proxy balancer://mongrel_cluster>
2      BalancerMember http://192.168.1.193:3000
3      BalancerMember http://192.168.1.193:3001
4      BalancerMember http://192.168.1.193:3002
5      BalancerMember http://192.168.1.193:3003
6      BalancerMember http://192.168.1.193:3004
7      BalancerMember http://192.168.1.193:3005
8      BalancerMember http://192.168.1.193:3006
9      BalancerMember http://192.168.1.193:3007
10 </Proxy>
```

Figura 23 - Configuração do Apache Proxy Balancer para Clusterização.

A Figura 24 demonstra a continuação das configurações.

```
12 <VirtualHost *:80>
13     DocumentRoot /var/www/html/meutwitterajax/public
14
15     LoadModule passenger_module /usr/local/rvm/gems/ruby-1.8.7-p352/gems/passenger-3.0.9/ext/apache2/mod_passenger.so
16     PassengerRoot /usr/local/rvm/gems/ruby-1.8.7-p352/gems/passenger-3.0.9
17     PassengerRuby /usr/local/rvm/wrappers/ruby-1.8.7-p352/ruby
18
19       <Directory /var/www/html/meutwitterajax/public>
20          Options FollowSymLinks
21          AllowOverride all
22          Options -MultiViews
23       </Directory>
24
25         RewriteEngine On
26         RewriteCond %{HTTP_HOST} ^regis.com$ [NC]
27         RewriteRule ^(.*)$ http://regis.com$1 [R=301,L]
28
29         RewriteCond %{DOCUMENT_ROOT}/system/maintenance.html -f
30         RewriteCond %{SCRIPT_FILENAME} !maintenance.html
31         RewriteRule ^.*$ /system/maintenance.html [L]
32
33         RewriteRule ^/$ /index.html [QSA]
34
35         RewriteRule ^([^.]+)$ $1.html [QSA]
36
37         RewriteCond %{DOCUMENT_ROOT}/%{REQUEST_FILENAME} !-f
38         RewriteRule ^/(.*)$ balancer://mongrel_cluster%{REQUEST_URI} [P,QSA,L]
39
40         ErrorLog  /tmp/meutwitterajax/log/error.log
41         CustomLog /tmp/meutwitterajax/access.log combined
42 </VirtualHost>
```

Figura 24 - Configuração do Host Virtual da Aplicação.

- A linha 12 demonstra abertura de um *host* virtual que será acessado na porta 80;
- A linha 13 demonstra onde está o *path* da aplicação de *microblogging*;
- A linha 15 demonstra que está sendo carregado o *passenger_module* (*Phusion Passenger*);
- A linha 16 demonstra o *path* da *Gem Passenger*;
- A linha 17 demonstra o *path* do *Ruby*;
- A linha 19 demonstra a configuração do diretório público da aplicação de *microblogging*;
- As linhas 20, 21 e 22 são padrões da configuração do ".htaccess" (arquivo de configuração distribuída) do Apache;
- As linhas 25, 26 e 27 são padrões do *mod_rewrite* do Apache, que utiliza regras baseadas em reescrita (analisador de expressões regulares) para reescrever URLs socilitadas;

- As linhas 29, 30 e 31 verificam arquivos de configuração e redirecionam as solicitações;
- Na linha 33 o *Rewrite* verifica se há uma página índex estática;
- Na linha 35 o *Rewrite* verifica páginas em cache do *Rails*;
- Nas linhas 37 *Rewrite* verifica todas as solicitações não estáticas e as agrupa;
- A linha 38 demonstra a configuração das requisições para os servidores de aplicação do *Proxy Balancer*;
- As linhas 40 e 41 demonstram os caminhos de logs gerados para erros e acessos.

A arquitetura do primeiro servidor ficou semelhante à Figura 25.

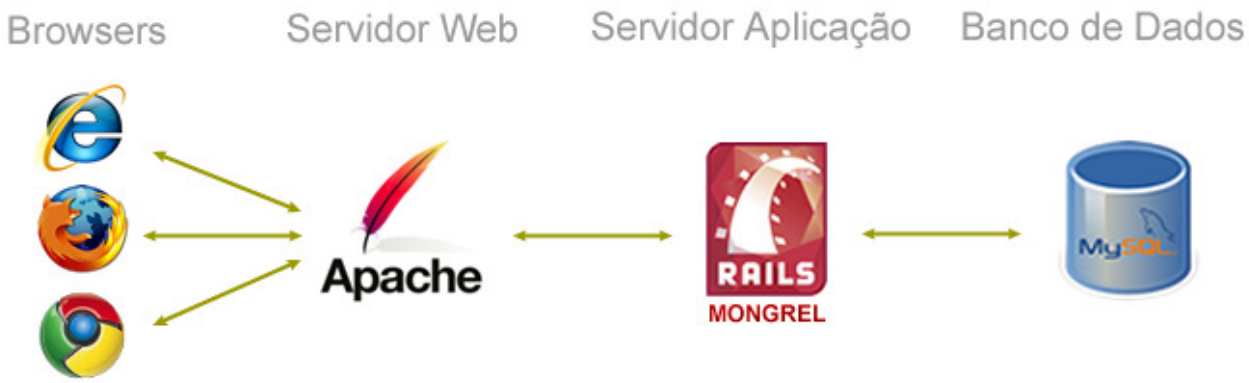

Figura 25 - Arquitetura Primeiro Servidor para Clusterização.

Depois das configurações de clusterização no primeiro servidor, deu-se inicio aos testes.

4.3.1 400 Usuários

Os testes com 400 usuários virtuais no Relatório de Sumário (Figura 26) demonstraram um aumento significativo na vazão, comparado ao segundo teste que resultou 17,8 segundos, obteve-se um aumento de 29,9 segundos.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	400	4118	193	7324	1955,83	0,00%	47,7/sec	103,46	2220,0
TOTAL	400	4118	193	7324	1955,83	0,00%	47,7/sec	103,46	2220,0

Figura 26 - Relatório de Sumário 400 usuários.

O Gráfico de Resultados pode ser observado na Figura 27.

