UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA
ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB

ANDERSON FERNANDO RIBEIRO DA SILVA

BALANCEAMENTO DE CARGA EM APLICAÇÕES WEB E REPLICAÇÃO DE DADOS COM POSTGRESQL.

MONOGRAFIA DE ESPECIALIZAÇÃO

LONDRINA
2015

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA
ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB

ANDERSON FERNANDO RIBEIRO DA SILVA

BALANCEAMENTO DE CARGA EM APLICAÇÕES WEB E REPLICAÇÃO DE DADOS COM POSTGRESQL.

Trabalho de Conclusão de Curso apresentado a Diretoria de Pesquisa e Pós-Graduação da Universidade Tecnológica Federal do Paraná do Câmpus Londrina, como requisito parcial à obtenção do grau de especialista em Desenvolvimento Web.

Orientador: Prof. Esp. Frederico F. Siena

LONDRINA
2015

TERMO DE APROVAÇÃO

Título da Monografia

BALANCEAMENTO DE CARGA EM APLICAÇÕES WEB E REPLICAÇÃO DE DADOS COM POSTGRESQL

por

Anderson Fernando Ribeiro da Silva

Esta monografia foi apresentada às 10h10 do dia **14 de novembro de 2015** como requisito parcial para a obtenção do título de ESPECIALISTA EM DESENVOLVIMENTO WEB. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho APROVADO.

_____
Prof. Esp. Frederico de Figueiredo Siena
(UTFPR)

_____
Prof. Dr. Alessandro Botelho Bovo
(UTFPR)

_____
Prof. Me. Eidy Leandro Tanaka Guandeline
(UTFPR)

Visto da coordenação:

_____
Prof. Me. Thiago Prado de Campos
Coordenador da esp. em Desenvolvimento Web

_____
Prof. Me. José Luis Dalto
Coordenador de Pós-Graduação Lato Senso

A Folha de Aprovação preenchida e assinada encontra-se na Coordenação do Curso

DEDICATÓRIA

Primeiro a você Pai, a você, que as saudades ainda são grandes, marcam a falta física de um sorriso, um afago, uma palavra. Foi, contudo, no meio dessa ausência que compreendemos a dimensão do ser eterno, da porção que fica e que, apesar de invisível, é sensível. Esse trabalho como fruto do seu ensino é dedicado especialmente a você, ainda que não esteja do meu lado para comemorar essa conquista, sempre farei meu melhor para orgulhar seu nome.

Um agradecimento especial a minha querida Mãe ***Marineuza Reis.*** Agora com responsabilidades ainda maiores, lhe admiro cada vez, mais e mais, obrigado por ser essa pessoa fantástica. As minhas 3 irmãs da qual não consigo ficar longe, ***Emanuelly Ribeiro, Ingrid Ribeiro, Lorraine Ribeiro*** obrigado por serem pessoas positivas e me animarem em momentos tristes.

À Deus, dedico o meu agradecimento maior, porque têm sido tudo em minha vida.

Agradeço meu grande irmão ***Guilherme Rocha*** que esteve do meu lado nos momentos mais difíceis da minha vida, posso dizer que você é o cara!

Agradeço a toda equipe **Finer** em especial ao meu sócio ***Tiago Alves*** por me apoiar e me ajudar a fazer a diferença em cada projeto, cada ideia, e em cada conquista.

E com muito amor a minha Namorada ***Renata Lais Delezuk***, pessoa fantástica, que me ajuda a fazer a diferença todos os dias. Obrigado Amor por fazer da minha vida um espetáculo no palco da existência.

A todos vocês, meu muito obrigado.

Anderson Fernando Ribeiro da Silva

RESUMO

SILVA, Anderson Fernando Ribeiro. **BALANCEAMENTO DE CARGA EM APLICAÇÕES WEB E REPLICAÇÃO DE DADOS COM POSTGRESQL.**. 2015. 40f. Monografia (Especialização em Desenvolvimento Web) – Programa de Pós-Graduação em Tecnologia, Universidade Tecnológica Federal do Paraná. Londrina, 2015.

