Lucas Rafael Araujo Andrade

Proposta de melhoria de desempenho do SIGA utilizando Nginx como servidor HTTP

Diamantina

15 de Dezembro de 2014

Lucas Rafael Araujo Andrade

Proposta de melhoria de desempenho do SIGA utilizando Nginx como servidor HTTP

Trabalho de conclusão de curso apresentado à banca examinadora como parte dos requisitos da disciplina de Projeto Orientado II para obtenção do grau de Bacharel em Sistemas de Informação.

Universidade Federal dos Vales do Jequitinhonha e Mucuri - UFVJM

Faculdade de Ciências Exatas e Tecnológicas - FACET

Departamento de Computação - DECOM

Bacharelado em Sistemas de Informação

Orientador: Professor Dr. Alexandre Ramos Fonseca

Diamantina

15 de Dezembro de 2014

Lucas Rafael Araujo Andrade

Proposta de melhoria de desempenho do SIGA utilizando Nginx como servidor HTTP

Trabalho de conclusão de curso apresentado à banca examinadora como parte dos requisitos da disciplina de Projeto Orientado II para obtenção do grau de Bacharel em Sistemas de Informação.

APROVADO em: / /

Professor Dr. Alexandre Ramos Fonseca
Orientador

Professor Me. Áthila Rocha Trindade
Universidade Federal dos Vales do Jequitinhonha e Mucuri

Professor Me. Eduardo Pelli
Universidade Federal dos Vales do Jequitinhonha e Mucuri

Diamantina
15 de Dezembro de 2014

Dedico esse trabalho à mulher que tornou isso tudo possível. Te Amo Mãe!

Agradecimentos

Agradeço primeiramente aos meus mestres que em todos esses anos se empenharam em passar o máximo de conhecimento possível, em especial o meu orientador Alexandre pela oportunidade de desenvolver esse projeto.

Agradeço aos meus amigos de Bocaiuva e Diamantina: Diego, Hebert, Mayra, Natália, Fernanda, Lorena e Paulo Barreto. Saibam que todos, de alguma forma, fizeram diferença na minha vida e com todos vocês eu aprendi algo.

Agradeço especialmente aos meus grandes amigos Arthur, Leandro e Thalita, por todas as conversas de bar, trabalhos, shows e tardes e mais tardes escutando música, jogando conversa fora e nos divertindo. Vocês fizeram uma diferença enorme na minha vida e eu devo muito a todos vocês!

Agradeço as minhas amigas de Winnipeg: Tamara, Carol, Stefanie, Juliana e Ingrid. Com vocês por perto tudo se tornou menos difícil e mais alegre durante a minha estadia no Canadá.

Agradeço a Lili pelo apoio durante os mais de dois anos em que estivemos juntos. Saiba você que teve um papel muito importante na minha vida e lembrarei sempre dos nossos momentos juntos pro resto da minha vida.

Agradeço aos meus familiares que diretamente ou indiretamente me ajudaram durante todos esses anos de Diamantina: Tio Vendoval, Tia Letícia (*In Memorian*), Vó Nita e Tio Antônio.

Agradeço aos Técnicos-Administrativos do DTI da UFVJM: Everton, William, Marcelo, Rodrigo, André e Ricardo Brasil, por todo o conhecimento e experiência compartilhados com nós, estagiários.

E por último, porém mais importante, a mulher que por todos os meus 26 anos de vida me aturou com um amor incondicional que somente uma Mãe é capaz de sentir. Saiba, Mãe, que sem você, nada disso seria possível. Te amo incondicionalmente também!

"Code is Poetry"
(Wordpress.org)

Resumo

O presente trabalho tem por objetivo identificar o grau de eficiência do servidor HTTP Nginx, para processamento das requisições recebidas, em comparação ao servidor Apache HTTP Server, atualmente utilizado pela Universidade Federal dos Vales do Jequitinhonha e Mucuri – UFVJM em seu Sistema Integrado de Gestão Acadêmica - SIGA. Ambos os servidores em estudo estão entre os sete mais utilizados no mundo e entre os cinco mais utilizados pelos sítios mais acessados mundialmente. Os dados foram coletados após realização de testes de desempenho com o uso da ferramenta ApacheBench, instalada em duas máquinas virtuais, igualmente configuradas e instaladas através do software de virtualização VirtualBox, hospedadas em computador portátil.
Com base em levantamento do número de usuários da comunidade acadêmica da UFVJM, foram definidos quinze valores de testes, quatro cenários de acesso simultâneo e seis métricas indicadoras de eficiência. A fim de permitir uma melhor visualização dos resultados obtidos e análise, os dados gerados foram dispostos em três gráficos divididos em três faixas de valores e um gráfico com a quantidade total de requisições testadas. A análise dos resultados foi realizada para cada métrica, de modo individualizado, explicitando a média de desempenho de cada servidor. Após análise é exposta a conclusão acerca da eficiência demonstrada por pelo servidor HTTP Nginx e Apache HTTP Server e parecer sobre qual servidor se mostra mais adequado e eficiente ao SIGA.

Palavras-chaves: Apache. Nginx. SIGA. UFVJM.

Abstract

The goal of this work is to identify the Nginx HTTP server efficiency, for processing requisitions compared to the Apache HTTP server, currently in use by the Federal University of the Vales of Jequitinhonha and Mucuri – UFVJM in its Academic Integrated Management System - SIGA. Both server studied are among the seven most used in the world and among the five most used by the most accessed web sites wide world. The data has been collected after performance tests performed with the ApacheBench tool, installed in two virtual machines, equally configured and installed in the visualization software VirtualBox hosted in a portable computer.

Based on the amount of users in the academic community oh the UFVJM, was defined fifteen tests values, four simultaneous access scenarios and six efficiency metrics. In order to provide a better visualization of the results acquired, the collected data was distributed in tree graphs divided in tree ranges of values an a graph with the total amount of requisitions tested. The result analysis was made for each metric in an individual way, showing the average performance from each server. After the analysis, the conclusion about the Nginx and Apache HTTP server is shown and concludes about which server is more adequate e efficient to SIGA.

Key-words: Apache. Nginx. SIGA. UFVJM.

Lista de ilustrações

Figura 1 – Utilização de Servidores *web* no mundo. 14
Figura 2 – Utilização de Servidores *web* entre os 1.000.000 de sítios mais acessados no mundo. 15
Figura 3 – Modelo de funcionamento do Nginx. 25
Figura 4 – Média do Tempo Total de Execução dos Testes - Faixa 1 37
Figura 5 – Média do Tempo Total de Execução dos Testes - Faixa 2 37
Figura 6 – Média do Tempo Total de Execução dos Testes - Faixa 3 38
Figura 7 – Média do Tempo Total de Execução dos Testes 38
Figura 8 – Média do Total de Dados Transferido - Faixa 1 39
Figura 9 – Média do Total de Dados Transferido - Faixa 2 40
Figura 10 – Média do Total de Dados Transferido - Faixa 3 40
Figura 11 – Média do Total de Dados Transferido 41
Figura 12 – Média do Total de Texto HTML Transferido - Faixa 1 42
Figura 13 – Média do Total de Texto HTML Transferido - Faixa 2 42
Figura 14 – Média do Total de Texto HTML Transferido - Faixa 3 43
Figura 15 – Média do Total de Texto HTML Transferido 43
Figura 16 – Média do Número de Requisições Atendidas - Faixa 1 44
Figura 17 – Média do Número de Requisições Atendidas - Faixa 2 45
Figura 18 – Média do Número de Requisições Atendidas - Faixa 3 45
Figura 19 – Média do Número de Requisições Atendidas 46
Figura 20 – Média do Tempo de Resposta por Requisição Simultânea - Faixa 1 . . 47
Figura 21 – Média do Tempo de Resposta por Requisição Simultânea - Faixa 2 . . 47
Figura 22 – Média do Tempo de Resposta por Requisição Simultânea - Faixa 3 . . 48
Figura 23 – Média do Tempo de Resposta por Requisição Simultânea 48
Figura 24 – Média da Taxa de Transferência - Faixa 1 49
Figura 25 – Média da Taxa de Transferência - Faixa 2 50
Figura 26 – Média da Taxa de Transferência - Faixa 3 50
Figura 27 – Média da Taxa de Transferência 51

Lista de tabelas

Tabela 1 – Dez dias com mais acessos no SIGA 16
Tabela 2 – Requisições Totais e Requisições Concorrentes 34

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CGI	*Common Gateway Interface*
CSS	*Cascading Style Sheets*
IP	*Internet Protocol*
HTML	*Hypertext Markup Language*
HTTP	*Hipertext Transport Protocol*
ODBC	*Open Database Connection*
PDF	*Portable Document Format*
PDI	Plano de Desenvolvimento Institucional
PDO	*PHP Data Object*
PHP	*Hypertext Preprocessor*
PHP-FPM	*PHP FastCGI Process Manager*
SIGA	Sistema Integrado de Gestão Acadêmica
SGBD	Sistema de Gerenciamento de Banco de Dados
TCP	*Transmission Control Protocol*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
URI	*Uniform Resource Identifier*
URL	*Uniform Resource Locator*
UFVJM	Universidade Federal dos Vales do Jequitinhonha e Mucuri
W3C	*World Wide Web Consortium*
WWW	*World Wide Web*
XML	*Extensible Markup Language*
XHTML	*Extensible Hypertext Markup Language*

Sumário

1 INTRODUÇÃO 13
1.1 Motivação 15
1.2 Objetivos 16
1.3 Objetivos específicos 16
1.4 Organização do trabalho 17

2 FUNDAMENTAÇÃO TEÓRICA 18
2.1 Aplicação cliente/servidor 18
2.1.1 Cliente 18
2.1.2 Servidor 18
2.1.3 Aplicações cliente/servidor 18
2.2 Protocolo de comunicação 19
2.3 Modelo TCP/IP 19
2.3.1 Camada de aplicação 20
2.3.2 Camada de transporte 21
2.3.3 Camada de rede 21
2.3.4 Camada de enlace 22
2.3.5 Camada física 22
2.4 *World Wide Web* 23
2.5 Protocolo HTTP 23

3 TECNOLOGIAS UTILIZADAS 24
3.1 Apache HTTP *Server* 24
3.2 Nginx 24
3.3 ApacheBench 25
3.4 FastCGI 25
3.4.1 *Common Gateway Interface* 25
3.4.2 Servidor de API 26
3.4.3 FastCGI 27
3.5 HTML 27
3.5.1 Hipertexto 27
3.5.2 HTML 28
3.6 PHP 28
3.6.1 PHP-FPM 29
3.7 PostgreSQL 30
3.8 Miolo *framework* 30

4 METODOLOGIA 32
4.1 Coleta dos dados 32
4.2 Ambiente de testes 32
4.2.1 Computador hospedeiro 32
4.2.2 Configurações comuns às duas máquinas virtuais 33
4.2.3 Máquina virtual Apache 33
4.2.4 Máquina virtual Nginx 33
4.3 Definição dos valores utilizados nos testes 34
4.4 Métricas utilizadas 34

5 ANÁLISE DOS DADOS 36
5.1 Organização dos gráficos e dados 36
5.2 Gráficos e análise dos dados 36
5.2.1 Média do Tempo Total de Execução Dos Testes 36
5.2.2 Média do Total de Dados Transferido 39
5.2.3 Média do Total de Texto HTML Transferido 41
5.2.4 Média do Número de Requisições Atendidas 44
5.2.5 Média do Tempo de Resposta por Requisição Simultânea 46
5.2.6 Média da Taxa de Transferência 49

6 CONCLUSÃO 52
6.1 Trabalhos futuros 52

Referências 54

1 Introdução

Com o avanço dos computadores e das redes de comunicação, os sistemas de informação foram sendo transferidos dos computadores pessoais para os servidores *web*, de onde eles podem ser acessados, virtualmente, de qualquer lugar do mundo. Para que esses sistemas funcionem, eles precisam de *softwares* que atendam as requisições que chegam ao servidor a partir da rede de computadores usando o protocolo HTTP. Esses softwares são chamados servidores HTTP.

O aumento do uso de sistemas baseados na *web* por organizações criou a necessidade de desenvolver aplicações que geram conteúdo dinamicamente. São essas aplicações que permitem às organizações entregarem produtos, serviços e mensagens cujas formas e conteúdos são, em parte, determinadas pelas necessidades do usuário.

Este movimento para se afastar de conteúdos estáticos está levando ao limite e expõe as fragilidades dos ambientes onde essas aplicações são executadas. Mais importante, esses ambientes não oferecem o desempenho que essas aplicações exigem. É preciso uma nova infraestrutura de comunicação para conectar servidores *web* com essas novas aplicações.

De acordo com Bondi (2000), escalabilidade é um atributo desejável de uma rede, sistema ou processo. O conceito tem a ver com a habilidade do sistema em acomodar uma quantidade sempre maior de elementos ou objetos, de processar uma quantidade crescente de trabalho de forma fácil e ou ser suscetível a ampliação. Quando se está desenvolvimento um sistema, deseja-se que ele seja escalável.

Quando se diz que um sistema não é escalável (ou que ele não escala), damos a entender que o custo adicional de lidar com o aumento em trafego ou tamanho é excessivo, ou que o sistema não consegue lidar com esse nível elevado de acesso. O custo pode ser quantificado de várias formas, incluído, porém não limitado à tempo de resposta, uso de processamento, espaço, memória ou até mesmo dinheiro. Um sistema que não escala bem, adiciona custos de manutenção ou danifica a qualidade do serviço, pode atrasar ou privar o usuário de oportunidades de lucro e, eventualmente, precisará ser substituído.

A escalabilidade de um sistema sujeito a expansão é crucial para o seu sucesso a longo prazo. Ao mesmo tempo, o conceito de escalabilidade e o entendimento dos fatores que aumenta ou diminui são vagos e até mesmo subjetivos. Desenvolvedores de sistemas e analistas de desempenho têm um sentimento intuitivo sobre escalabilidade, pois os fatores determinantes não são claros e podem variar de um sistema para outro.

A escalabilidade de um sistema pode ser comprometida por desperdícios herdados de ações repetidas de forma frequente, pela presença de algoritmos de acesso que levam a *deadlock* ou que resultam em um escalonamento ruim de recursos. Tais sistemas podem funcionar bem quando a carga está baixa, mas sofrer uma degradação substancial de desempenho

quando a carga aumenta.

Atualmente, a Universidade Federal dos Vales do Jequitinhonha e Mucuri - UFVJM, utiliza o SIGA – Sistema Integrado de Gestão Acadêmica, adquirido da Universidade Federal de Juiz de Fora em 2007. O sistema é desenvolvido na linguagem de programação PHP com o apoio do *framework* Miolo, utiliza o Apache HTTP *Server* como servidor HTTP e o PostgreSQL como sistema de gerenciamento de banco de dados - SGBD.

O Apache, desde 1.995 tem sido o servidor HTTP mais utilizado no mundo. Em Outubro de 2.014, de acordo com pesquisa realizada em 1.028.932.208 de sítios pela empresa Netcraft, o Apache era usado por 37,79% (385.354.994) desses sítios, com o Microsoft IIS aparecendo em segundo com 33,58% (345.485.419) e o Nginx em terceiro com 14,42% (148.330.190) de utilização nos sítios, como visto na figura 1.

Nessa mesma pesquisa realizada com base nos um milhão de sítios mais acessados no mundo, o Apache aparece como o servidor mais utilizado com 50,19% (501.922) destes, sendo o Nginx aparece em segundo lugar com 20,34% (203.439) conforme a figura 2

.