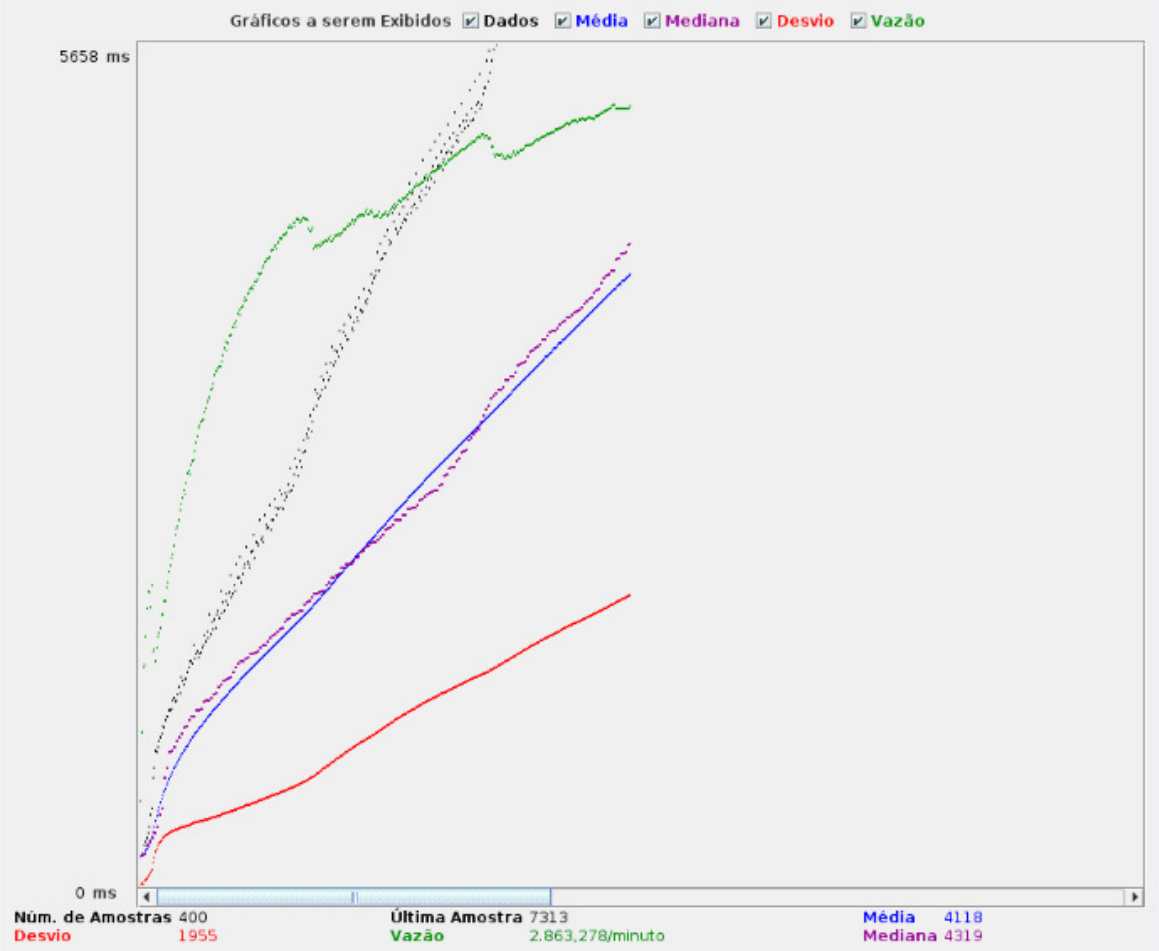

Figura 27 - Gráfico de Resultados 400 usuários.

Na Figura 27, observou-se um aumento significativo da vazão que foi de 1.067.521/minutos no segundo teste para 2.863.278/minutos no terceiro teste com o mesmo número de usuários virtuais.

4.3.2 25000 Usuários

Os testes com 25000 usuários puderam ser observados através do Relatório de Sumário na Figura 28.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	25000	922	0	31271	3905,17	94,04%	791,5/sec	1034,89	1338,9
TOTAL	25000	922	0	31271	3905,17	94,04%	791,5/sec	1034,89	1338,9

Figura 28 - Relatório de Sumário 25000 usuários.

Pode-se perceber em relação ao segundo teste, um aumento significativo da vazão que foi de 217,4 segundos (segundo teste) para 791,5 segundos no terceiro teste, um aumento de 574,1 segundos na vazão. Também observou-se pouca diferença na porcentagem de erro das requisições que foi de 94,14% no segundo teste para 94,4% no terceiro teste.

O Gráfico de Resultados (Figura 29) pode demonstrar os seguintes resultados.

Figura 29 - Gráfico de Resultados 25000 usuários.

Pode-se perceber que o Gráfico de Resultados (Figura 29) do terceiro teste apresenta a vazão disponibilizada de uma melhor forma comparada ao Gráfico de Resultados do segundo teste, pois além de não conter linhas de desvio, teve aumento significativo no seu resultado onde no segundo teste com o mesmo número de usuários teve 13.044.045/minutos de resultado na vazão, nesse teste a vazão subiu para 47.489.394/minutos.

4.3.3 50000 Usuários

Para 50000 usuários o Relatório de Sumário (Figura 30) trouxe uma vazão um pouco abaixo comparada ao segundo teste que foi de 875,5 segundos, uma diminuição de 66 segundos comparada ao terceiro teste que resultou em 809,5. Isso pode ser justificado pela oscilação da rede, a qual foi executada os testes.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	50000	1001	0	61500	4682,18	94,11%	809,5/sec	1057,53	1337,8
TOTAL	50000	1001	0	61500	4682,18	94,11%	809,5/sec	1057,53	1337,8

Figura 30 - Relatório de Sumário 50000 usuários.

A Figura 31 ilustra o Gráfico de Resultados.