Com a expansão da internet e o aumento de novos serviços ofertados, a adesão por novos usuários que utilizam os mais diversos dispositivos de acesso, obrigaram os desenvolvedores e provedores de soluções repensarem a maneira e as técnicas empregadas em seus produtos, visando contemplar essa miscelânea e concomitantemente manter a qualidade. Esta demanda aumenta drasticamente o emprego de novos equipamentos de alta performance e banda de acesso à internet, levando o mercado optar entre otimizar os serviços com os recursos já disponíveis ou investir na atualização ou novas aquisições em seus *data centers*. Este trabalho tem como objetivo principal comprovar que é possível através do emprego de diversas técnicas de tunning e alta disponibilidade, aumentar a performance dos atuais serviços de um dado cenário com a implementação de diversas novas características de segurança e demonstrar que em alguns casos com pequenos ajustes de hardware, é possível até suprimir equipamentos, desta maneira, diminuir custos com infraestrutura.

Palavras-Chave: Alta Disponibilidade, Replicação de Dados, balanceamento de carga.

ABSTRACT

SILVA, Anderson Fernando Ribeiro. **Load balancing web applications and data replication with postgresql**. 2015. 40p . Monograph (Specialization in Web Development) - Programa de Pós-Graduação em Tecnologia, Universidade Tecnológica Federal do Paraná. Londrina, 2015.

With the expansion of the Internet and the rise of new services offered, membership for new users using the various access devices, forced the developers and solutions providers to rethink the way and the techniques used in their products, in order to contemplate this and miscellany concurrently maintain quality. This demand dramatically increases the use of new high-performance equipment and bandwidth internet access, which forced the market choose to optimize services with the resources already available or invest in updating or acquisition of their data centers. This paper aims to prove that it is possible by employing various techniques of tuning and high availability, increased performance of current services of a given scenario with the implementation of several new safety features and demonstrate that in some cases with minor adjustments hardware, it is possible to suppress equipment, thereby, reduce infrastructure costs.

Keywords: High Availability, Data Replication, Load Balancing.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

GNU	Sistema operacional tipo unix.
SGBD	Sistema de gerenciamento de banco de dados.
WWW	World wibe web.
PGSQL	Uma linguagem procedural carregável desenvolvida para o sistema de banco de dados PostgreSQL.
SQL	Linguagem de Consulta Estruturada.
CRUD	Acrónimo de Create, Read, Update e Delete na língua Inglesa.
TCP	Protocolo de controle de transmissão.
IP	*Protocolo de Internet é uma identificação de um dispositivo (computador, impressora, etc).*
TCP/IP	Protocolos de comunicação entre computadores em rede.
AWS	*Amazon Web Services.*

SUMÁRIO

1 INTRODUÇÃO

A internet é hoje uma das maiores ferramentas de comunicação do mundo, a maneira mais comum de acesso é por meio da World Wide Web (www). Através da internet o usuário pode se conectar a qualquer lugar do mundo, conhecer lugares por imagem, acesso a sites de notícias, conversar em bate-papos, dentre diversas outras coisas que são possíveis de se fazer na internet.

A cada ano que se passa aumenta o volume de informações em toda a rede, uma prova deste aumento é a progressão em serviços de busca (STATISTIC BRAIN, 2015). Analisando um período de sete anos, observar-se que no ano 2008 foram realizadas 1.745.000.000 (um bilhão e setecentos e quarenta e cinco milhões) de buscas por dia, já no ano de 2014 foram realizadas 5.740.000.000 (cinco bilhões e setecentos e quarenta milhões) de buscas por dia, isso significa um aumento aproximado de 328.93%, assim demonstrando o crescimento da *www*.

Com esse grande volume de serviços que são disponibilizados na internet, pode-se montar um cenário dos quais empresas e organizações precisam garantir a disponibilidade do seu serviço web, seja ele gratuito ou pago, com isso surge a necessidade de performance em aplicações web.

Neste trabalho serão apresentados os roteiros para a construção de um cenário de alta disponibilidade e balanceamento de carga, utilizando como servidor web o apache e o nginx, e como servidor de cache o Varnish para garantir maior performance em páginas estáticas além do SGBD PostgreSQL.

Será abordado como implementar um cenário composto por um ambiente com os servidores web apache e nginx, além do varnish para o armazenamento em cache das solicitações dos objetos requisitados, desta maneira garantindo maior performance para conteúdos estáticos. A solução de banco de dados escolhida foi o PostgreSQL em sua versão 9.4. Como citado pelo SMANIOTO o PostgreSQL tem maior capacidade para trabalhar com grande volume de dados, por isso sua escolha com SGBD principal. Todas as ferramentas utilizadas e apresentadas serão open source ou e m algum outro formato de distribuição compatível com as premissas básicas do projeto GNU[1].

Este trabalho é motivado a desenvolver uma solução que melhore o desempenho de um servidor com algumas técnicas de tuning e uso de aplicações em conjunto.