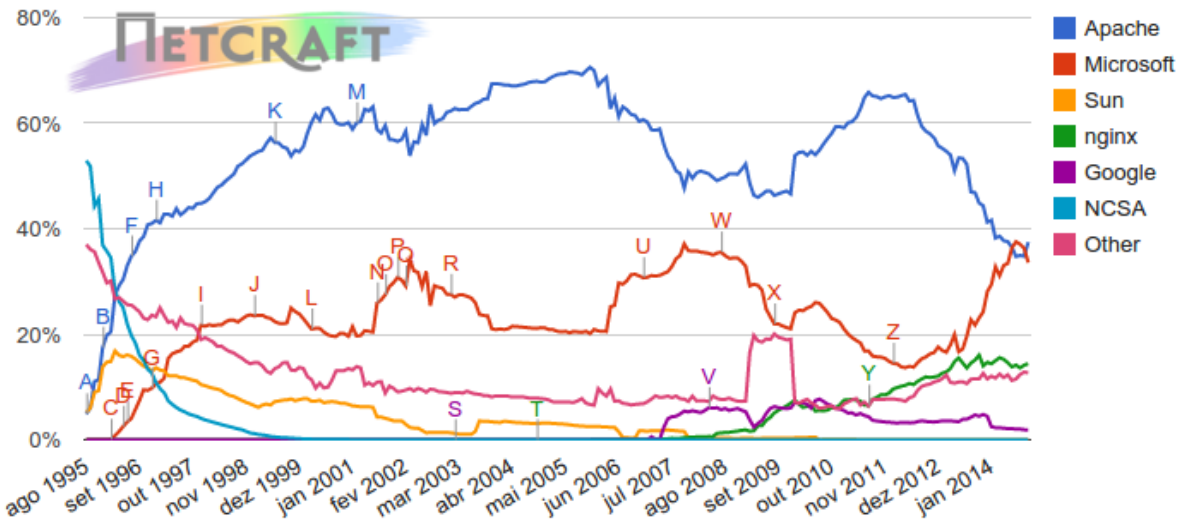

Figura 1 – Utilização de Servidores *web* no mundo.

Ao analisar os gráficos apresentados nas figuras 1 e 2, é possível notar uma queda na utilização do Apache em detrimento de outros servidores HTTP, sendo mais notória a queda na utilização entre 1 milhão de sítios mais acessados. É possível notar também o aumento no uso do Nginx, principalmente entre os 1 milhão de sítios mais acessados no mundo. É necessário frisar que só porque uma tecnologia está em ascensão, não significa que ela é melhor, mas sim é necessário investigar os motivos pelos quais a tecnologia está crescendo em número de utilizadores.

Tendo sido desenvolvidos em épocas diferentes, o Apache e o Nginx trabalham de forma diferente na hora de atender as requisições HTTP que chegam ao servidor.

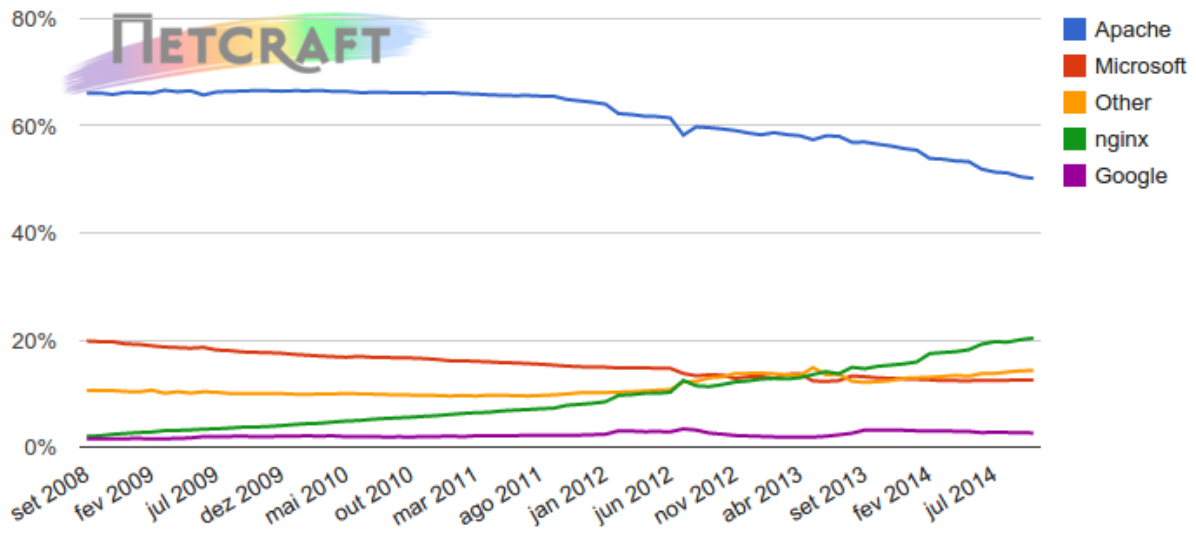

Figura 2 – Utilização de Servidores *web* entre os 1.000.000 de sítios mais acessados no mundo.

De acordo com Rowe (2014), o Apache cria processos e *threads* para lidar com as requisições, ficando a cargo do administrador do servidor a tarefa de configurar o Apache para controlar o numero máximo de processos e *threads* permitidos. Muitas *threads* podem exaurir a memória principal(RAM) e pode forçar o servidor a usar memória *SWAP*, degradando severamente o desempenho. Além disso, quando chega ao limite de processos, o Apache passa a recusar conexões.

1.1 Motivação

De acordo com o Plano de Desenvolvimento Institucional para o ciclo 2.012 - 2.016 da UFVJM, além dos quatro *campi* já implantados (Diamantina, Teófilo Otoni, Janaúba e Unaí), existe o projeto para a implantação de outros quatro *campi* universitários nas cidade de Capelinha, Araçuaí, Almenara e Nanuque, totalizando oito *campi* espalhados pelas regiões Norte, Noroeste, Vales do Jequitinhonha e Mucuri do estado de Minas Gerais. O PDI contempla também a ampliação da oferta de vagas e cursos de graduação nos já existentes *campi* de Diamantina e Teófilo Otoni. Com a expansão das universidades federais através do programa REUNI, por ano são admitidos 3500 alunos de graduação, pós-graduação e educação à distância além de novos servidores técnicos-administrativos e professores.

Em julho de 2.014, data do último levantamento, a Universidade Federal dos Vales do Jequitinhonha e Mucuri tinha 8.121 alunos, 576 professores e 421 servidores técnicos-administrativos, espalhados em quatro *campi* universitários, fazendas experimentais e pólos de ensino em educação à distância, totalizando 9.118 pessoas que interagem com a

universidade diariamente. Em épocas de pico de utilização do SIGA, as reclamações de lentidão e problemas no SIGA são frequentes, as vezes impossibilitando a utilização do mesmo. Os picos mais notório são: fim de período letivo da graduação, quando alunos e professores acessam o sistema para olhar e lançar notas, respectivamente; e rematricula dos alunos da graduação, quando os mesmos escolhem as matérias que desejam cursar no período seguinte.
De acordo com dados retirados do registro de acesso da base de dados do SIGA, do dia 08 de Fevereiro de 2013, quando se começou a fazer esse registro, até o dia 27 de Novembro de 2.014, a data em que o SIGA obteve o maior número de acessos foi em 16 de Agosto de 2.014 com 20.942 acessos em 24 horas, uma média de 873 registros de entrada por hora. O período de 9 à 10 horas do dia 16/08 foi a hora com mais acessos da história com 1.351 registros. Os dez dias com mais acessos ao SIGA estão representados na tabela 1. Com o

Tabela 1 – Dez dias com mais acessos no SIGA

Dia	**Acessos**	**Ocasião**
16/08/2014	20.942	Início da Pré Matrícula
25/08/2014	17.536	Início do Período Letivo
16/04/2013	17.091	Dias Finais do Período Letivo
30/09/2013	17.007	Início da Pré Matrícula
17/04/2013	16.772	Dias Finais do Período Letivo
15/04/2013	16.597	Dias Finais do Período Letivo
31/07/2014	16.035	Dias Finais do Período Letivo
29/07/2014	15.744	Dias Finais do Período Letivo
28/07/2014	15.477	Dias Finais do Período Letivo
30/07/2014	14.854	Dias Finais do Período Letivo

Fonte: Base de Dados do SIGA

crescente aumento de alunos, servidores públicos (professores e técnicos-administrativos) e teceirizados na universidade, a tendência é que a utilização do SIGA se torne mais problemática. Com isso em mente, a utilização do servidor HTTP Nginx pode ajudar a amenizar os problemas de desempenho do SIGA.

1.2 Objetivos

Identificar se a utilização do servidor HTTP Nginx é mais eficiente do que o utilizado atualmente, o Apache HTTP *Server*.

1.3 Objetivos específicos

Analisar os dados coletados a partir de testes realizados para identificar se o Nginx é mais eficiente do que o Apache; apresentar a solução encontrada e analisar o que pode

ser feito para evitar a substituição ou reconstrução do SIGA.

1.4 Organização do trabalho

O trabalho está organizado em seis capítulos, sendo essa introdução o primeiro. No capítulo 2, será exposto toda a teoria por trás do estudo feito. No capítulo 3, serão descritos os programas e ferramentas utilizados de forma direta ou indireta nos testes. No capítulo 4 será descrito como os testes foram realizados, quais dados foram coletados e descrever os ambientes onde foram feitos os teste de desempenho do Apache e do Nginx. No capítulo 5 os dados serão apresentados e será feita uma análise sobre o desempenho dos dois servidores HTTP. E finalmente, no capítulo 6 será apresentada uma conclusão sobre o estudo.

2 Fundamentação Teórica

Este capítulo se destina a apresentar alguns conceitos teóricos para o entendimento da proposta desse estudo.

2.1 Aplicação cliente/servidor

Para entender o que é uma aplicação cliente/servidor, é preciso primeiramente definir o que é cliente e servidor.

2.1.1 Cliente

Cliente é:

> Um solicitante de informações em rede, normalmente um computador ou estação de trabalho, que pode consultar o banco de dados e/ou outras informações de um servidor. (STALLINGS, 2005)

Ainda de acordo com Stallings (2005), o cliente é um computador ou estação de trabalho, que apresenta ao usuário final uma interface gráfica de alto nível e amigável, incluindo o uso de janelas e *mouse*.

2.1.2 Servidor

Servidor é:

> Um computador, normalmente uma estação de trabalho poderosa ou um *mainframe* que abriga informações para manipulação por clientes em rede. (STALLINGS, 2005)

Ainda de acordo com Stallings (2005), cada servidor no ambiente cliente/servidor fornece um conjunto de serviços compartilhados.

2.1.3 Aplicações cliente/servidor

As aplicações cliente/servidor são um conjunto de programas que funcionam no computador cliente, ou seja, aquele que esta solicitando um recurso pela rede, e no computador servidor, aquele que provê o recurso solicitado. As solicitações de recursos ou informações partem do programa instalado no computador do usuário (cliente), trafegam por uma rede de computadores, geralmente utilizando o protocolo de comunicação HTTP até o servidor responsável por prover o recurso ou informação solicitado pelo programa do

cliente.
As aplicações cliente/servidor são muito utilizadas na construção de sistemas de informação, onde os dados armazenados pelos servidores precisam estar disponíveis para os usuários do sistema. É também o principal modelo utilizado em sistemas de informação baseados na *web*, onde o cliente acessa o sistema através de um navegador *web*.

2.2 Protocolo de comunicação

De acordo com o Dicionário Online de Português, um dos significados de protocolo é:

> Acordo regulamentado entre países ou empresas. (PORTUGUêS, 2014)

Aplicando esse conceito na computação, protocolo pode ser entendido como uma convenção que controla e possibilita uma conexão, comunicação e transferência de dados entre dois sistemas computacionais. De maneira simples, protocolos podem ser definidos como as regras que governam a sintaxe, semântica e sincronização da comunicação. De acordo com Stallings (2005), um protocolo é usado para a comunicação entre entidades em sistemas diferentes. Para que entidades diferentes se comuniquem com facilidade, elas precisam adotar uma "língua" em comum. O quê e como é comunicado, deve estar de acordo com as convenções combinadas entre as entidades envolvidas. Essas convenções são denominadas protocolos, que são conjuntos de regras para controlar a troca de dados entre entidades. Os principais elementos de um protocolo são:

- Sintaxe: Inclui elementos como formato de dados e níveis de sinal;
- Semântica: Inclui informações de controle para coordenação e tratamento de erro;
- Temporização: Inclui combinação de velocidade e sequência.

2.3 Modelo TCP/IP

O TCP/IP (*Transmission Control Protocol/Internet Protocol*), é um conjunto de protocolos, resultante de pesquisas e desenvolvimentos realizados pela ARPANET (*Advanced Research Project Agency Network*) e financiada pela DARPA (*Defense Advanced Research Projects Agency*), agência de pesquisa militar do governo dos Estados Unidos, e que hoje são utilizados como padrões de comunicação na Internet.
Ao contrário do modelo OSI(Open Systems Interconnections), o TCP/IP não pode ser considerado um modelo e sim um conjunto de protocolos. Esse protocolos, assim como no modelo OSI, podem ser organizados em camadas relativamente independentes, com a

diferença de que no modelo OSI existem sete camadas e no TPC/IP existem cinco. As camadas do TCP/IP são:

- Camada de aplicação
- Camada de transporte
- Camada de rede
- Camada de enlace
- Camada física

Para efeito de comparação, as camadas do modelo OSI são:

- Camada de aplicação
- Camada de apresentação
- Camada de sessão
- Camada de transporte
- Camada de rede
- Camada de enlace
- Camada física

2.3.1 Camada de aplicação

A camada de aplicação é a camada que a maioria dos programas usa de forma a se comunicar através de uma rede com outros programas. Processos que funcionam nessa camada são específicos da aplicação; o dado é passado do programa de rede, no formato usado internamente por essa aplicação, e é codificado dentro do padrão de um protocolo. Existem diversos protocolos nesta camada. Os mais comuns são:

- SMTP (*Simple Mail Transport Protocol*): é utilizado para a comunicação entre serviços de correio eletrônico na Internet.
- POP (*Post Office Protocol*): é utilizado para recuperação de mensagens de correio eletrônico via Internet.
- IMAP (*Internet Mail Access Protocol*): é utilizado para recuperação de mensagens de correio eletrônico via Internet, mais avançado que o POP.

- HTTP (*Hypertext Transport Protocol*): utilizado para a publicação de sites *web* na Internet.
- FTP (*File Transfer Protocol*): utilizado para publicação e transferência de arquivos via Internet.

2.3.2 Camada de transporte

A camada de transporte (ou *host* a *host*) é responsável pela transferência eficiente, confiável e econômica dos dados entre o computador de origem e de destino, independente do tipo, topologia ou configuração das redes físicas existentes entre elas, garantindo ainda que os dados cheguem sem erros e na sequência correta.
A camada de transporte é uma camada fim-a-fim, isto é, uma entidade desta camada só se comunica com a sua entidade semelhante do destinatário. A camada de transporte provê mecanismos que possibilitam a troca de dados fim-a-fim, não possibilitando, assim, a comunicação com entidades intermediárias. A camada possui dois protocolos: o UDP (*User Datagram Protocol*) e o TCP (*Transmission Control Protocol*).
O protocolo UDP realiza apenas a multiplexação para que várias aplicações possam acessar o sistema de comunicação de forma coerente.
O protocolo TCP realiza, além da multiplexação, uma série de funções para tornar a comunicação entre origem e destino mais confiável. São responsabilidades do protocolo TCP: o controle de fluxo e erro, a sequenciação e multiplexação de mensagens.

2.3.3 Camada de rede

A camada de rede (ou inter-rede) é responsável por controlar a operação da rede de um modo geral. Suas principais funções são o roteamento dos pacotes entre fonte e destino, mesmo que estes tenham que passar por diversos nós intermediários durante o percurso, o controle de congestionamento e a contabilização do número de pacotes ou bytes utilizados pelo usuário, para fins de tarifação.
O principal aspecto que deve ser observado nessa camada é a execução do roteamento dos pacotes entre fonte e destino, principalmente quando existem caminhos diferentes para conectar entre si dois nós da rede. Em redes de longa distância é comum que a mensagem chegue do nó fonte ao nó destino passando por diversos nós intermediários no meio do caminho e é tarefa dos protocolos da camada de rede escolher o melhor caminho para essa mensagem.
A escolha da melhor rota pode ser baseada em tabelas estáticas, que são configuradas na criação da rede e raramente são modificadas; ser determinada no início de cada conversação, ou ser altamente dinâmica, sendo determinada a cada novo pacote, a fim de refletir exatamente a carga da rede naquele instante. Se muitos pacotes estão sendo transmitidos

através dos mesmos caminhos, eles vão diminuir o desempenho global da rede, formando gargalos.