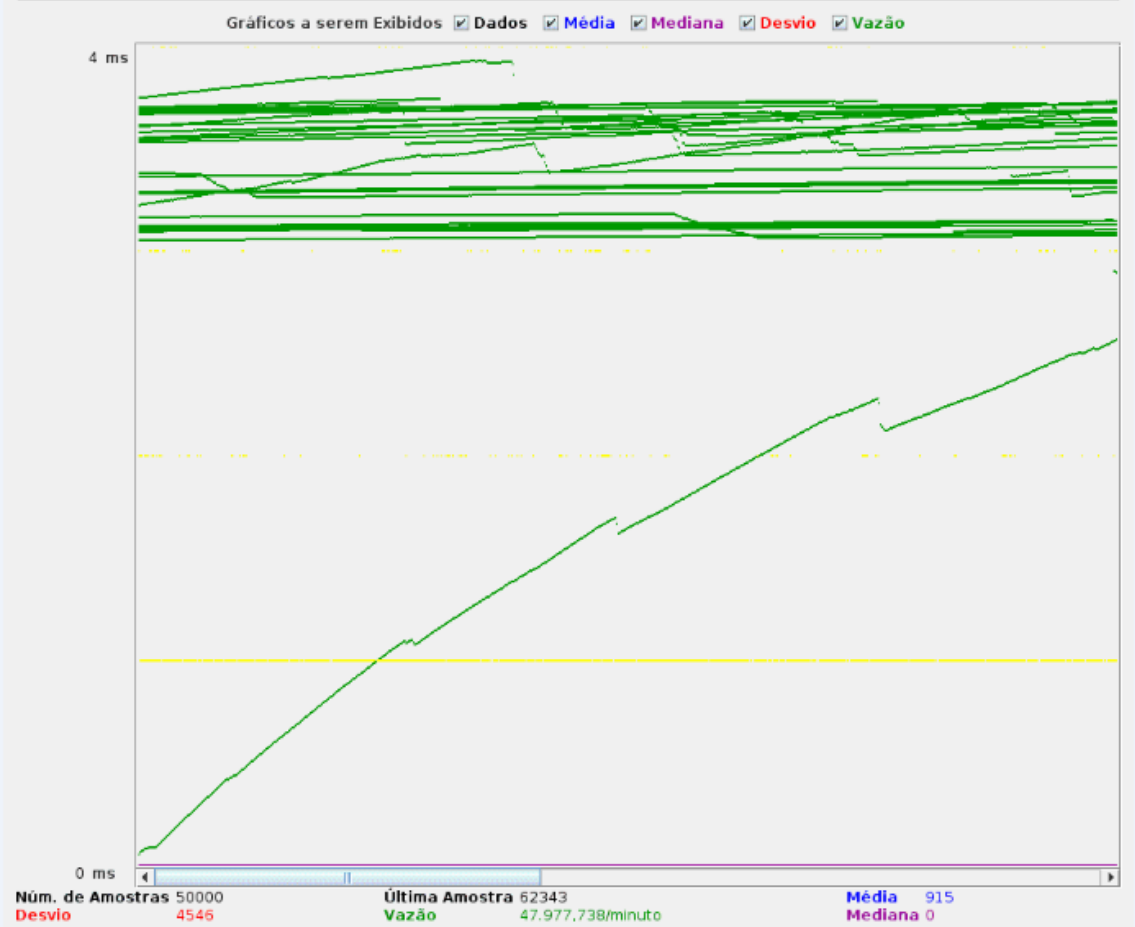

Figura 31 - Gráfico de Resultados 50000 usuários.

Na Figura 31, pode-se perceber uma disponibilidade da vazão abaixo comparada ao segundo teste que foi de 52.529.285/minutos com a mesma proporção de usuários, em relação ao terceiro teste, esse resultou em 47.977.738/minutos a vazão. Esse resultado pode ser justificado pela oscilação da rede de Intranet a qual foram executados os testes.

4.4 QUARTO TESTE

No quarto teste utilizou-se o conceito de **Escalabilidade Horizontal**. Para isso, necessariamente o segundo servidor teve a mesma configuração de Clusterização aplicada no primeiro servidor. Porém a Figura 32 mostra a diferença na configuração no *httpd.conf* do Apache:

- Nas linhas 1 e 2 definiu-se o IP de cada servidor apenas.

```
1 <Proxy balancer://mongrel_cluster>
2         BalancerMember http://192.168.1.193:3000
3         BalancerMember http://192.168.1.180:3000
4 </Proxy>
```

Figura 32 - Configuração do Apache Proxy Balancer para escalar Horizontal.

Após as configurações da arquitetura para **escalabilidade horizontal**, foram realizados os testes.

4.4.1 400 Usuários

Os testes de 400 usuários no Relatório de Sumário trouxeram uma vazão maior comparada ao terceiro teste onde subiu de 47,7 (terceiro teste) para 60,3 no

quarto teste, um aumento de 12,6 segundos no atendimento as requisições. A Figura 33 demonstra os dados.

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	400	2867	63	5446	1618,32	0,00%	60,3/sec	143,30	2433,0
TOTAL	400	2867	63	5446	1618,32	0,00%	60,3/sec	143,30	2433,0

Figura 33 - Relatório de Sumário 400 usuários.

A Figura 34 demonstra os dados no Gráfico de Resultados.