O maior impulso para realizar esse trabalho foi pela situação atual da empresa em que sou colaborador. Hoje temos uma arquitetura de servidores em nuvem utilizando serviços como o apache e o PostgreSQL de maneira nativa, ou seja, em seu formato padrão sem nenhuma melhoria de configuração para exploração da performance dos serviços. A minha justificativa para o trabalho é melhorar esse cenário com desempenho, segurança e disponibilidade para uma maior economia interna em hardware para a empresa.

Levando-se em consideração um serviço que aumenta diariamente a sua taxa de acessos e demanda no servidor, por exemplo um E-commerce, um sistema ou

1 https://www.gnu.org/philosophy/free-sw.html

até mesmo um site, dentre outros serviços, podem surgir problemas como:

- Sobrecarga no servidor;
- Limite físico da máquina;
- Alto custo com hardware;
- Diminuição de desempenho do serviço, deixando-o lento;
- Indisponibilidade.

Com o balanceamento de carga é possível obter um aumento de performance bastante satisfatório e significativo, obtendo-se uma melhora na entrega do conteúdo aos usuários finais com pouca ou nenhuma alteração no hardware utilizado. Sendo assim o trabalho é justificado por seu fim didático e por apresentar uma melhoria em serviços web, demonstrando de uma maneira simples e objetiva a implementação de um servidor web com baixo custo operacional.

É possível destacar como objetivos principais:

- Configurar apache para responder páginas que o nginx não consegue interpretar.
- Configurar o Varnish para fazer cache de páginas estáticas.
- Configurar PostgreSQL para fazer replicação *master-slave*.
- Testar as configurações do cenário proposto.

A coleta de dados para realização deste trabalho foi feita através de artigos, documentações e livros, também foi utilizado o software PGADMIN para testar as implementações realizadas no PostgreSQL.

2 REFERÊNCIAL TEÓRICO

Neste capítulo contém a base teórica para compreensão do trabalho, está dividido em seções que engloba desde o termo apache até a diferença entre conteúdos estáticos e dinâmicos.

2.1 APACHE

O servidor apache é responsável e capaz de executar códigos PHP, Shell Script e Perl. Também pode ser utilizado como servidor HTTP entre outros. Ele é mais popular combinado com a linguagem de programação PHP e o SGBD Mysql. (ABREU, 2012)

O Apache HTTP Server Project é um esforço para desenvolver e manter um servidor HTTP de código aberto para sistemas operacionais modernos, incluindo UNIX e Windows NT. O objetivo deste projeto é fornecer um servidor seguro, eficiente e extensível que fornece serviços HTTP em sincronia com os padrões HTTP atuais.(APACHE).

2.2 NGINX

O servidor Nginx foi desenvolvido por um russo chamado Igor Sysoev no ano de 2002. Alguns anos depois, ele fez sua estreia nas formas de servidor HTTP e servidor proxy. O Nginx ganhou popularidade por usar o método EDA (Event-Driven) ou seja, capaz de lidar com um maior número de clientes com menos recursos. (Reis, 2015)

Nginx (pronuncia-se "motor-x") é um servidor open source proxy inverso para HTTP, HTTPS, SMTP, POP3, IMAP e protocolos, bem como um balanceador de carga, cache de HTTP, e um servidor web (servidor de origem). O projeto nginx começou com um forte foco em alta concorrência, alto desempenho e baixo uso de memória. (NGINX DOCUMENTATION, 2015).

2.3 VARNISH

Proxy HTTP reverso, eficiente por armazenar todo o conteúdo HTTP requisitado, fazendo com que o servidor não precise consultar por várias vezes o mesmo conteúdo e muitas vezes de maneira simultânea. O varnish busca em cache o objeto da requisição e o retorna ao cliente final de uma maneira muito ágil. (DIAS, 2015).

Varnish Cache é um acelerador de aplicações web também conhecido como HTTP caching proxy reverso. Varnish Cache é muito, muito rápido. Ele normalmente acelera a entrega com um fator de 300 - 1000x, dependendo de sua arquitetura. (VARNISH, 2015).

2.4 POSTGRESQL

O projeto surgiu em 1995 que foi derivado de outro projeto que teve seu início em 1976. PostgreSQL é um dos bancos de dados livres mais avançados do mundo. (4LINUX).