2.3.4 Camada de enlace

A camada de enlace (ou de acesso à rede) trata da troca de dados entre um sistema final e a rede à qual está conectada (STALLINGS, 2005). Para que haja uma comunicação, o computador de origem da mensagem deve informar para a camada de enlace qual o endereço do computador de destino, para que a rede possa entregar a mensagem para o destinatário correto. O computador de origem pode requerer alguns serviços especiais, como por exemplo, prioridade no envio da mensagem. O software utilizado nessa camada irá depender do tipo de rede a ser usado, sendo que diferentes padrões foram desenvolvidos para comutação de circuitos de pacotes.
Quando dois computadores estão conectado a redes diferentes, faz-se necessário o uso de procedimentos para que os dados trafeguem entre as redes. Nesses casos, se utiliza o protocolo IP (*Internet Protocol*). Esse protocolo oferece a função de interconectar várias redes através de roteadores. Roteadores são computadores especializados em conectar duas ou mais redes diferentes, e o protocolo IP é também utilizado e implementado nos roteadores.

2.3.5 Camada física

A camada física (interface de rede) é a primeira camada do modelo TCP/IP. Também chamada de camada de abstração de hardware, tem como função principal fazer a interface do modelo TCP/IP com os diversos tipos de redes (X.25, ATM, FDDI, Ethernet, *Token Ring*, *Frame Relay*, sistema de conexão ponto-a-ponto SLIP, etc.) e transmitir os datagramas pelo meio físico.
Esta camada lida com os meios de comunicação, corresponde ao nível de hardware, ou meio físico, que trata dos sinais eletrônicos, conector, pinagem, níveis de tensão, dimensões físicas, características mecânicas e elétricas etc. Os protocolos da camada física enviam e recebem dados em forma de pacotes, que contém um endereço de origem, os dados propriamente ditos e um endereço de destino. É responsável pelo endereçamento e tradução de nomes e endereços lógicos em endereços físicos. Ela determina a rota que os dados seguirão do computador de origem até o de destino. Tal rota dependerá das condições da rede, prioridade do serviço dentre outros fatores.
Também gerencia o tráfego e taxas de velocidade nos canais de comunicação. Outra função que pode exercer é o agrupamento de pequenos pacotes em um único pacote para transmitir pela rede (ou a subdivisão de pacotes grandes). No destino os dados são recompostos no

seu formato original.

2.4 *World Wide Web*

A *World Wide Web* (WWW, ou *Web*), foi proposta em 1.989 pelo cientista britânico Sir Tim Berners-Lee quando trabalhava no CERN (Laboratório Europeu para Partículas Físicas). A ideia de Berners-Lee era propor uma tecnologia de hipermídia distribuída com o objetivo de prover o compartilhamento internacional de descobertas científicas usando a Internet.
A *web* é um sistema distribuído que consiste em uma coleção de arquivos armazenados em servidores e que podem ser acessados a partir de programas instalados nos computadores dos clientes, os chamados navegadores. Cada arquivo tem um endereço em forma de URL. Os usuário podem navegar de um arquivo para outro (ou entre páginas) fazendo uso do *mouse* do computador para clicar em um *link* que irá direcionar o usuário para a página requisitada. Por ser uma tecnologia que está presente na camada de aplicação, o protocolo utilizado é o HTTP

2.5 Protocolo HTTP

De acordo com Stallings (2005), o *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto, em tradução livre) é o protocolo básico da *World Wide Web* e pode ser usado em qualquer aplicação cliente/servidor que envolve hipertexto e está presente na camada de aplicação do modelo TCP/IP.
O HTTP não serve somente para transferir hipertexto. É um protocolo para transferir informações com uma eficiência necessária. Os dados transferidos podem ser texto puro, hipertexto, áudio, imagens ou qualquer outro dado disponível na Internet.
O HTTP é um protocolo cliente/servidor orientado à transações. O seu uso mais típico é acontece entre um navegador *web* e um servidor *web*, sempre utilizando o protocolo TCP, presente na camada de transporte, afim de manter a confiabilidade.
Cada transação utiliza uma nova conexão TCP entre cliente e servidor, sendo tratada de forma independente. Após o término de cada transação, a conexão entre cliente e servidor é fechada. Por ser um protocolo sem estado, é o adequado para a maior parte das aplicações.

3 Tecnologias Utilizadas

3.1 Apache HTTP *Server*

O Apache HTTP *Server* foi lançado oficialmente em Abril de 1.995. Foi criado para ocupar o lugar deixado pelo HTTP *Daemon*, na época o servidor para aplicações web mais utilizado no mundo. O HTTP *Daemon* foi desenvolvido por Rob McCool quando trabalhava no *National Center for Supercomputing Applications* – NCSA, na Universidade de Illinois, nos Estados Unidos. Porém, o desenvolvimento do HTTP *Daemon* estagnou-se quando McCool deixou a universidade. Como o código do HTTP *Daemon* era aberto (*open source*), vários desenvolvedores criaram correções e desenvolveram novas funcionalidades para o mesmo. Vendo a necessidade de juntar todos esses códigos desenvolvidos de forma separada, um grupo de desenvolvedores resolveram se juntar para compilar essas correções e novas funcionalidades. Usando como base a versão 1.3 do HTTP *Daemon*, em Abril de 1.995 foi publicado o Apache HTTP *Server* na versão 0.6.2. Também, nessa mesma época, foi criado o Apache *Group*, grupo que mais tarde viria a se tornar o Apache *Software Foundation*.

Hoje, quase 20 anos após o seu primeiro lançamento, o Apache HTTP *Server* é o servidor HTTP mais utilizado no mundo e a sua versão estável atual é a 2.4.

3.2 Nginx

O Nginx (lê-se *Engine-X*) foi criado pelo russo Igor Sysoev em 2.002 tendo a primeira versão publicada em 2.004. O Nginx foi desenvolvido com o intuito de resolver o C10K *problem*.

Diferentemente de outros servidores HTTP, o Nginx não usa *threads* como base para manipular requisições. Ao invés disso, utiliza uma arquitetura mais escalável orientada à eventos (*event-driven*) assíncrona. Essa arquitetura permite utilizar quantidades pequenas, porém previsíveis, de memória principal quando está funcionando.

O Nginx é utilizado por vários sítios de grande tráfego como Netflix, GitHub, Pinterest, dentre outros.

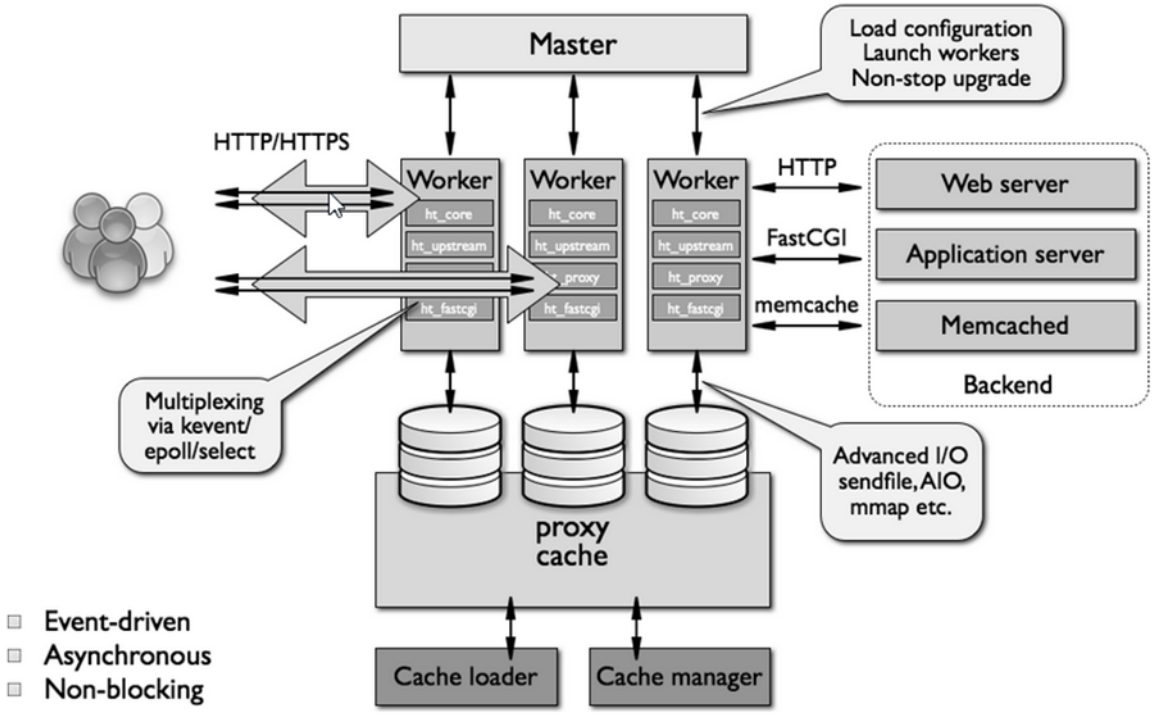

Figura 3 – Modelo de funcionamento do Nginx.

3.3 ApacheBench

O ApacheBench foi criado em 1.996 por Adam Twiss e, posteriormente doado ao Apache *Group*. Originalmente, a ferramenta foi desenvolvida para verificar o desempenho em servidores HTTP Apache, mas hoje é utilizada para fazer testes de desempenho em praticamente qualquer servidor HTTP.

3.4 FastCGI

De acordo com FastCGI (2014) FastCGI é uma interface para servidores *web* rápida, aberta e segura, que resolve os problemas de desempenho herdados do CGI, sem introduzir *overhead* e complexidade de APIs proprietárias.

3.4.1 *Common Gateway Interface*

A interface de fato de aplicações em servidores *web* é o CGI, implementado pela primeira vez nos servidores da NCSA. O CGI tem muitos benefícios:

- Simplicidade: é fácil de entender;
- Independente de linguagem: Aplicações em CGI podem ser escritas em quase todas a linguagens;

- Isolamento do processo: Como os processos são executados separadamente, aplicações com problema não param ou acessam o estado interno do servidor *web*;
- Padrão aberto: CGI já foi implementado em praticamente todos os servidores;
- Independência de arquitetura: O CGI não é ligado a uma arquitetura de computador em particular.

CGI tem alguns inconvenientes significantes, sendo o principal problema o desempenho: como um novo processo é criado para cada requisição e descartada quando a requisição acaba, a eficiência é baixa.

3.4.2 Servidor de API

Em resposta ao problema de desempenho do CGI, várias empresas desenvolveram API's para os seus servidores.
Aplicações conectadas em um servidor de API pode ser significativamente mais rápido do que programas desenvolvidos em CGI. O problema da inicialização do CGI é melhorada, pois a aplicação é executada no processo do servidor e persiste pelas requisições. As API's dos servidores *web* também oferecem funcionalidades adicionais comparadas com o CGI. O desenvolvedor pode criar extensões que permitem realizar controle de acesso, pegar um acesso dos aquivos de registro (*log*) do servidor e, se conectar a outros estágios do processamento de uma requisição do servidor. No entanto, API's sacrificam todos os benefícios do GCI. São eles:

- Complexidade: API's de empresas introduzem uma curva de aprendizado, custos de implementação e manutenção maiores;
- Dependência de linguagem: as aplicações devem ser escritas na linguagem de programação suportada pela API;
- Não há isolamento de processos: como o processo é executado dentro do endereço de memória do servidor, aplicações com problemas podem corromper o núcleo do servidor, comprometer a segurança e problemas no núcleo do servidor podem corromper as aplicações;
- Proprietário: Codificar a aplicação para uma determinada API força o desenvolvedor a utilizar aquele servidor em particular;
- Arquiteturas de computador iguais: As aplicações devem usar a mesma arquitetura do servidor *web*.

3.4.3 FastCGI

A interface do FastCGI combina os melhores aspectos do CGI e das API's proprietárias. Assim como o CGI, o FastCGI executa os processos de forma separada e isolada. As vantagens do FastCGI incluem:

- Desempenho: Os processos do FastCGI são persistentes, sendo reutilizados para manipular várias requisições, resolvendo o problema da criação de novos processos para cada requisição;
- Simplicidade com fácil migração do CGI: a biblioteca de aplicação do FastCGI simplifica a migração de aplicações existentes feitas usando CGI. Aplicações feitas utilizando a biblioteca de aplicações do FastCGI podem ser executadas como programa CGI;
- Independência de linguagem: Assim como o CGI, as aplicações FastCGI podem ser escritas em qualquer linguagem;
- Isolamento de processos: Uma aplicação com problemas não pode corromper o núcleo do servidor ou de outra aplicação.
- Independência de arquitetura: O FastCGI não é ligado a uma arquitetura de computador em particular. Qualquer servidor *web* pode implementar a interface do FastCGI;
- Suporte à computação distribuída: FastCGI provê a habilidade de executar aplicações remotamente, o que é útil para distribuir carga e gerenciar sítios da *web* externos.

3.5 HTML

De acordo com o Consortium (2014) a Web é baseada em 3 pilares:

- Um esquema de nomes para localização de fontes de informação na Web a URI (*Uniform Resource Identifier*);
- Um Protocolo de acesso para acessar estas fontes, hoje o HTTP;
- Uma linguagem de Hipertexto, para a fácil navegação entre as fontes de informação: o HTML.

3.5.1 Hipertexto

HTML é a abreviação para *Hypertext Markup Language* - Linguagem de Marcação de Hipertexto. Em resumo, HTML é uma linguagem para publicação de conteúdo (texto,

imagem, vídeo, áudio e etc) na *web*.
HTML é baseado no conceito de Hipertexto. Hipertextos são conjuntos de elementos – ou nós – ligados por conexões. Estes elementos podem ser palavras, imagens, vídeos, áudio, documentos etc.. Estes elementos conectados formam uma grande rede de informação que não estão conectados linearmente como se fossem textos de um livro, onde um assunto é ligado ao outro seguidamente. A conexão feita em um hipertexto é algo imprevisível que permite a comunicação de dados, organizando conhecimentos e guardando informações relacionadas.
Para distribuir informação de maneira global, é necessário haver uma linguagem que seja entendida universalmente por diversos meios de acesso. O HTML se propõe a ser esta linguagem. Desenvolvido originalmente por Tim Berners-Lee o HTML ganhou popularidade quando o *Mosaic - browser*, desenvolvido por Marc Andreessen na década de 1990, ganhou força. A partir de então, desenvolvedores e fabricantes de navegadores passaram a utilizar o HTML como base, compartilhando sas mesmas convenções.

3.5.2 HTML

Entre 1993 e 1995, o HTML ganhou as versões HTML+, HTML2.0 e HTML3.0, onde foram propostas diversas mudanças para enriquecer as possibilidades da linguagem. Contudo, até 1.995 o HTML ainda não era tratado como um padrão. Apenas em 1997, o grupo de trabalho W3C, responsável por manter o padrão do código, trabalhou na versão 3.2 da linguagem, fazendo com que ela fosse tratada como padrão.
Desde o começo o HTML foi criado para ser uma linguagem independente de plataformas, navegadores e outros meios de acesso. Interoperabilidade significa menos custo. O desenvolvedor cria apenas um código HTML e este código pode ser lido por diversos dispositivos ao invés de versões diferentes para cada dispositivo. Dessa forma, evita-se que a *Web* seja desenvolvida em uma base proprietária, com formatos incompatíveis e de forma limitada. Por esses motivos o HTML foi desenvolvido para que essa barreira fosse ultrapassada, fazendo com que a informação publicada por meio deste código fosse acessível por dispositivos e outros meios com características diferentes, não importando o tamanho da tela, resolução, variação de cor, dentre outros. O HTML deve ser entendido universalmente, dando a possibilidade para a reutilização dessa informação de acordo com as limitações de cada meio de acesso.