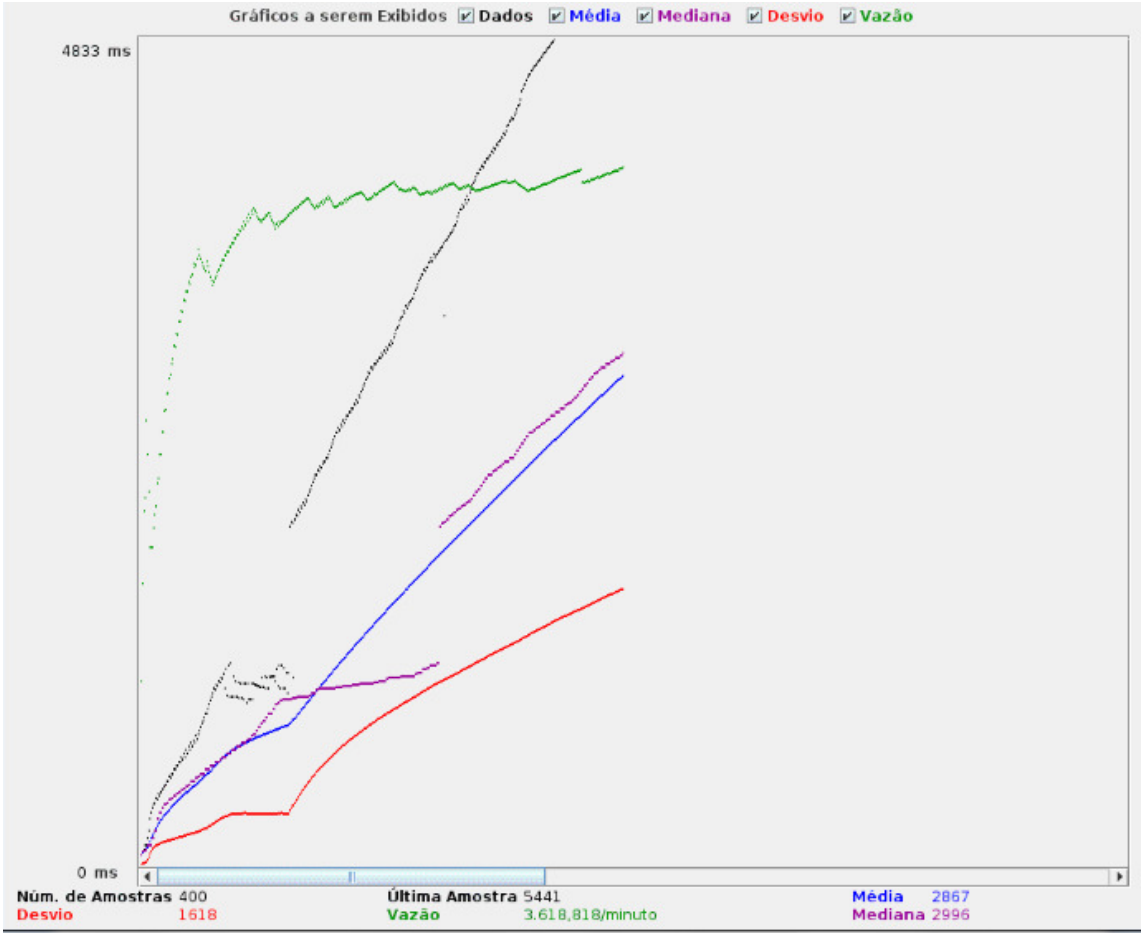

Figura 34 - Gráfico de Resultados 400 usuários.

Pode se perceber uma alta disponibilidade dos dados, onde esses representados no Gráfico de Resultados (Figura 34) estão acima da linha de vazão. Também observou-se um aumento na vazão comparado ao terceiro teste que subiu de 2.863.278/minutos para 3.618.818/minutos.

4.4.2 25000 Usuários

O teste com 25000 usuários no Relatório de Sumário (Figura 35) demonstra os seguintes resultados:

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	25000	862	0	30595	3533,36	93,28%	804,0/sec	1068,71	1361,2
TOTAL	25000	862	0	30595	3533,36	93,28%	804,0/sec	1068,71	1361,2

Figura 35 - Relatório de Sumário 25000 usuários.

Pode-se perceber que comparado ao terceiro teste com o mesmo número de usuários virtuais a vazão subiu de 791,5 segundos (terceiro teste) para 804,0 segundos, um aumento de 12,5 segundos na vazão.

O Gráfico de Resultados (Figura 36) demonstra os seguintes resultados com os testes de 25000 usuários.

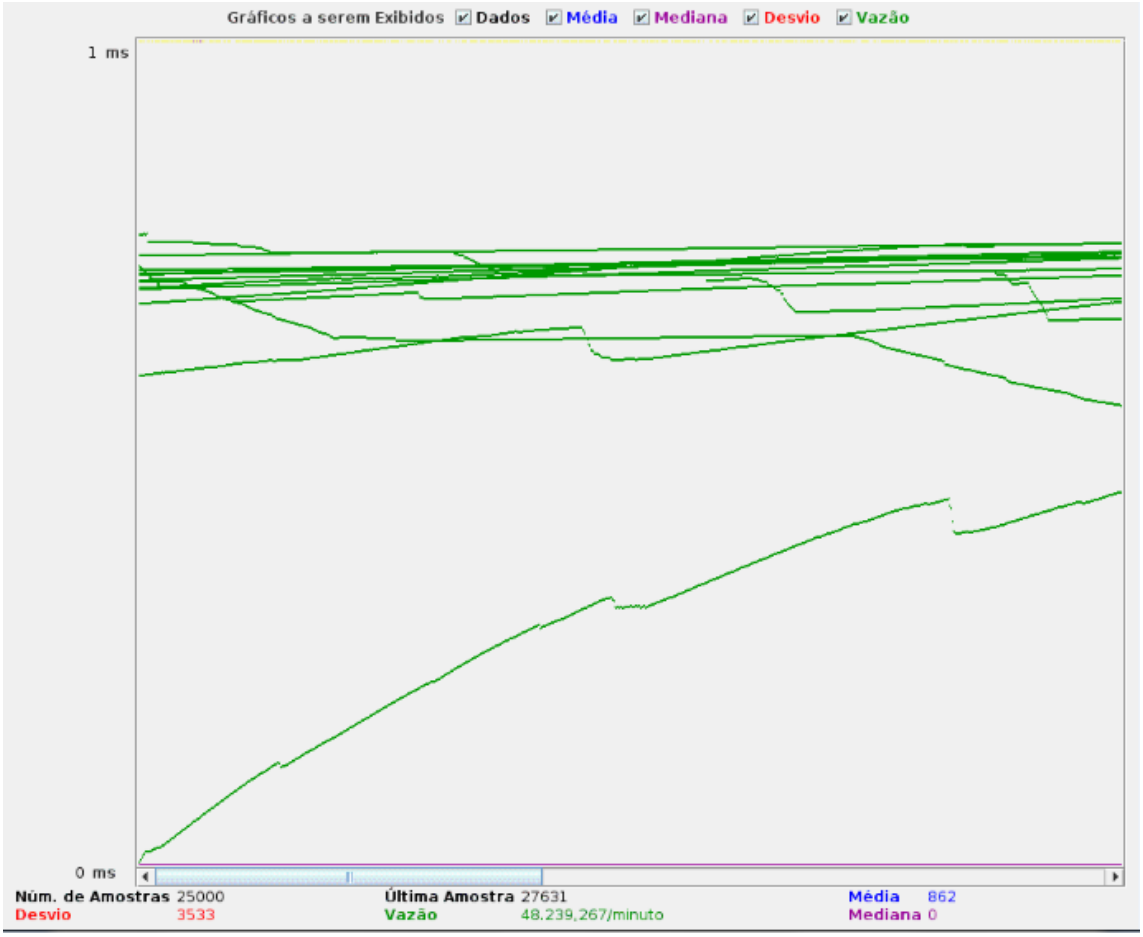

Figura 36 - Gráfico de Resultados 25000 usuários.