O PostgreSQL (conhecido anteriormente como Postgres95) derivou do projeto POSTGRES da universidade de Berkley, cuja última versão foi a 4.2. O POSTGRES foi originalmente patrocinado pelo DARPA (Agência de Projetos de Pesquisa Avançada para Defesa), ARO (Departamento de Pesquisa Militar), NSF (Fundação Cinetífica Nacional) e ESL Inc. (POESTGRESQL, 2015).

2.5 BALANCEAMENTO DE CARGA

Balanceamento de carga é a técnica de distribuir a carga de trabalho de maneira uniforme entre computadores, a fim de otimizar recursos e garantir maior desempenho, e assim tornando altamente disponíveis e adaptáveis. (TECH NET, 2015).

2.6 REPLICAÇÃO

A replicação de dados é basicamente a cópia de informações de um ou mais bancos de dados para uma outra estrutura de dados semelhante.(SMANIOTTO, 2015). Pode se dizer também que replicação é um meio de se copiar (replicar) de forma gerenciada os dados entre servidores. (RAMOS, 2015).

2.7 CONTEÚDO ESTÁTICO

O conteúdo estático são páginas que não podem ser alteradas através do navegador, basicamente, são informações que não precisam ser alteradas com frequência. Todo o conteúdo estático é mais econômico que o conteúdo dinâmico (LANDEMAINE, 2015).

2.8 CONTEÚDO DINÂMICO

Conteúdo dinâmico é aquele conteúdo que o navegador envia uma solicitação para o servidor web e o servidor responde com uma página solicitada de maneira exclusiva para o usuário (CULTURA MIX, 2015).

3 CENÁRIO PROPOSTO PARA BALANCEAMENTO DE CARGA

O cenário proposto se divide em duas partes, balanceamento de carga, e replicação de dados.

Os principais responsáveis pelo balanceamento de carga serão o nginx e apache, ambos são serviços de alta performance e robustez mas cada qual com suas características e vantagens.

Como citado anteriormente o principal objetivo desse trabalho é unir o que ambos os serviços têm de melhor.

O varnish é um proxy HTTP reverso, pode ser chamado também de acelerador HTTP. A sua principal função é armazenar arquivos na memória, permitindo que eles sejam servidos de uma maneira mais rápida. O varnish é projetado para hardware e sistemas operacionais modernos. O principal foco é resolver problemas reais, otimizar hardware e garantir desempenho e flexibilidade. (HEEN, 2015)

Como a proposta do presente trabalho também contempla redundância na persistência de dados, o PostgreSQL será configurado em modo *replication* distribuindo igualmente as operações entre suas réplicas.

Optou-se para a implementação do cenário os serviços AWS da Amazon[2], estando umas das instâncias[3] em um ponto de presença em São Paulo (Brasil) e outro em Nova York (EUA), aumentando desta maneira mais ainda o nível de redundância.

2 http://aws.amazon.com/pt/

3 Máquinas virtuais

3.1 ORÇAMENTO E CONFIGURAÇÃO DOS SERVIDORES

Neste capítulo será possível analisar as vantagens em se ter um servidor na AWS do que se ter uma estrutura física de servidores, será possível analisar custos e benefícios de ambos os cenários.

3.1.1 ORÇAMENTO

1º CENÁRIO - CUSTOS SERVIDOR AWS NO PRIMEIRO ANO			
NOME	**CONFIGURAÇÃO**	**VALOR MÊS**	**VALOR ANUAL**
AWS	512 MB 2 Mb/s 2 vCPU 50 GB	R$ 79,80	R$ 957,60
AWS	512 MB 2 Mb/s 2 vCPU 50 GB	R$ 79,80	R$ 957,60
	TOTAL	**R$ 159,60**	**R$ 1 915,20**

2º CENÁRIO - CUSTOS DE HARDWARE NO PRIMEIRO ANO SERVIDOR FISICO			
NOME	**CONFIGURAÇÃO**	**VALOR MÊS**	**VALOR ANUAL**
HARDWARE	DELL R220 Pentium G3450 3.1Ghz 4Mb 500GB	R$ 382,42	R$ 4 589,00
HARDWARE	Load Balance HP Quad Core 3.1Ghz 8MB 500GB	R$ 200,97	R$ 2 411,65
HARDWARE	No-Break 1400VA Biv/110V	R$ 49,16	R$ 589,90
HARDWARE	Lenovo Nas Storage ix4-300d 8TB	R$ 362,49	R$ 4 349,90
HARDWARE	Switch 24 Portas HP V1410-24G - 10/100/1000	R$ 96,91	R$ 1 162,90
HARDWARE	Rack Para Servidor 42us X 600mm	R$ 148,75	R$ 1 785,00
	TOTAL	**R$ 1 240,70**	**R$ 14 888,35**