3.6 PHP

PHP (acrônimo para preprocessador de hipertexto) é uma linguagem de programação de código aberto, interpretada, de propósito geral, de tipagem dinâmica e fraca, procedural, reflexiva, orientada a objetos e funcional; criada em 1.995 por Rasmus Lerdorf.

A linguagem é melhor utilizada para criação de sistemas baseados na *web*, já que pode ser inserida diretamente em códigos HTML.
De acordo com Php.net (2014), o PHP pode realizar qualquer tipo de atividade computacional. Tem como foco executar tarefas do lado do servidor, realizando qualquer tarefa que uma aplicação feita em CGI pode fazer tais como: coletar dados de um formulário, gerar páginas com conteúdo dinâmico ou enviar e recebe *cookies*. Existem três áreas onde o PHP é mais utilizado:

- *Script* do lado do servidor: é a forma mais tradicional e o principal foco do PHP;
- *Script* de linha de comando: é uma forma de utilizar o PHP sem um servidor *web* ou navegador de internet. Esse forma de uso é ideal para rotinas programadas executadas no sistema operacional. Pode ser usada também para tarefas de processamento de texto;
- Aplicações para *Desktop*: PHP provavelmente não é a melhor linguagem para desenvolver aplicações com interface gráfica para computadores de mesa, mais pode ser utilizada para criação de aplicações do lado do cliente utilizando o a extensão PHP-GTK, não distribuída de forma oficial pelos mantenedores da linguagem PHP.

Para fazer o PHP funcionar é preciso três coisas: Um interpretador, um servidor *web* e um navegador de *internet*. Pode ser utilizado em praticamente todos os sistemas operacionais e ser executado em qualquer servidor *web* que utilize a extensão FastCGI PHP.
Com o PHP, o desenvolvedor não fica limitado a somente gerar páginas HTML, incluindo as habilidades de entregar imagens e arquivos em geral, geração de páginas em XHTML e qualquer outro arquivo XML e servindo como *cache* no servidor para o conteúdo gerado dinamicamente.
Uma das funcionalidades mais importantes do PHP é o suporte a uma grande variedade de bancos de dados. Desenvolver páginas que utilizam base de dados é simples, desde que se use uma extensão para base de dados presente na linguagem, uma camada de abstração, como por exemplo o PDO, ou conectar a qualquer base de dados que suporte o padrão ODBC.
O PHP tem ferramentas de processamento de texto úteis, incluindo analisador de expressões regulares compatível com a linguagem de programação Perl, além de várias outras extensões e ferramentas para analisar e acessar documentos no formato XML.

3.6.1 PHP-FPM

O PHP-FPM (*(FastCGI Process Manager*) é uma implementação alternativa do PHP FastCGI com algumas funcionalidades úteis, principalmente, para sítios com grande acesso. Essas funcionalidade incluem:

- Gerenciamento avançado de processos;
- Habilidade para iniciar processos com diferentes usuários, grupos e ambientes, escutando portas diferentes e usando diferentes arquivos de configuração;
- Registro (*log*) de atividades nas saídas padrão de texto (stdout) e de erros (stderr) do sistema operacional;
- Reinicialização emergencial em caso de destruição acidental da memória cache;

3.7 PostgreSQL

De acordo com PostgreSQL (2014) PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional que tem sido desenvolvido de várias formas desde 1.977. Começou como um projeto chamado Ingres na *University of California* em Berkeley, Estados Unidos. O Ingres foi, posteriormente, desenvolvido comercialmente pela empresa Relational Technologies.
Em 1.986 uma outra equipe chefiada por Michael Stonebraker continuou o desenvolvimento do código do Ingres para criar um sistema de banco de dados usando o paradigma objeto-relacional chamado Postgres. Em 1.996, o Postgres foi renomeado para PostgreSQL. O PostgreSQL é considerado por muitos o melhor SGBD de código aberto do mundo, provendo várias funcionalidades que, normalmente, são vistas somente em produtos comerciais desenvolvidos para grande empresas.

3.8 Miolo *framework*

O Miolo é um *framework* para criação de sistemas de informação acessíveis via *web* escrito em PHP. O Miolo utiliza *scripts* javascript e conceitos de programação orientada a objetos, gerando páginas HTML e arquivos PDF. Com um projeto modular, baseado em componentes, e uma arquitetura em camadas, o Miolo atua como "*kernel*" de todos os sistemas criados. Favorecendo a reutilização, os vários sistemas podem ser facilmente integrados, funcionando como módulos de um sistema mais complexo. Além de proporcionar as funcionalidades para o desenvolvimento de sistemas, o Miolo também define uma metodologia de codificação para que os resultados esperados sejam obtidos de forma simples e rápida.
Os sistemas desenvolvidos com o Miolo seguem algumas regras básicas que devem ser conhecidas, como as definições de classes/*forms*/*handlers* dos módulos, definição dos arquivos de configuração (localização dos programas, componentes, bancos de dados, temas, etc.), o ciclo de vida de execução de uma requisição do cliente, além das principais

classes, métodos e controles. Algumas das principais funções implementadas pelo framework são:

- Controles de interface com o usuário, escritos em PHP e renderizados em HTML;
- Autenticação de usuários;
- Controle de permissão de acesso;
- Camada de abstração para acesso a bancos de dados;
- Camada de persistência transparente de objetos;
- Gerenciamento de sessões e estado;
- Validação de entrada em formulários;
- Customização de *layout* e temas usando CSS;
- Geração de arquivos em PDF;

4 Metodologia

A descrição da metodologia utilizada para a coleta e análise dos dados é de importante para o entendimento do estudos e dos resultados. A partir dos dados coletados é que será possível determinar se os objetivos foram alcançados.

4.1 Coleta dos dados

Para realizar a coleta de dados dos servidores foi utilizada o *software* ApacheBench. O ApacheBench é uma ferramenta destinada a realizar testes de carga em servidores *web* geralmente utilizada em linha de comando sendo invocada com o comando "ab" no terminal, aceitando como parâmetros, entre outros, a quantidade total de requisições que serão feitas, quantas requisições serão feitas de forma simultânea e a URL da página que será requisitada. O comando de uma forma genérica fica assim:

```
ab -n numero_de_requisições -c requisições_simultâneas endereço_do_servidor
```

O parâmetro "-n" indica a quantidade total de requisições serão feitas no teste e o parâmetro, "-c" indica a quantidade de requisições serão feitas de forma simultânea. Ao fim da execução do teste, o ApacheBench retorna os os dados coletados e calculados.

4.2 Ambiente de testes

Para fazer os teste da forma mais neutra possível, foram criadas duas máquinas virtuais usando o software de virtualização VirtualBox na sua versão 3.4.10.
Todos os programas utilizados nos ambientes de teste foram instalados pelo gerenciador de pacotes nativo da distribuição Linux Debian *aptitude* e disponíveis nos repositórios oficiais do Debian, exceto o SGBD PostgreSQL que foi instalado usando o *aptitude* porém através do repositório oficial do PostgreSQL para o Linux Debian 7 pois nos repositórios do Debian não havia disponível a versão mais recente e utilizada pelo banco de dados do SIGA.

4.2.1 Computador hospedeiro

O computador onde foram instaladas as máquinas virtuais possui a seguinte configuração:

- **Tipo**: Computador portátil (*Notebook*)
- **Processador**: Intel Core i5-3337U
- **Frequência do Processador**: 1,8 Giga Hertz
- **Tamanho da Memória Principal (RAM)**: 8 Giga Bytes;
- **Tamanho da Memória Secundária (HD)**: 1 Tera Bytes;
- **Quantidade de núcleos disponível para processamento**: 4 núcleos.

4.2.2 Configurações comuns às duas máquinas virtuais

Ambas máquinas virtuais tinhas as configurações rigorosamente iguais. São elas:

- **Sistema Operacional**: Linux Debian 7 "Wheezy" 64 bits mais atualizado;
- **Sistema Gerenciador de Banco de Dados**: PostgreSQL na versão 9.3.5
- **Tamanho da Memória Principal (RAM)**: 2.048 Mega Bytes;
- **Tamanho da Memória Secundária (HD)**: 30 Giga Bytes;
- **Quantidade de núcleos disponível para processamento**: 1 núcleo.

4.2.3 Máquina virtual Apache

Na máquina virtual destinada aos testes com o servidor HTTP Apache, os programas utilizados foram:

- Apache HTTP Server na versão 2.2.22;
- libapache2-mod-php5 na versão 5.4.4;
- php5-common na versão 5.4.4.

4.2.4 Máquina virtual Nginx

Na maquina virtual destinada aos testes com o servidor HTTP Nginx, os programas utilizados foram:

- Servidor HTTP Nginx na versão 1.2.1;
- php5-fpm na versão 5.4.4;
- php5-common na versão 5.4.4.

4.3 Definição dos valores utilizados nos testes

De acordo com os últimos dados disponibilizados em Julho de 2014, a universidade contava, na época, com 9.118 utilizadores, entre alunos, professores e funcionários técnicos-administrativos. Baseando-se nesses dados, os valores utilizados nos testes foram estimados com cerca de 7.000 usuários ativos atualmente projetando mais que o dobro de usuário no futuro (15.000) e com cargas de acesso simultâneo de 10%, 20%, 40% e 80% para cada quantidade total de utilizadores. Os valores utilizados nos testes estão descritos na tabela 2. Para cada valor de requisições totais, foram testados quatro cenários de acesso

Tabela 2 – Requisições Totais e Requisições Concorrentes

Requisições Totais	**Requisições Concorrentes**			
	10%	20%	40%	80%
1.000	100	200	400	800
2.000	200	400	800	1.600
3.000	300	600	1.200	2.400
4.000	400	800	1.600	3.200
5.000	500	1.000	2.000	4.000
6.000	600	1.200	2.400	4.800
7.000	700	1.400	2.800	5.600
8.000	800	1.600	3.200	6.400
9.000	900	1.800	3.600	7.200
10.000	1.000	2.000	4.000	8.000
11.000	1.100	2.200	4.400	8.800
12.000	1.200	2.400	4.800	9.600
13.000	1.300	2.600	5.200	10.400
14.000	1.400	2.800	5.600	11.200
15.000	1.500	3.000	6.000	12.000

Quantidade de Requisições Totais e Requisições Concorrentes

simultâneo: 10, 20, 40, e 80 porcento, totalizando 60 testes. Os dados obtidos nos testes foram tabelados em uma planilha eletrônica com a ajuda do LibreOffice Calc e analisados a partir de gráficos e dos dados brutos.

4.4 Métricas utilizadas

As métricas utilizadas para comparar o desempenho dos servidores HTTP são as mesmas métricas calculadas e entregues pelo ApacheBench. São elas:

- Tempo total do teste em segundos (s);
- Total de dados transferido em bytes (b);
- Total de texto em HTML transferido em bytes (b);
- Tempo médio por requisição em milissegundos (ms);

- Tempo médio de resposta por requisição entre as requisições concorrentes em milissegundos (ms);
- Taxa de transferência em Quilo Bytes por segundo (Kb/s);

5 Análise dos dados

A análise dos dados é essencial para o estudo. A partir da análise é que será possível determinar se o servidor HTTP objeto do estudo, o Nginx é mais eficiente do que o Apache. Os dados utilizados nessa análise são a média dos quatro testes realizados para cada quantidade total de requisições, correspondentes as porcentagens de requisições feitas de forma simultânea.
A análise será feita a partir da construção de gráficos comparativos e análise dos dados brutos coletados nos testes, observando a média entre os quinze valores coletados de acordo com a quantidade de requisições e dentro de cada faixa de valores para requisições. As faixas de valores de requisição estão explicadas a seguir.

5.1 Organização dos gráficos e dados

Para melhor analisar o comportamento dos servidores em cada métrica, foram gerados quatro gráficos: três separados em faixas de valores de acordo com a quantidade total de requisições e um gráfico com todos os valores de requisições. As faixas são:

- **Faixa 1** - Entre 1.000 e 5.000 requisições totais;
- **Faixa 2** - Entre 6.000 e 10.000 requisições totais;
- **Faixa 3** - Entre 11.000 e 15.000 requisições totais;

Dispondo os gráficos dessa forma, será possível analisar os dados de acordo com a quantidade de usuários utilizando o sistema e analisar o comportamento dos servidores HTTP de uma forma geral.

5.2 Gráficos e análise dos dados

Todos os gráficos foram gerados usando o LibreOffice Calc.

5.2.1 Média do Tempo Total de Execução Dos Testes

O tempo total de execução do tem como finalidade mostrar o tempo em que o processo do ApacheBench responsável pelo teste ficou em execução no sistema operacional. Tendo em vista que o processo do ApacheBench pode fica em execução por mais tempo do que a realização do teste e que uma requisição pode demorar a responder, atrasando o fim do teste, essa métrica provê uma noção do desempenho do servidor, mais não deve ser

levado em consideração em uma análise adequada. No caso desta métrica, quanto menor o valor observado, melhor.