No Gráfico de Resultados (Figura 36), pode se perceber uma vazão um pouco acima do terceiro teste que foi de 47.489.394/minutos para 48.239.267/minutos realizado com a mesma proporção de usuários.

4.4.3 50000 Usuários

Para 50000 usuários o teste no Relatório de Sumário obteve os seguintes resultados (Figura 37).

Rótulo	# Amostras	Média	Mín.	Máx.	Desvio Padrão	% de Erro	Vazão	KB/s	Média de Byt...
Requisição ...	50000	809	0	23917	3519,68	95,46%	1077,0/sec	1396,43	1327,8
TOTAL	50000	809	0	23917	3519,68	95,46%	1077,0/sec	1396,43	1327,8

Figura 37 - Relatório de Sumário 50000 usuários.

Pode-se perceber na Figura 37, que a vazão aumentou em relação ao terceiro teste que foi de 809,5 segundos para 1077,0 segundos, um aumento de 267,5 segundos no atendimento das requisições.

Para o Gráfico de Resultados a Figura 38 demonstra os resultados obtidos.

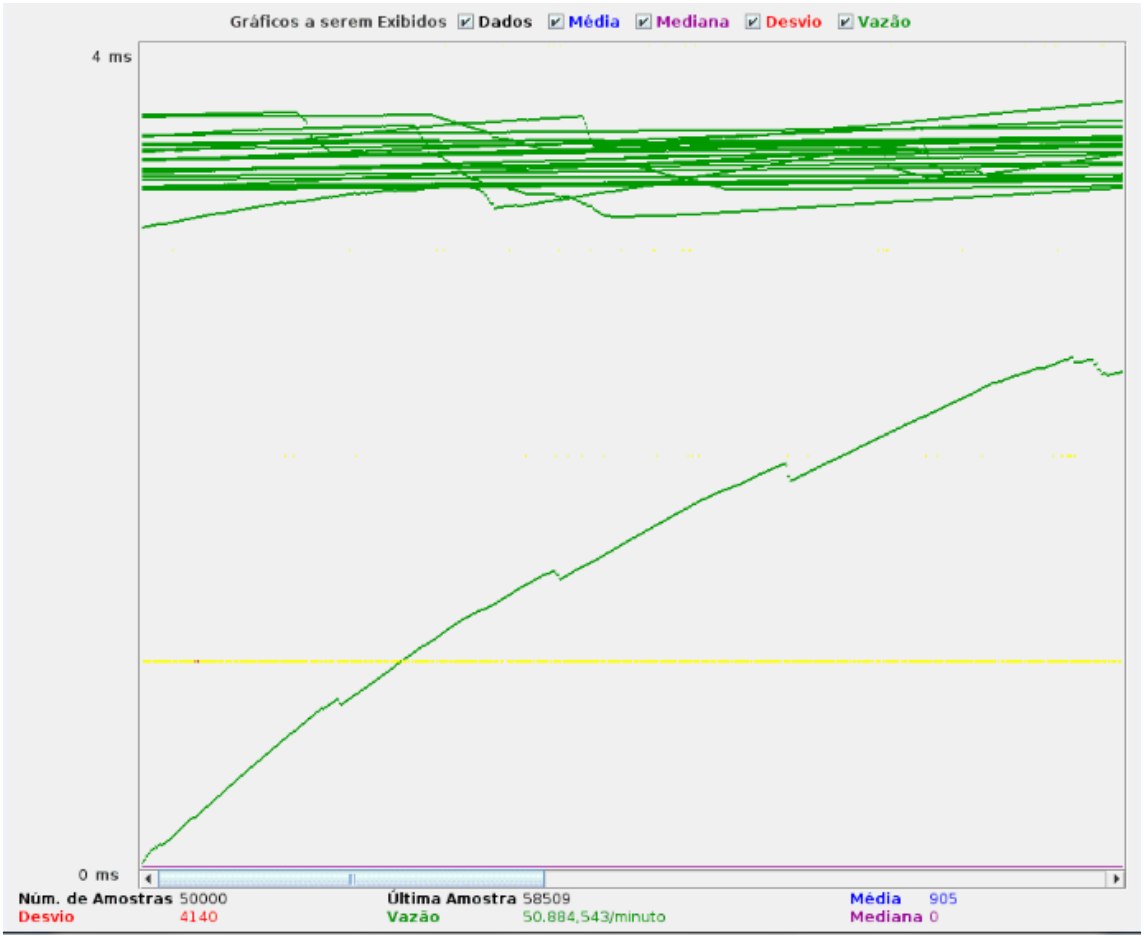

Figura 38 - Gráfico de Resultados 50000 usuários.

Pode-se perceber uma alta vazão no Gráfico de Resultados (Figura 38) comparada ao terceiro teste com a mesma proporção de usuários que foi de 47.977.738/minutos para 50.884.543/minutos.

4.5 DESEMPENHO NA APLICAÇÃO

Uma aplicação *Web* pode ser escalável e não ter um bom desempenho, o inverso pode ocorrer também. Mas faz-se necessário que uma aplicação escalável tenha um bom desempenho, pois aumenta a satisfação dos usuários do sistema. (Henderson, 2006). Para isso, existem funcionalidades que podem ser configuradas no servidor Apache que aumentam consideravelmente o desempenho da aplicação.

A Figura 39 demonstra a configuração utilizada no *httpd.conf* no Apache:

```
1  <IfModule mod_deflate.c>
2      SetOutputFilter DEFLATE
3      SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png|rar|zip|pdf)$ no-gzip dont-vary
4      <IfModule mod_headers.c>
5          Header append Vary User-Agent
6      </IfModule>
7  </IfModule>
```

Figura 39 - Configuração do Módulo Deflate.