2º CENÁRIO - MANUTENÇÃO MENSAL SERVIDOR FISICO			
NOME	**CONFIGURAÇÃO**	**VALOR MÊS**	**VALOR ANUAL**
COPEL	5 MB Dedicado + IP Fixo – Plano Corporativo	R$ 840,00	R$ 2 411,65
GVT	150MB + IP FIXO Plano Corporativo	R$ 249,90	R$ 2 998,80
MANUTENÇÃO	Contrato Manutenção Servidor	R$ 540,00	R$ 6 480,00
COPEL	Energia	R$ 230,00	R$ 2 760,00
ALUGUEL	Espaço Fisico	R$ 630,00	R$ 7 560,00
	TOTAL	**R$ 2 489,90**	**R$ 22 210,45**

Ilustração 2: Orçamentos anuais e mensais de cenários.

Na ilustração 2 é possível analisar de maneira claro o quanto um servidor virtual terceirizado gerará de custos fixos anual X o custo fixo de um servidor fisico anual.

Como é possível ver na ilustração o custo anual que a AWS gerará é de R$ 1.915,20 (um mil novecentos e quinze reais e vinte centavos), para duas instâncias com as seguintes configurações 512MB de RAM, processador de 2vCPU 1.4GHz, HD de 50GB, ficando com custo mensal de R$ 159,60 (cento e cinquenta e nove reais e sessenta centavos), ou seja, é um custo mensal relativamente baixo em virtude das garantias que AWS fornece, redundância de link no datacenter já é uma

das grandes vantagens em manter o servidor fora de uma estrutura própria.

Também é possível ver na ilustração 2 que os custos para montar e se ter uma estrutura interna de servidor é muito maior que as opções virtualizadas terceirizadas, como citada a AWS. Para elaboração dos valores acima foi previsto o mínimo de hardware para funcionar e operar o servidor da mesma maneira que funcionaria um servidor na AWS.

Tendo como cenário uma empresa que não possui infraestrutura interna, pode se observar que o valor gasto com equipamentos de hardware seria de R$ 14.888,35 (Quatorze mil e oitocentos e oitenta e oito reais e trinta e cinco centavos) anual, esse valor seria apenas no primeiro ano. É possível ver que o custo já está muito acima do 1º cenário AWS.

Ainda no segundo cenário a empresa além dos hardwares, necessita de serviços mensais para manutenção e funcionamento adequado dos equipamentos, como listado na tabela 1, pode se observar que são serviços básicos para o pleno funcionamento do servidor, totalizando esses serviços em R$ 2.489,90 (dois mil e quatrocentos e oitenta e nove reais e noventa centavos) mensais e R$ 22.210,45 (vinte e dois mil e duzentos e dez reais e quarenta e cinco centavos) anuais.

Na ilustração 3 é possível analisar em porcentagem o que cada cenário representa em custos.

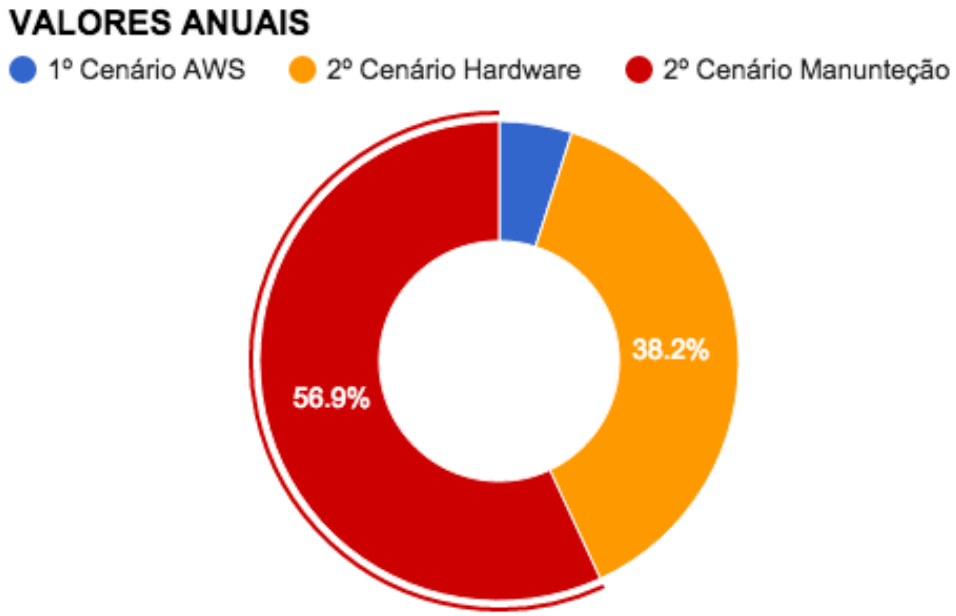


Ilustração 3: Custos anuais entre cenários

1º Cenário AWS	**R$ 1.915,20**
2º Cenário Hardware	**R$ 14.888,35**
2º Cenário Manuntenção	**R$ 22.210,45**