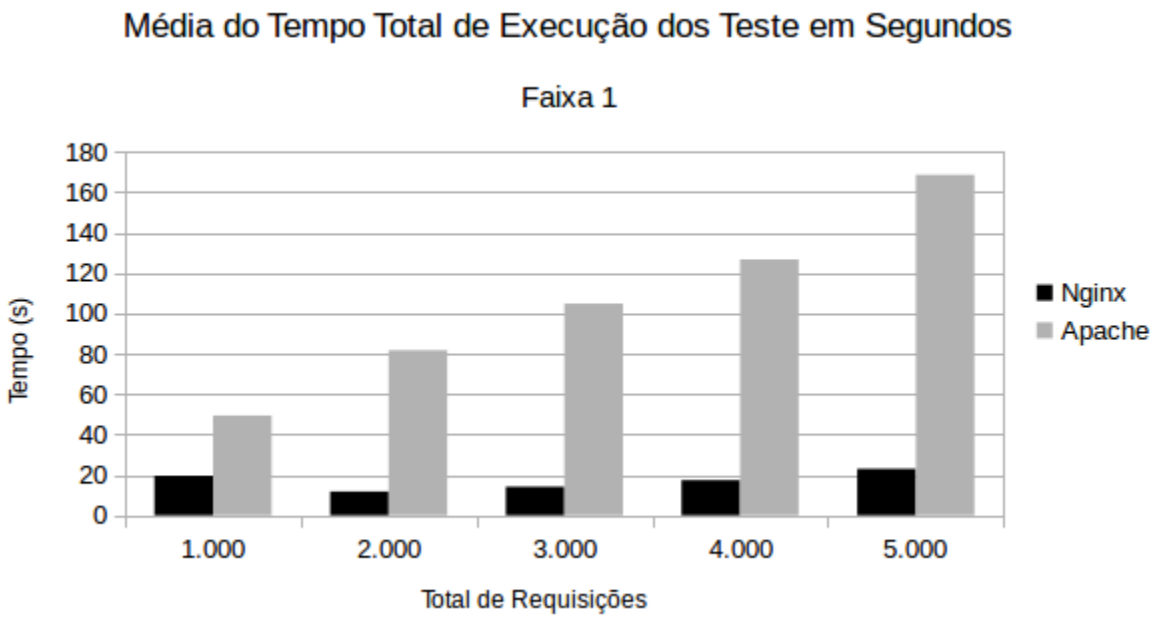

Figura 4 – Média do Tempo Total de Execução dos Testes - Faixa 1

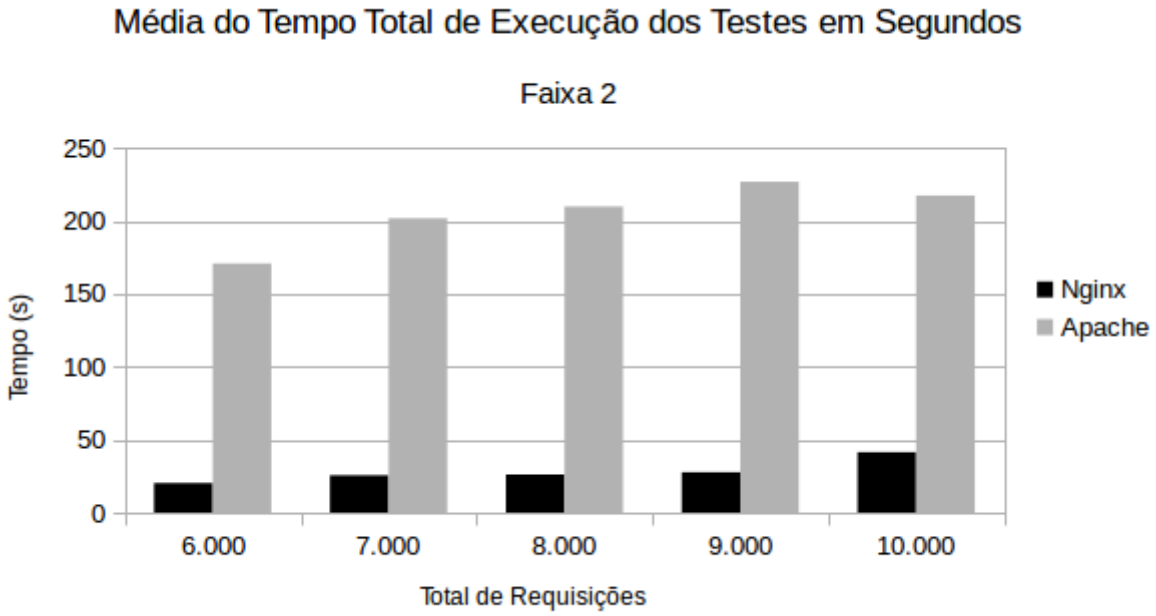

Figura 5 – Média do Tempo Total de Execução dos Testes - Faixa 2

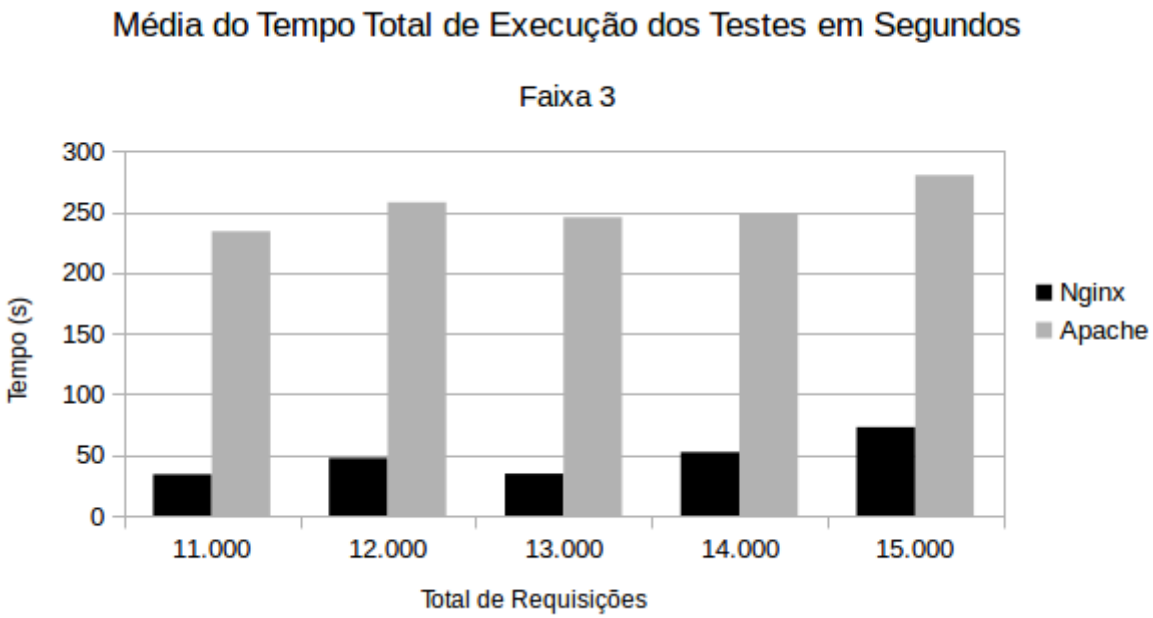

Figura 6 – Média do Tempo Total de Execução dos Testes - Faixa 3

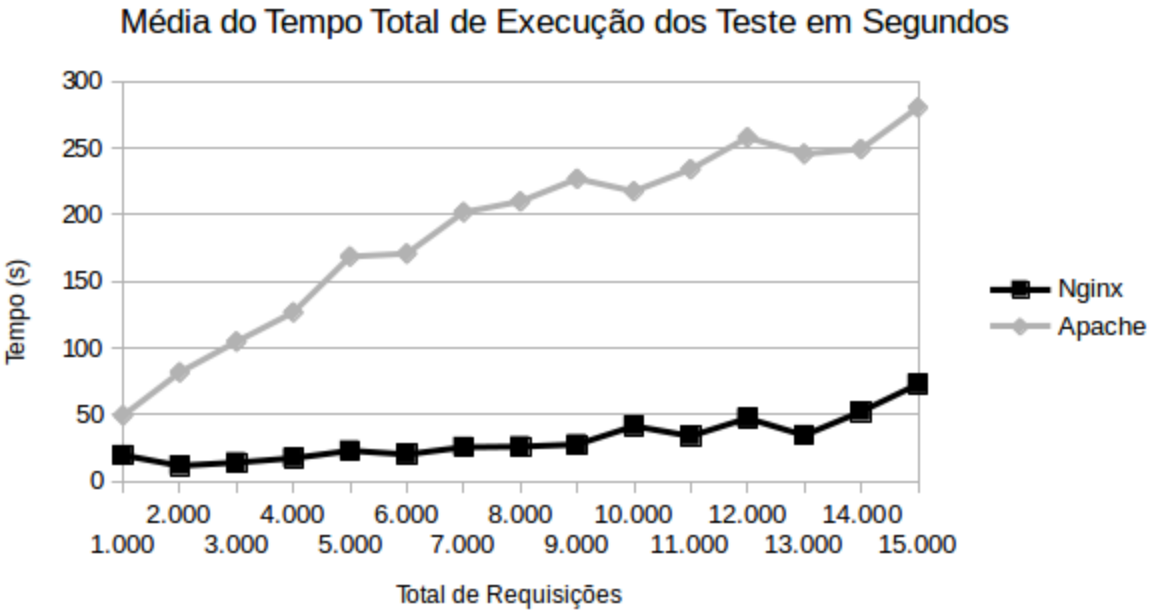

Figura 7 – Média do Tempo Total de Execução dos Testes

Ao analisar o gráfico e os dados da Faixa 1, é possível ver que os testes no servidor Apache demoraram, em média 6,31 vezes mais do que no servidor Nginx. Na Faixa 2, os

testes demoraram 7,57 vezes mais para serem executados no Apache. Na Faixa 3, os testes no Nginx demoraram, em média 5,61 vezes menos para serem executados.
Em termos gerais, levando em consideração os quinze valores de requisições totais, os tempos de execução dos testes no Apache foram em média 6,50 vezes mais lentos do que no Nginx. A soma dos tempos de totais dos testes foram: 467,19 segundos para o Nginx contra 2.824,64 segundos para o Apache.

5.2.2 Média do Total de Dados Transferido

O total de dados transferidos mostra a quantidade de informações que foram trocadas entre os computadores, incluindo os dados enviados pelas requisições e os dados retornados pelo servidor. Para fins de análise, os dados foram convertidos de bytes para Mega bytes. No caso desta métrica, quanto maior o valor observado, melhor.

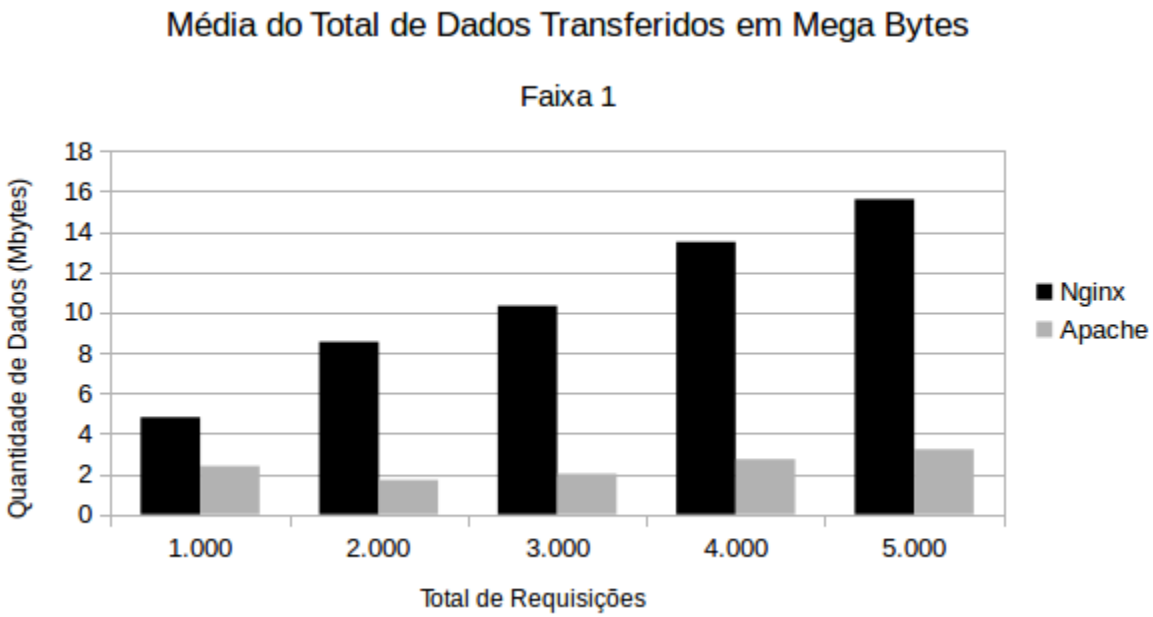

Figura 8 – Média do Total de Dados Transferido - Faixa 1

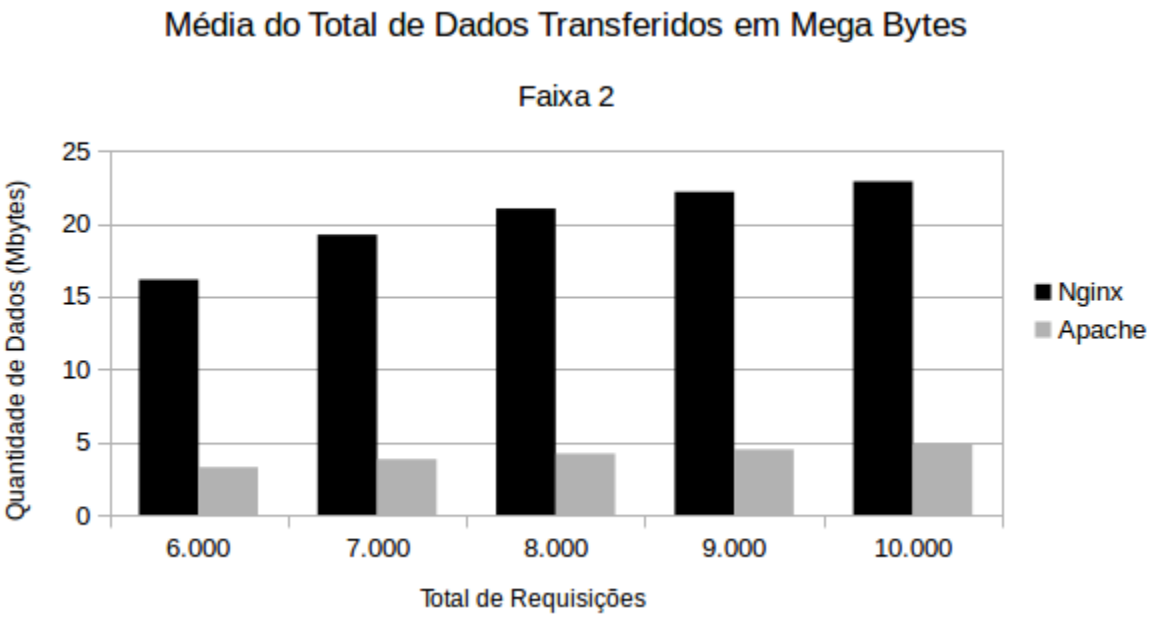

Figura 9 – Média do Total de Dados Transferido - Faixa 2

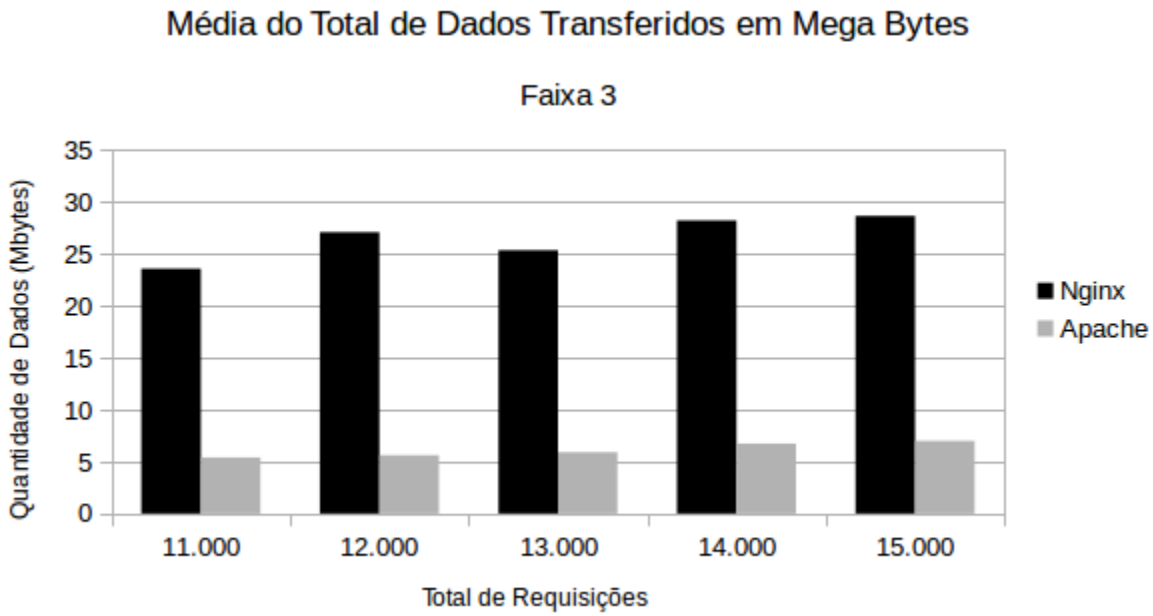

Figura 10 – Média do Total de Dados Transferido - Faixa 3

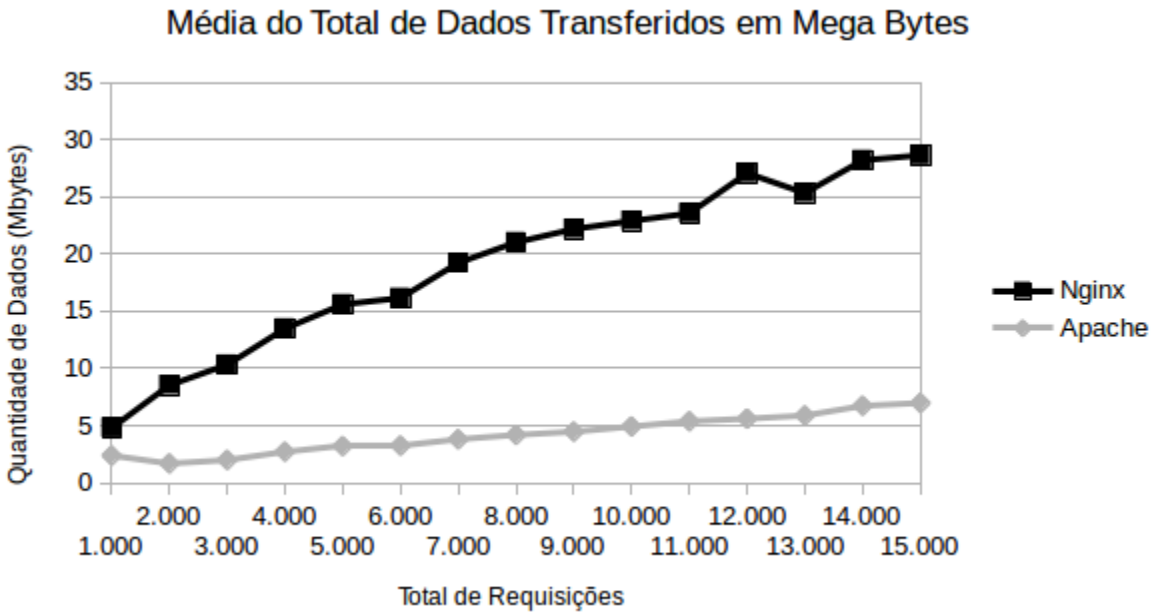

Figura 11 – Média do Total de Dados Transferido

No geral, nos testes com o Nginx foram trafegados 4,55 vezes mais dados comparado com o Apache. Nas faixa de 1.000 a 5.000 requisições, o Nginx conseguiu um desempenho 4,39 vezes superior em relação ao Apache. Nas outras faixas de requisições, 6.000 à 10.000 e 11.000 à 15.000 requisições, os resultados mostraram números semelhantes, apontando que o Nginx transferiu 4,91 e 4,35 vezes mais dados do que o Apache.