- A linha 1 demonstra uma condição para verificar se o módulo "*Deflate*" está habilitado no Apache, esse módulo permite criar um filtro que comprime a saída do conteúdo de uma página *Web*;
- A linha 2 demonstra que é adicionado a compressão de saída implementada pelo filtro "*Deflate*";
- A linha 3 demonstra a compactação nas extensões de arquivo;
- A linha 4 verifica se o módulo *mod_headers* está habilitado, esse módulo oferece diretrizes para controlar e modificar solicitações HTTP e cabeçalhos de resposta;

- A linha 5 demonstra a configuração do cabeçalho de resposta HTTP que é enviado como respostas para as requisições dos clientes.
- As linhas 6 e 7 fecham os módulos.

Para testar as modificações feitas nas configurações no Apache, utilizou-se o *plugin Page Speed* do navegador Google Chrome. A Figura 40 demonstra o resultado do teste na página principal da aplicação antes de ativar a compressão dos arquivos.

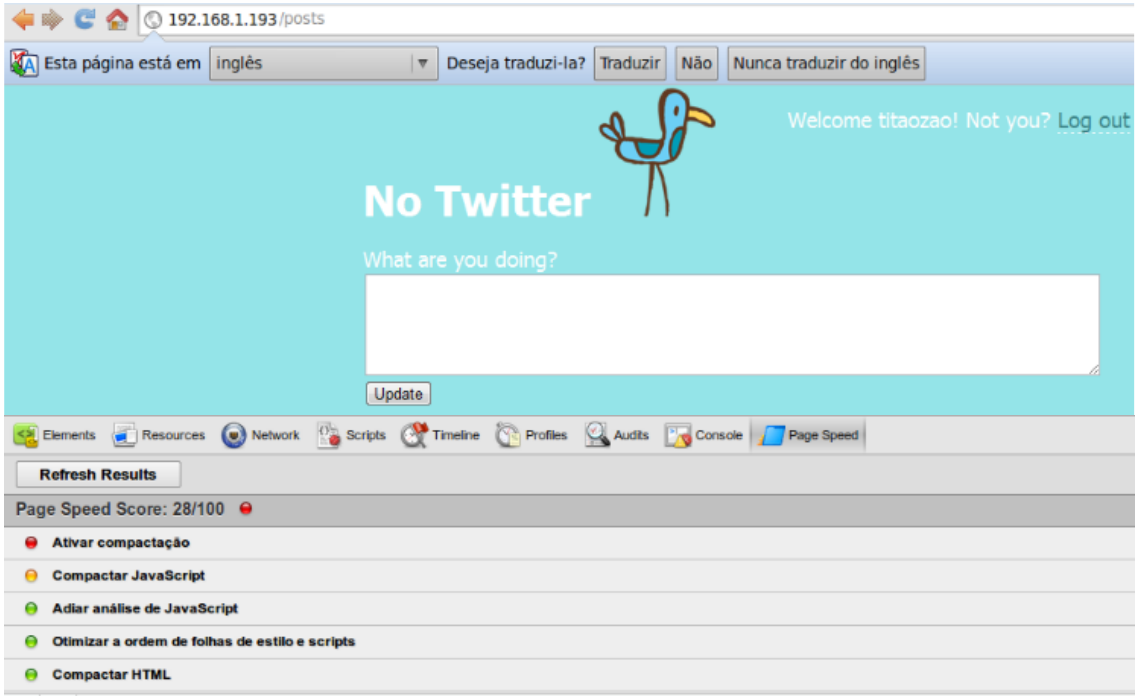

Figura 40 - Teste de desempenho na aplicação antes da configuração Deflate.

É possível perceber que se destaca em vermelho e amarelo os itens de Ativar compactação e Compactar *JavaScript*.

Após feita as configurações dos recursos de compressão com *Deflate*, também utilizou-se o *Closure Compiler* (disponível em *http://closure-compiler.appspot.com/home*.) que é uma aplicação on-line responsável por compactar *JavaScripts* utilizados em aplicação *Web*. A Figura 41 pode demonstrar o resultado após a utilização dos recursos de desempenho.

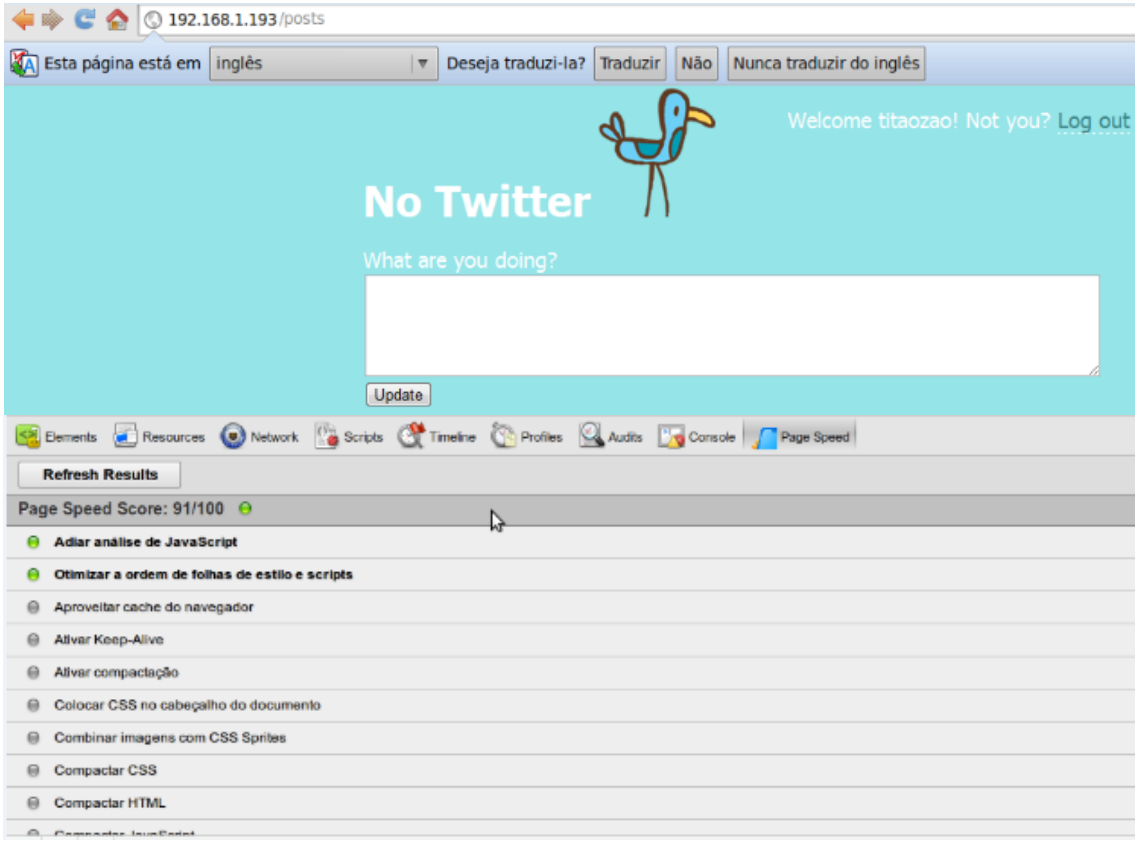

Figura 41 - Teste de desempenho na aplicação depois da configuração Deflate.