5.2.3 Média do Total de Texto HTML Transferido

Nessa indicador, será possível analisar a quantidade de páginas em HTML foram retornados pelos servidores nos teste. Os números obtidos nessa métrica influenciam diretamente os dados coletados na métrica analisada acima pelo fato de que, as páginas em HTML correspondem a maior parte do tráfego de dados entre o servidor e o cliente, respondendo por, em média, 90% dos dados. Para fins de análise, os dados foram convertidos de bytes para Mega bytes. No caso desta métrica, quanto maior o valor observado, melhor.

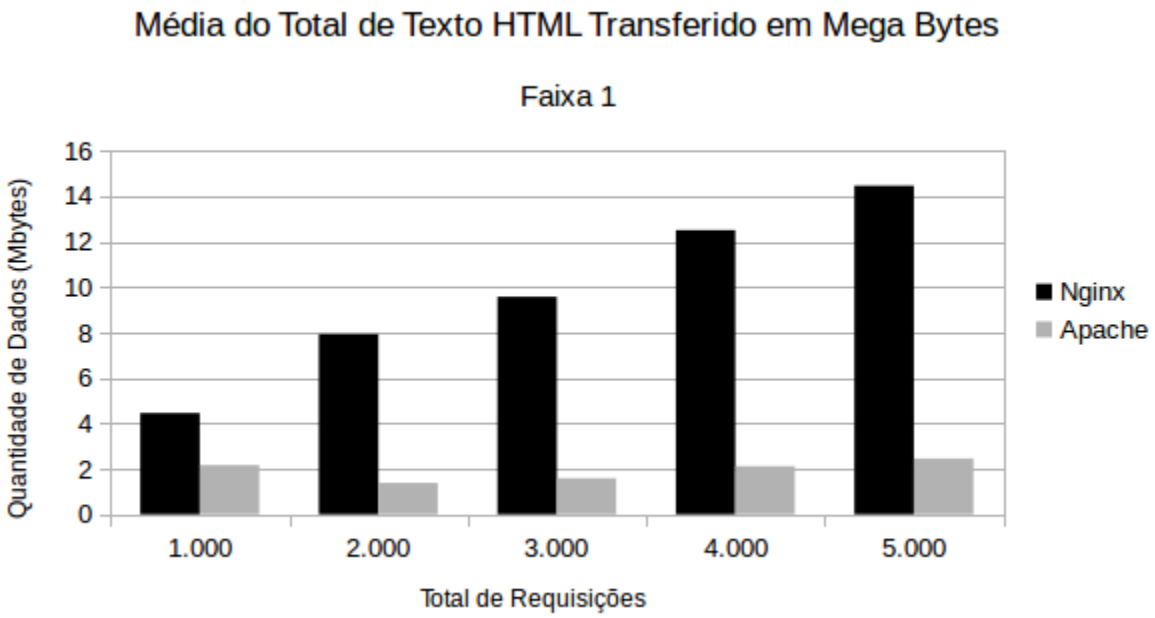

Figura 12 – Média do Total de Texto HTML Transferido - Faixa 1

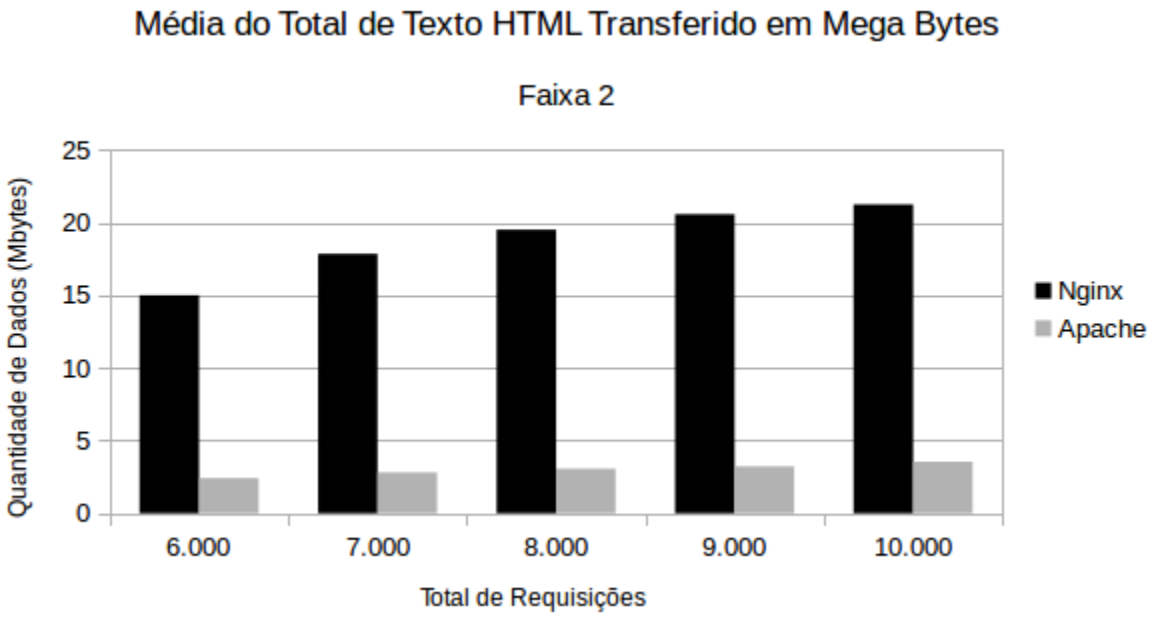

Figura 13 – Média do Total de Texto HTML Transferido - Faixa 2

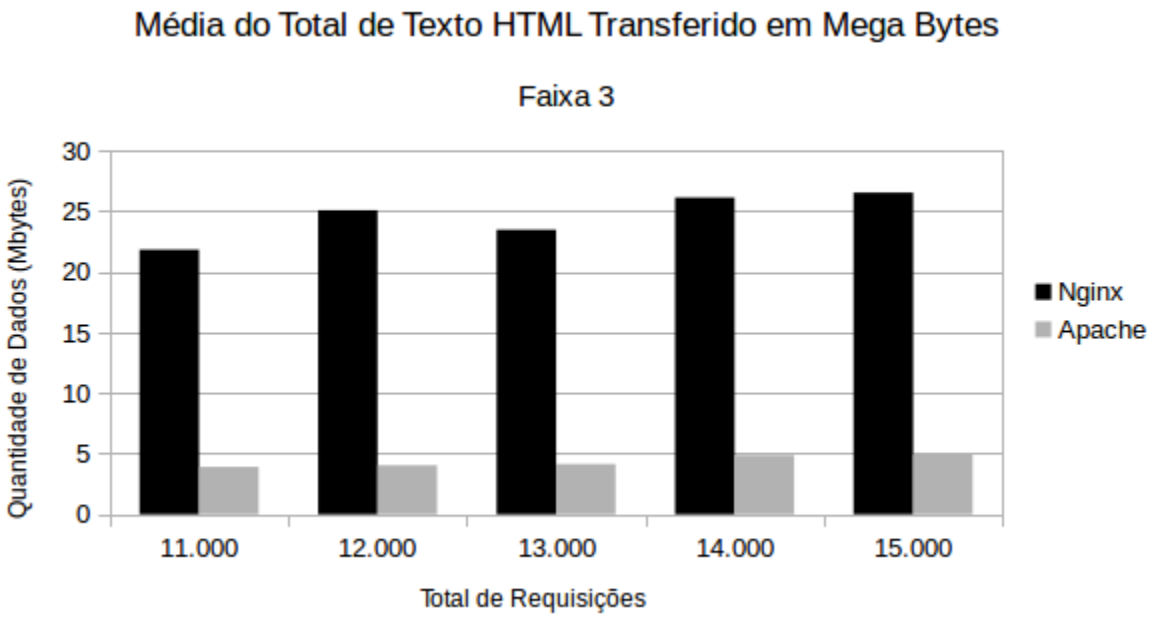

Figura 14 – Média do Total de Texto HTML Transferido - Faixa 3

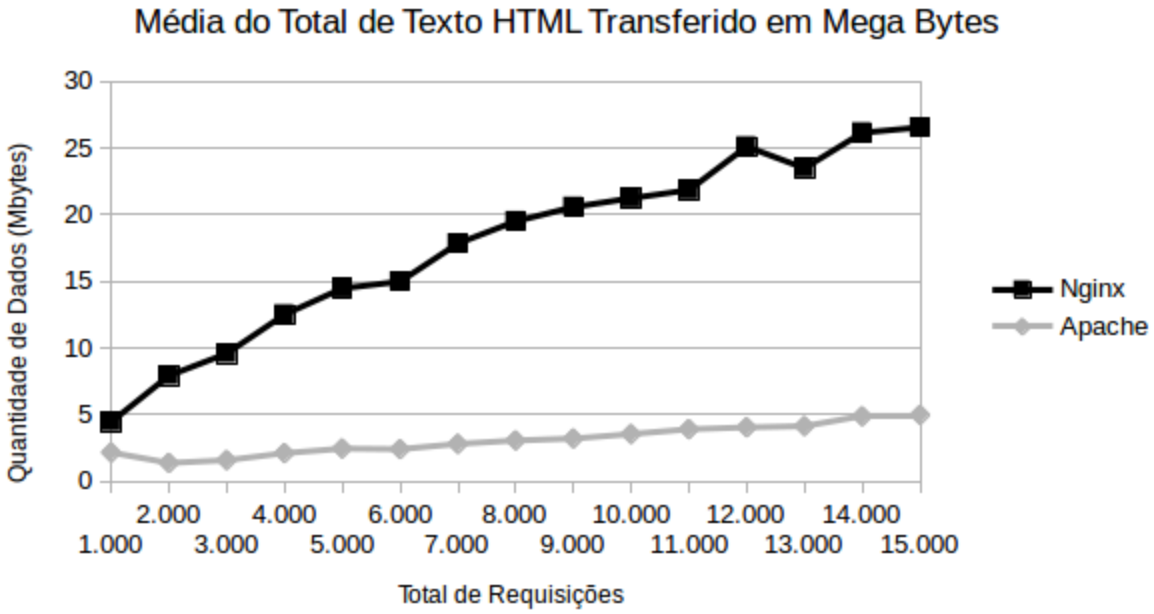

Figura 15 – Média do Total de Texto HTML Transferido

O Nginx entregou, em média, 5,67 vezes mais dados em HTML do que o Apache. Analisando as faixas de valores de requisição, o Nginx conseguiu entregar 5,12, 6,26 e 5,63 vezes mais dados em HTML do que o Apache.

5.2.4 Média do Número de Requisições Atendidas

Esse indicador é calculado de acordo com o tempo total gasto pela execução do teste e, como já falado, o tempo de execução do teste é uma métrica para auxiliar na análise do desempenho dos servidores. Assim sendo, o indicador de número do requisições atendidas também deve ser interpretado como um indicador auxiliar na análise de desempenho. No caso desta métrica, quanto maior o valor observado, melhor.

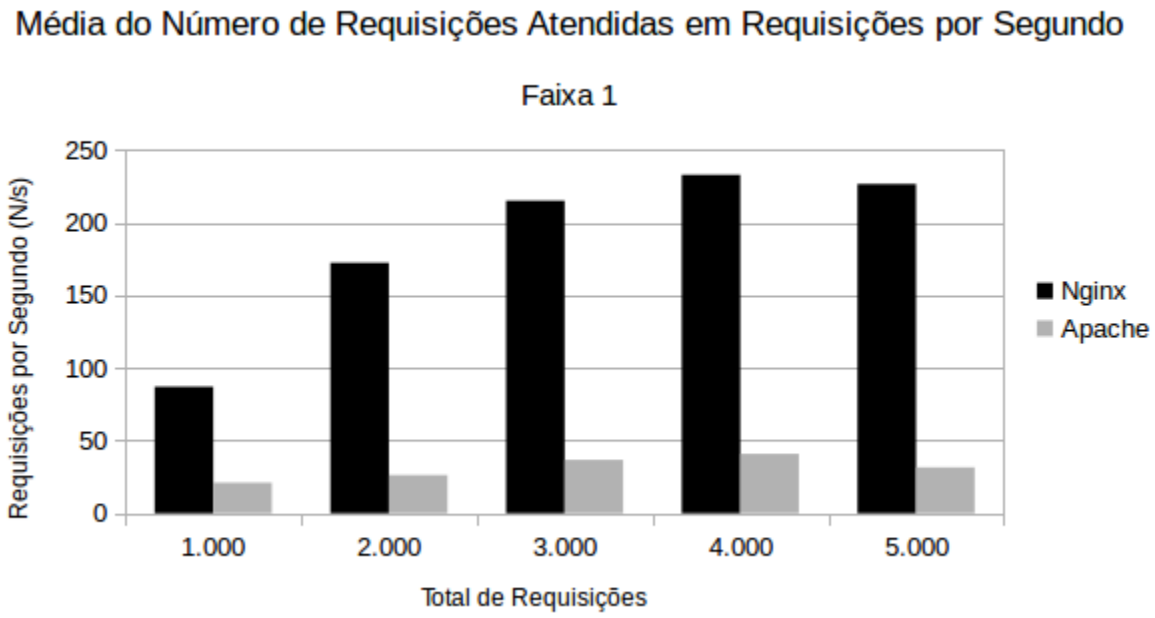

Figura 16 – Média do Número de Requisições Atendidas - Faixa 1

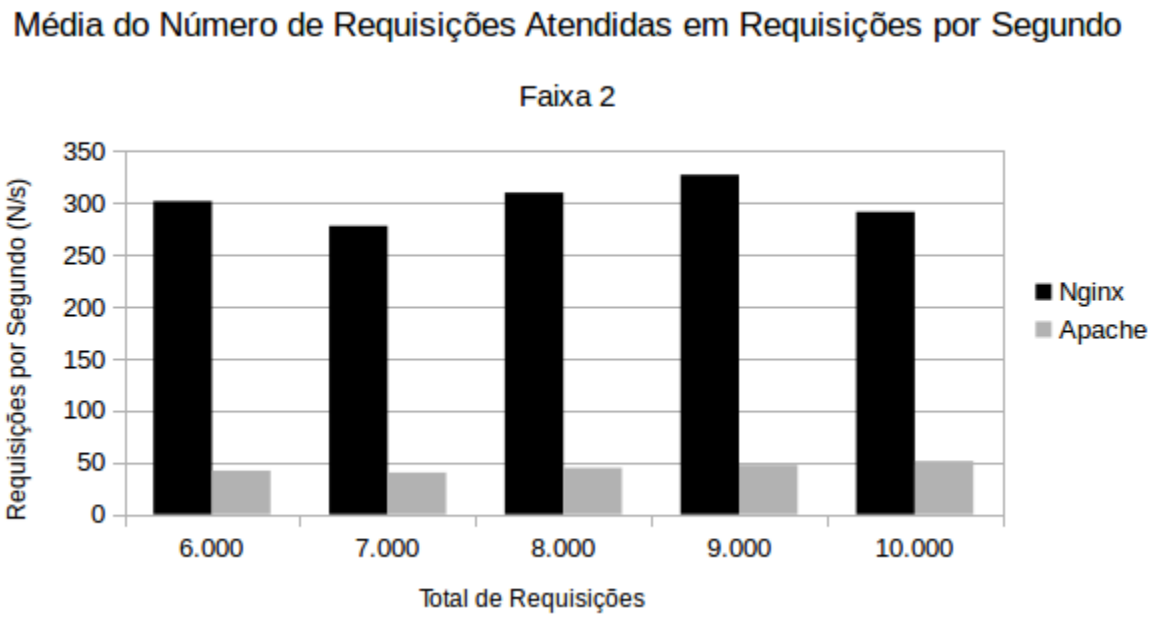

Figura 17 – Média do Número de Requisições Atendidas - Faixa 2

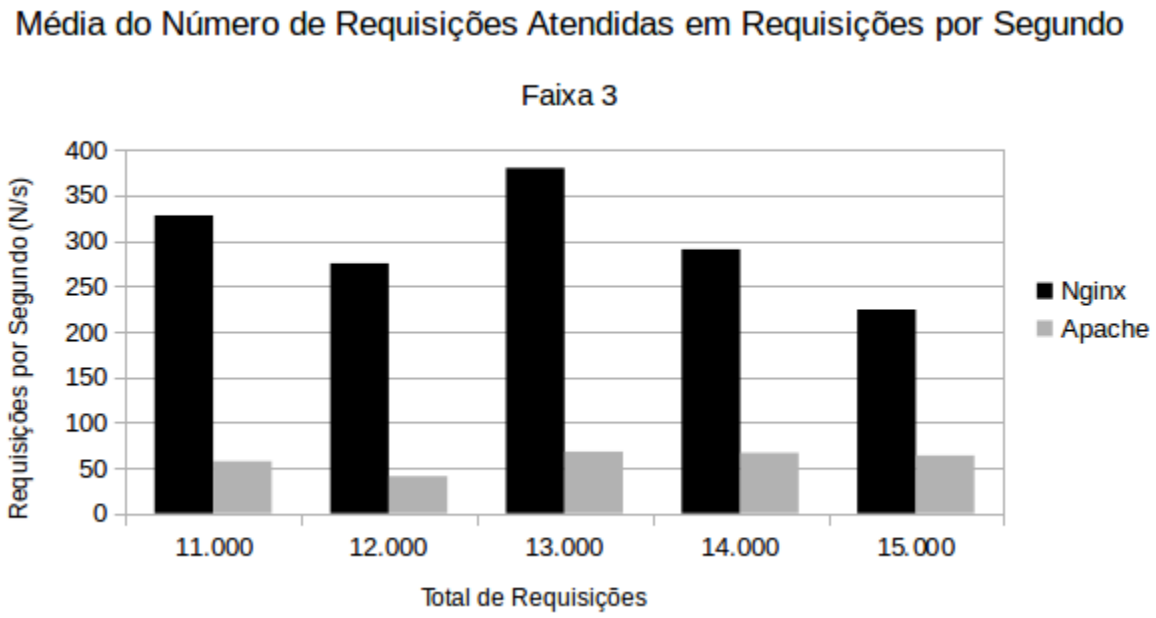

Figura 18 – Média do Número de Requisições Atendidas - Faixa 3

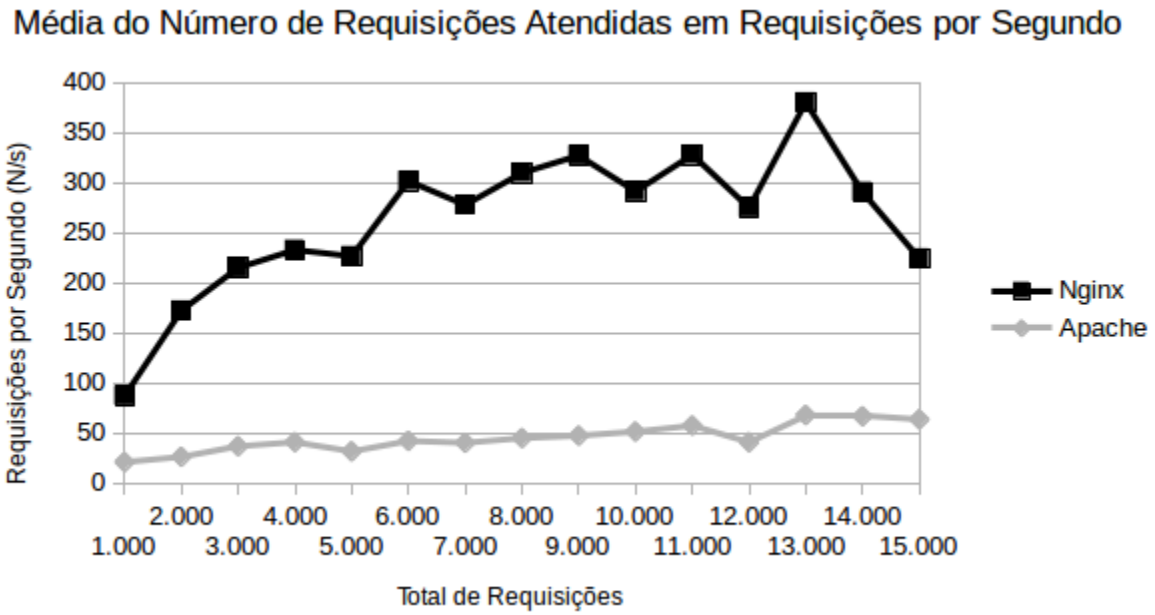

Figura 19 – Média do Número de Requisições Atendidas

Ao analisar os dados e os gráficos vemos que o Nginx, no geral, obteve uma taxa de requisições atendidas por segundo 5,92 vezes maior do que o Apache. Analisando os gráficos e os dados das faixas 1, 2 e 3, vemos que a taxa de requisições atendidas pelo Nginx é, respectivamente, 5,88, 6,71 e 5,18 vezes maior do que no Apache.

5.2.5 Média do Tempo de Resposta por Requisição Simultânea

Nessa métrica, é calculado o tempo de resposta para as requisições que foram feitas de forma simultânea. Como essa métrica mede o tempo entre a emissão da requisição e a resposta do servidor, é uma métrica mais acurada para analisar o desempenho dos servidores nos testes. No caso desta métrica, quanto menor o valor observado, melhor.

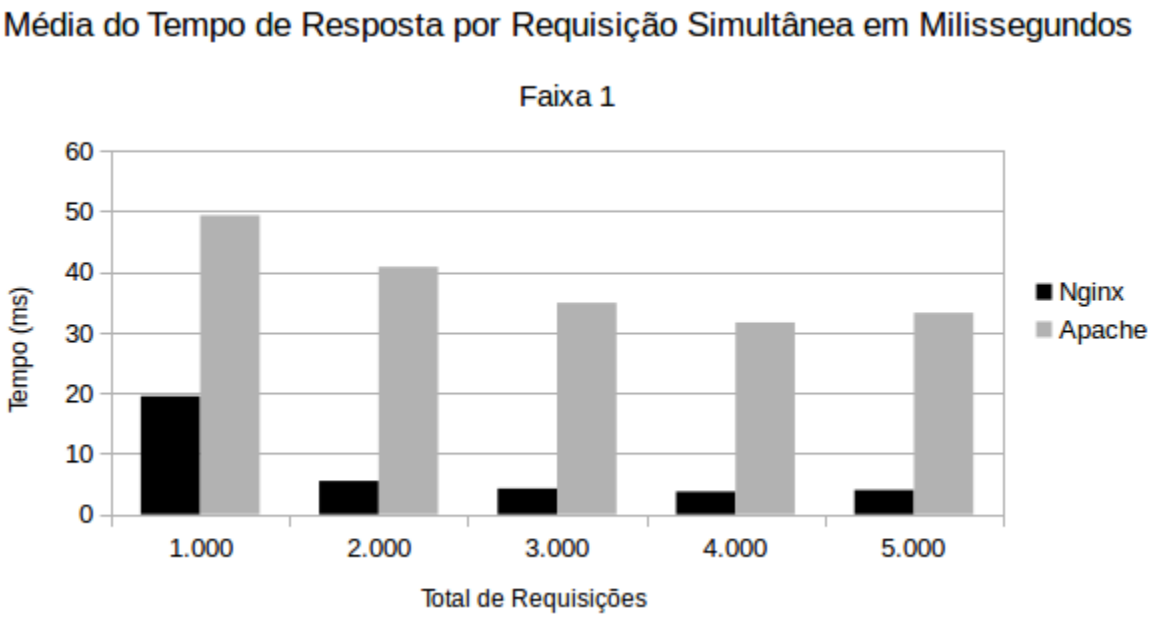

Figura 20 – Média do Tempo de Resposta por Requisição Simultânea - Faixa 1

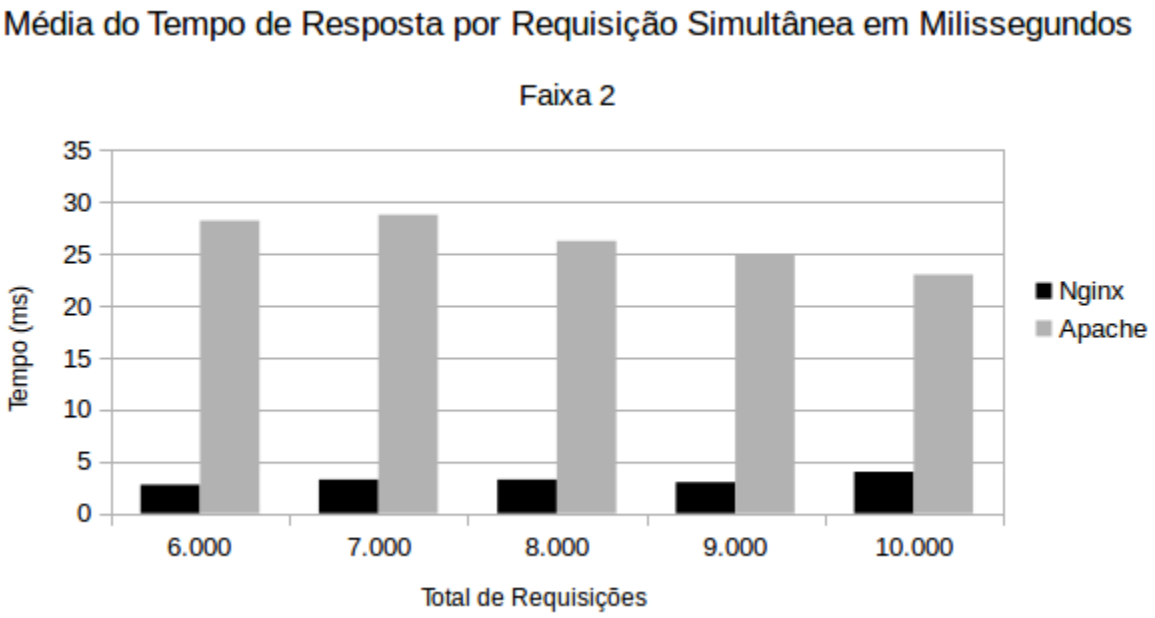

Figura 21 – Média do Tempo de Resposta por Requisição Simultânea - Faixa 2

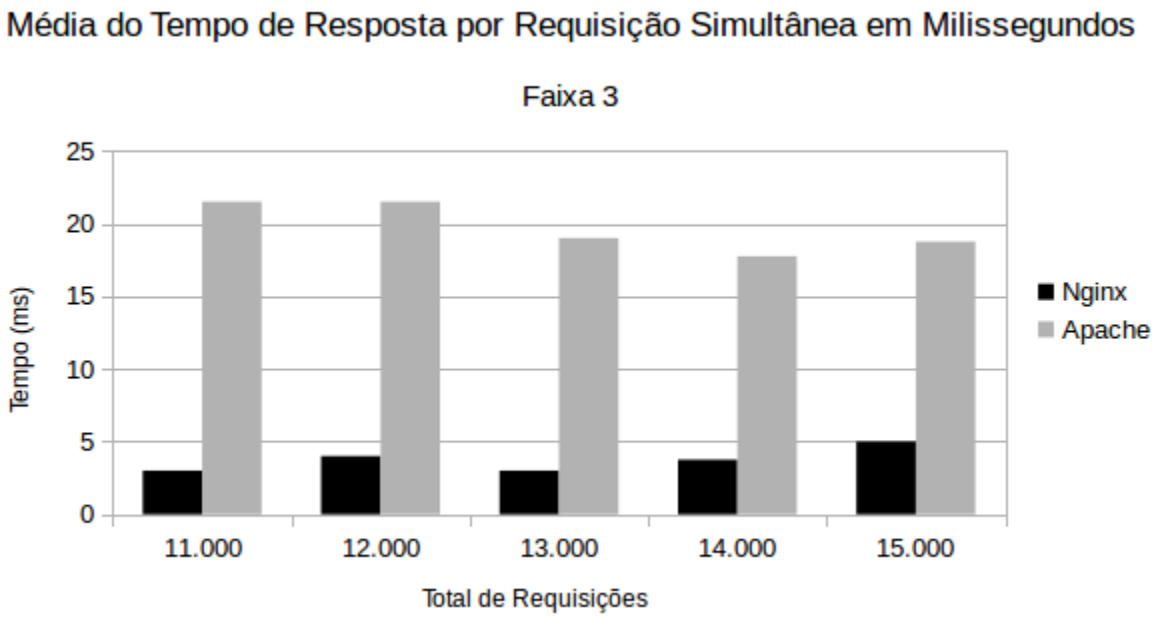

Figura 22 – Média do Tempo de Resposta por Requisição Simultânea - Faixa 3

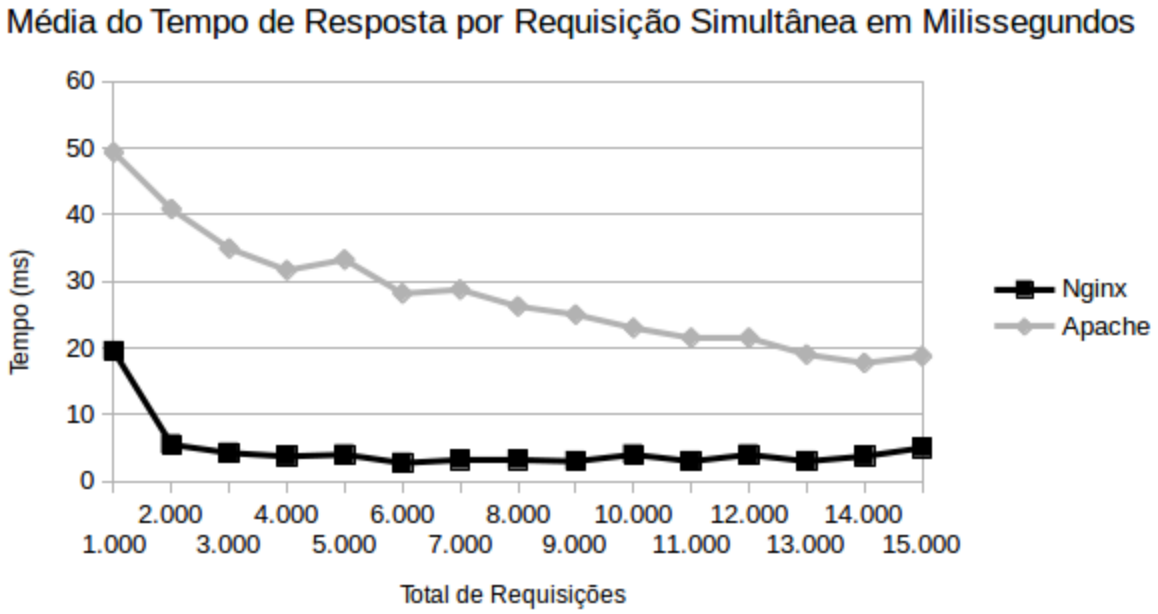

Figura 23 – Média do Tempo de Resposta por Requisição Simultânea

Nós gráficos é possível ver que os tempos de resposta nos testes com o servidor Apache foram superiores comparado com os do Nginx. Em números, analisando o desempenho nos quinze valores de requisições, o tempo de resposta nos testes com o servidor

Apache foi, em média, 6,90 vezes maior do que os observados nos teste com o Nginx. Ao analisar os dados separados pelas faixas de valores de requisições, em todos os casos o Nginx teve um desempenho superior, sendo que os valores observados nos testes com o Apache foram 6,98, 8,24 e 5,47 vezes maior do que os observados no Nginx, mostrado um tempo maior para que a requisição fosse atendida.