Pode-se perceber um aumento significativo realizando o teste do *Page Speed* após a configuração dos recursos de compreesão com *Deflate* e compactação dos *JavaScripts*, ou seja, ouve aumento significativo no desempenho da aplicação.

5 CONSIDERAÇÕES FINAIS

Este capítulo trata das conclusões finais que foram abstraídas durante o desenvolvimento desse trabalho e também sugere idéias para quem desejar dar continuidade ao trabalho.

5.1 CONCLUSÃO

Para desenvolver aplicações *Web* existem várias linguagens e *frameworks* disponíveis que podem ser utilizados. Porém projetar uma arquitetura escalável para essas aplicações pode depender de vários fatores.

Não se pode afirmar que existe uma receita para escalabilidade em aplicações *Web*, assim como para o desenvolvimento de *software*, pois cada caso é um caso. Deve-se analisar e identificar onde se pode melhorar, quais recursos utilizar para suprir a necessidade de atender uma demanda crescente de usuários acessando simultaneamente uma aplicação *on-line*.

Esse trabalho demonstrou alguns dos vários recursos que podem ser implementados em arquiteturas de servidor ou nos projetos de aplicações *Web*, visando à alta disponibilidade dessas aplicações para os usuários. Também abordou questões sobre o desempenho dessas aplicações, pois, aliado à escalabilidade uma aplicação deve ter um bom desempenho.

Nesse trabalho notou-se que, conforme foram utilizados os recursos para a escalabilidade na aplicação *Web*, aumentou-se a disponibilidade das informações. Pode-se perceber que, quando aplicada à escalabilidade vertical, essa teve resultados significativos em relação às configurações originais do servidor. O que também pôde ser observado quando utilizado o conceito de clusterização no mesmo servidor. Outra aplicabilidade observada também foi a escalabilidade horizontal, que demonstrou a vantagem de distribuir ou balancear cargas de trabalho entre os servidores da arquitetura de rede.

Pode-se concluir, portanto, que combinando os recursos descritos nesse trabalho, é possível obter escalabilidade e alta disponibilidade em uma aplicação *Web*.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

O presente trabalho demonstrou algumas técnicas de como escalar uma aplicação *Web* e também apresentou os resultados de cada técnica utilizada em dados numéricos e gráficos. Para os recursos utilizados, poderiam ser explorados técnicas para outras linguagens de programação e *frameworks*, ou até mesmo a possibilidade de escalar o banco de dados MySQL. Por outro lado, poderia-se explorar mais sobre os resultados obtidos nos testes, principalmente com as populações de amostra média e alta, que demonstraram seus resultados com uma porcentagem de erro elevado em comparação a população de amostra baixa. Além disso, esses resultados poderiam ser discutidos estatisticamente.

6 REFERENCIAS BIBLIOGRÁFICAS

ALECRIM, Emerson**. Cluster: principais conceitos**. 2004. Disponível em: <http://www.infowester.com/cluster.php>. Acesso em 23 de agosto de 2011, às 13:00.

ALMEIDA, Ricardo. **Escalabilidade != Performance.** 2008. Disponível em: <http://manifestonaweb.wordpress.com/2008/06/18/escalabilidade-performance/>. Acesso em: 02 de agosto de 2011 às 19:43.

APPLICATION SERVERS. **Servidores de Aplicações**. 2003. Disponível em:<http://www.iweb.com.br/iweb/pdfs/20031008-appservers-01.pdf>. Acesso em: 24 de agosto de 2011, às 21:51.

BUYENS, Jim. **Aprendendo MySQL e PHP**. 2002. São Paulo: Makron Books Ltda. Tradução: Lavio Pareschi, Revisão Técnica: Odemir Bruno. ISBN: 85-346-1312-5.

CAMPOS, Fábio Martinho. **Testes de Performance – Testes de Carga, Stress e Virtualização – Parte 3.** 2011. Disponível em: <http://www.testexpert.com.br/?q=node/2005>. Acesso em: 01 de dezembro de 2011, às 00:13.

FAUNA. **Mongrel**. 2008. Disponível em: <https://github.com/fauna/mongrel/>. Acesso em 22 de agosto de 2011, às 23:57.

FERREIRA, Edmar. **Escolhendo entre escalabilidade horizontal e escalabilidade vertical**. 2010. Disponível em: < http://escalabilidade.com/2010/09/21/escolhendo-entre-escalabilidade-horizontal-e-escalabilidade-vertical/>. Acesso em 18 de agosto de 2011, às 21:15.

FREITAS, Cintia Pereira. **Implementação e comparação de servidores de mapas em ambiente de alta demanda**. 2008. Disponível em: <http://www.feg.unesp.br/ceie/Monografias-Texto/CEIE0803.pdf>. Acesso em 27 de agosto de 2011, às 13:30.

G1, Globo. **Facebook atinge número recorde de acessos nos Estados Unidos**. 2010. Disponível em: <http://g1.globo.com/tecnologia/noticia/2010/07/facebook-atinge-numero-recorde-de-acessos-nos-estados-unidos.html>. Acesso em: 02 de agosto de 2011 às 23:05.

GALVES, Miguel. **Variações sobre o tema Twitter.** 2008. Disponível em: < http://log4dev.com/2008/06/06/variacoes-sobre-o-tema-twitter/>. Acesso em: 31 de agosto de 2011 às 23:16.

GNOME, The Project. **Gedit Text Editor**. 2011. Disponível em:< http://projects.gnome.org/gedit/ > Acesso em 21 de agosto de 2011, às 22:33.