5.2.6 Média da Taxa de Transferência

Nesse indicador é medida a taxa de transferência dos dados entre cliente e servidor. No caso desta métrica, quanto maior o valor observado, melhor.

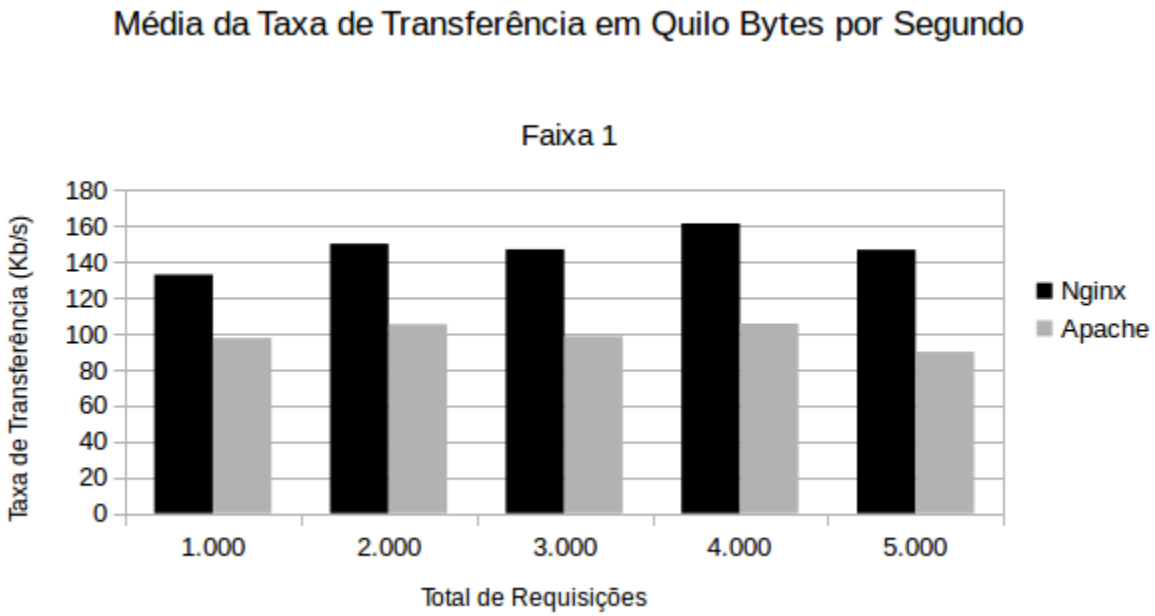

Figura 24 – Média da Taxa de Transferência - Faixa 1

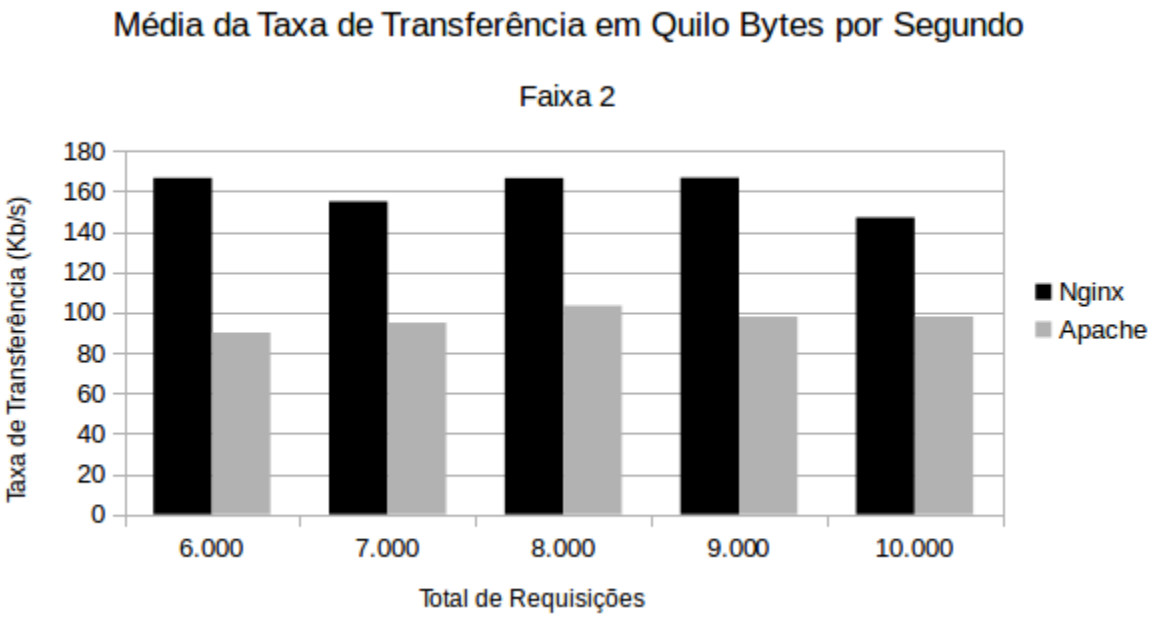

Figura 25 – Média da Taxa de Transferência - Faixa 2

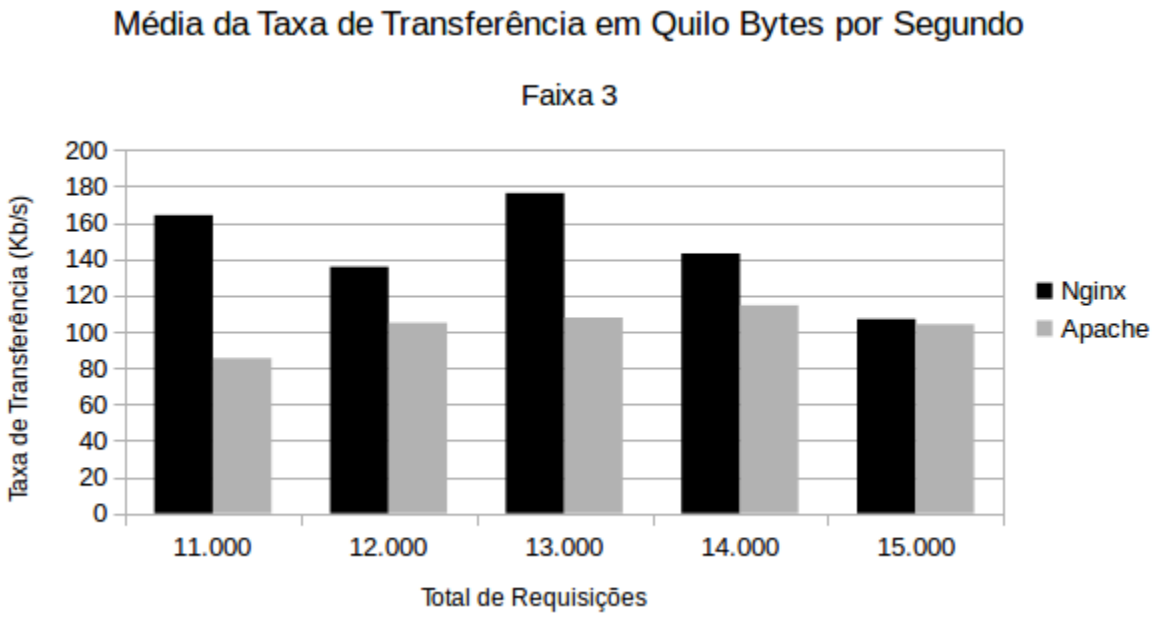

Figura 26 – Média da Taxa de Transferência - Faixa 3

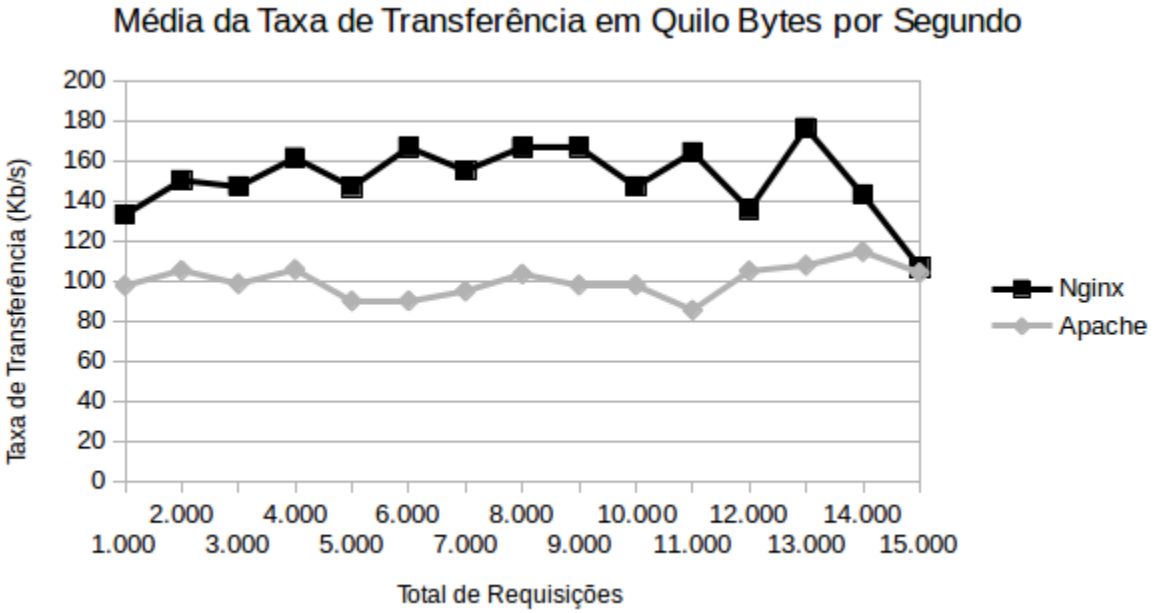

Figura 27 – Média da Taxa de Transferência

Nessa métrica, os dois servidores mostraram o desempenho menos discrepante entre as métricas analisadas. Entre os quinze valores de requisições testados, nos testes com o Nginx, foram observadas taxas de transferência, em média, 1,52 vezes maior do que os observados no Apache.

Analisando as três faixas de valores de requisição, as taxas de transferência observadas no Nginx foram, em média, 1,48, 1,43 e 1,42 maior do que os números observados no Apache.

6 Conclusão

Com base nos dados coletados e analisados, é possível dizer que o servidor HTTP Nginx conseguiu ser, em média, 5 vezes mais eficiente que o servidor HTTP Apache, dando uma margem de sobrevida ao SIGA da forma como o sistema é desenvolvido hoje, usando o *framework* Miolo. Porém, mesmo com a utilização de um servidor HTTP mais eficiente, é difícil dizer até quando o SIGA conseguirá escalar em relação a quantidade de usuários ativos.

Como dito na introdução, escalabilidade é um atributo desejável de um sistema. Ao meu ver, o *framework* Miolo não oferece um nível de escalabilidade suficientemente bom, sendo necessário investimento em novos servidores com desempenho superior ao que é utilizado atualmente, para que o sistema suporte uma grande demanda. O Miolo, ao "esconder" o código HTML do desenvolvedor com a intenção de facilitar e agilizar o desenvolvimento, acaba exigindo uma alta carga de processamento devido ao seu mecanismo de geração de páginas dinâmicas.

Além da substituição do servidor HTTP para o Nginx, o ideal seria a total substituição do SIGA, sendo desenvolvido do zero, podendo utilizar a mesma linguagem de programação (PHP) porém com um *framework* mais atual e em constante desenvolvimento, reduzindo a curva de aprendizado. Alguns dos *frameworks* PHP mais utilizados hoje são:

- Zend Framework
- CakePHP
- Code Igniter
- Laravel

Uma substituição mais radical envolveria a mudança até mesmo da linguagem de programação utilizada no desenvolvimento, sendo Ruby junto com o *framework* Ruby on Rails, e a linguagem Python junto com o *framework* Django, os mais recomendados, por serem utilizadas com sucesso por várias empresas, serem de código aberto (sem custo com licenças) e contam com uma grande comunidade de desenvolvedores.

6.1 Trabalhos futuros

A troca do servidor HTTP, do Apache para o Nginx, não é a única coisa que pode ser feita para que o SIGA não tenha o seu ciclo de vida encerrado em um futuro próximo. A atualização dos *softwares* utilizados pelo SIGA traria não somente melhoria

no desempenho, como também mais segurança. A versão do PHP utilizada no servidor do SIGA a 5.3.3, lançada em Junho de 2.009, sendo versão estável atual 5.6, lançada em Outubro de 2.014. A versão do *framework* Miolo utilizado no SIGA é a 2.0, sendo a versão estável atual 2.6.

A ferramenta de relatórios utilizada representa outro gargalo importante no desempenho do SIGA. O Jaspereport foi desenvolvido originalmente para ser utilizado com a linguagem Java, sendo que, para cada relatório gerado, uma máquina virtual Java é criada para cada relatório gerado, consumindo os recursos do servidor e degradando o desempenho geral do sistema. Uma alternativa para esse problema seria a adoção da biblioteca PHPJasperXML (www.simitgroup.com/?q=PHPJasperXML), que gera relatórios usando o próprio interpretador do PHP a partir da leitura dos arquivos XML utilizados pelo Jaspereport na criação de relatórios.

A utilização da *Hip-Hop Virtual Machine* - HHVM (www.hhvm.com), máquina virtual desenvolvida pelo Facebook para ser utilizada nos servidores da empresa e de código aberto, possibilita mais uma oportunidade de melhoria no desempenho de sistemas desenvolvidos em PHP, pois a ferramenta provê uma compilação em tempo real (*just-in-time*), preservando a dinamismo do desenvolvimento em PHP.

Referências

BONDI, A. B. Characteristics of scalability and their impact on performance. 2000.

CONSORTIUM, W. W. W. *Visão Geral do HTML5*: Manual. 2014. Disponível em: <http://www.w3c.br/cursos/html5/conteudo/capitulo1.html>. Acesso em: 17 Nov. 2014.

FASTCGI. *FastCGI: A High-Performance Web Server Interface*. 2014. Disponível em: <http://www.fastcgi.com/drupal/node/6?q=node/15>. Acesso em: 16 Nov. 2014.

PHP.NET. *PHP: What can PHP do?*: Manual. 2014. Disponível em: <http://php.net/manual/en/intro-whatcando.php>. Acesso em: 16 Nov. 2014.

PORTUGUêS, D. O. de. *Dicionário Online de Português*: Manual. 2014. Disponível em: <http://www.dicio.com.br>. Acesso em: 19 Nov. 2014.

POSTGRESQL. *About*. 2014. Disponível em: <http://www.postgresql.org/about/>. Acesso em: 17 Nov. 2014.

ROWE, W. *ANTURIS BLOG*: Nginx vs apache. 2014. Disponível em: <https://anturis.com/blog/nginx-vs-apache/>. Acesso em: 12 Nov. 2014.

STALLINGS, W. *Redes e sistemas de comunicação de dados: teoria e aplicações corporativas*. 5. ed. [S.l.]: Elsevier, 2005.