GNU. **Licenças de Software Livre**. 2008. Disponível em: < http://www.gnu.org/licenses/licenses.pt-br.html>. Acesso em 22 de agosto de 2011, às 22:00.

GoGrid, Complex Infrastructure Made Easy™. **Scaling Your Internet Business.** 2010. Disponível em: < https://www.cloudsleuth.net/c/document_library/get_file?uuid=535be2b1-0ff3-47f8-bdb7-650245975a68&groupId=100135>. Acesso em 22 de agosto de 2011, às 12:35.

GOMES, Diego. **O que é escalabilidade?**. 2010. Disponível em: <http://escalabilidade.com/2010/01/31/o-que-e-escalabilidade/>. Acesso em: 31 de agosto de 2011 às 21:34.

HENDERSON, Cal. **Building Scalable Web Sites**. 2006. O'Reilly Media, Inc, 2006, 1ª Edição. ISBN-10: 0-596-10235-6. ISBN-13: 978-0-596-10235-7.

JAKARTA, Apache. **Apache JMeter**. 2011. Disponível em: < http://jakarta.apache.org/jmeter/>. Acesso em 29 de agosto de 2011, às 20:09.

JavaEE: Conceitos Básicos. **Conceitos de Java**. 2010. Disponível em: <http://javasemcafe.blogspot.com/2010/08/aula-04082010-4tads-conceitos-de-java.html>. Acesso em 24 de agosto, às 22:09.

KYUSIK. **Mongrel_cluster**. 2009. Disponível em: <https://github.com/kyusik>. Acesso em: 01 de setembro de 2011, às 20:21.

LANG, Ruby. **About Ruby**. 2011. Disponível em: < http://www.ruby-lang.org/en/about/>. Acesso em 19 de agosto de 2011, às 21:42.

LINGHAM, Vinny. **Top 20 Reasons why Web Apps are Superior to Desktop Apps**. 2007. Disponível em < http://www.vinnylingham.com/top-20-reasons-whyweb-apps-are-superior-to-desktop-apps.html >. Acesso em 20 de agosto de 2011, às 20:45.

MUTO, Claudio Adonai. **PHP & MySQL: guia completo**. 2ª Ed. Rio de Janeiro: Brasport. 2004. ISBN 85-7452-154-X.

MYSQL. **Why MySQL?**. 2011. Disponível em: <http://www.mysql.com/why-mysql/>. Acesso em 22 de agosto de 2011, às 21:36.

NATÁRIO, Rui. **Balanceamento de Carga.** 2011. Disponível em: < http://redes-e-servidores.blogspot.com/2011/03/balanceamento-de-carga-i.html>. Acesso em 21 de agosto de 2011, às 09:54.

NIEDERAUER, Juliano. **Web Interativa com Ajax e PHP.** 2007. São Paulo: Novatec Editora, 2007.

NORONHA, Rina. **A importânica da escalabilidade para o crescimento de sua empresa.** 2010. Disponível em: <http://imasters.com.br/artigo/17099/governanca/a_importancia_da_escalabilidade_para_o_crescimento_da_sua_empresa/>. Acesso em: 03 de agosto de 2011 às 10:55.

OLIVEIRA, Eustáquio Rangel Jr. **Tutorial de Ruby**. 2005. Disponível em: <http://www.eustaquiorangel.com/downloads/tutorialruby.pdf >. Acesso em 13 de setembro de 2011, às 21:01h.

OSI, Open Source Initiative. **Open Source Initiative OSI - The MIT License (MIT):Licensing. The MIT License (MIT).** 2011. Disponível em: <http://www.opensource.org/licenses/mit-license.php>. Acesso em 21 de agosto de 2011, às 13:30.

PHUSION, Passenger. **Phusion Passenger**. 2011. Disponível em: <http://www.modrails.com/>. Acesso em: 02 de setembro de 2011, às 20:31.

RAYMOND, Scott. **Ajax on Rails.** 2006. 1ª Ed. O'Reilly Media, Inc, 2007. Gravenstein Highway North, Sebastopol, CA 95472. ISBN-10: 0596527446. ISBN-13: 978-0596527440.

SANTOSO, Yohanes. **Gnome's Guid to WEBrick**. 2004. Disponível em: <http://microjet.ath.cx/webrickguide/html/html_webrick.html>. Acesso em 22 de agosto de 2011, às 23:22.

SEGMENT7. **Writing WEBrick Servlets**. 2011. Disponível em: < http://segment7.net/projects/ruby/WEBrick/servlets.html>. Acesso em 22 de agosto, às 23:46.

SILVA, Cláudia Marin. **As novas tecnologias de informação e comunicação e a emergência da sociedade informacional**. 2006. Disponível em: <http://www.angelfire.com/sk/holgonsi/claudia.html>. Acesso em: 31 de agosto de 2011 às 20:45.

SILVA, E. L. e MENEZES, E.M. Metodologia da Pesquisa e Elaboração de Dissertação. Florianópolis: UFSC, 2001.

SOURCEFORGE, **Disponibilidade e Performance em ambientes de aplicações WEB**. 2010. Disponível em: <http://sourceforge.net/apps/wordpress/demoiselle/2010/02/12/disponibilidade-e-performance-em-ambientes-de-aplicacoes-web/>. Acesso em: 23 de agosto de 2011, às 12:04.

STEPPAT, Nico. **Banco de dados não relacionais e o movimento NoSQL**. 2009. Disponível em: <http://blog.caelum.com.br/bancos-de-dados-nao-relacionais-e-o-movimento-nosql/>. Acesso em: 20 de julho de 2011 às 09:05.

THOMAS, Dave et al. **Desenvolvimento Web Ágil com Rails. Edson Furmankiewicz**. 2ª. ed. Porto Alegre: Bookman, 2008. 680p. ISBN 978-85-7780-264-7.

VEJA DIGITAL, Revista. **Acesso à web em casa ou trabalho sobe 14% em um ano**. 2011. Disponível em: < http://veja.abril.com.br/noticia/vida-digital/acesso-a-web-em-casa-ou-trabalho-sobe-14-em-um-ano>. Acesso em: 02 de agosto de 2011 às 22:05.