Analisando o 1º Cenário, é bastante claro que o custo anual não passa de R$ 2.000 (dois mil reais), já o 2º cenário terá custos apenas de manutenção de R$

22.210,45 (vinte e dois mil e duzentos e dez reais e quarenta e cinco centavos), isso representa 10 vezes o valor do 1º cenário em apenas um ano, ainda terá custos de equipamentos no primeiro ano de implantação da estrutura física, representando R$ 14.888,35 (Quatorze mil e oitocentos e oitenta e oito reais e trinta e cinco centavos), que comparado com o 1º cenário representa 7 vezes o valor. Na ilustração 3 é bastante claro o custo anual entre os cenários.

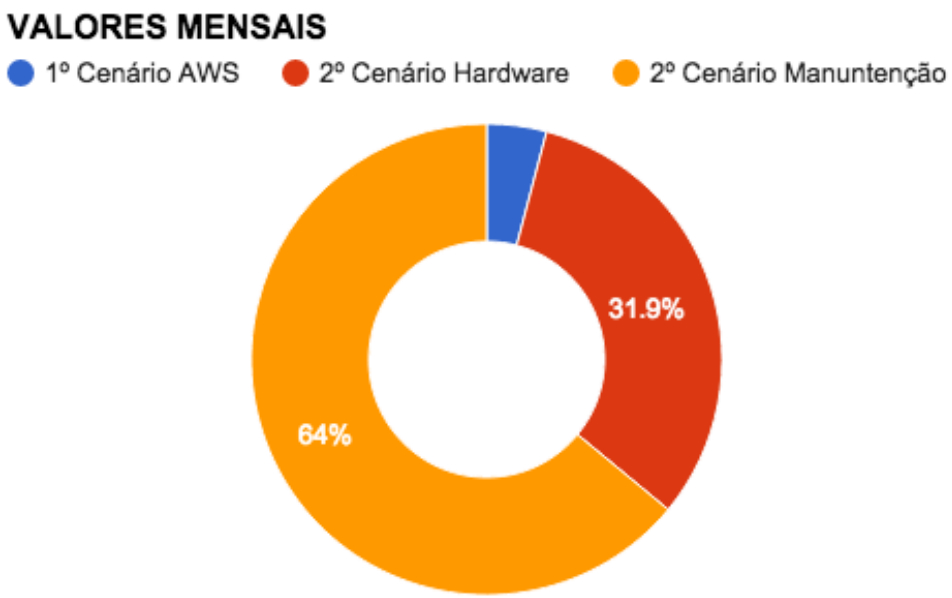


Ilustração 4: Custos mensais entre cenários

1º Cenário AWS	R$ 159,60
2º Cenário Hardware	R$ 1.240,70
2º Cenário Manuntenção	R$ 2.489,90

Na ilustração 4 o gráfico representa os custos mensais, também fica claro que o 1º Cenário é mais atraente que o 2º cenário, representando apenas R$ 159,60 (Cento e cinquenta e nove reais e sessenta centavos), e no cenário 2 a soma dos custos de manutenção de hardware será de R$ 3.730,60 (três mil e setecentos e trinta reais e sessenta centavos).

A solução adotada para esse trabalho foi a AWS, por representar o menor custo mensal, maior eficiência e garantia de disponibilidade. Com isso é possível observar que os custos com uma infraestrutra interna pode passar e superar muito as opções de virtualizações terceirizadas, como no caso da Amazon Web Services.

3.1.1.1 CONFIGURAÇÃO SERVIDOR 1

- Ponto de presença: South America (São Paulo)
- IP: 54.207.32.129
- Tipo de disco: SSD (GB)
- Sistema Operacional: Ubuntu Server 14.04.2 LTS
- Memória RAM: 1GB (Pouco recurso para o cenário)
- Aplicações: Varnish, Apache, NGINX
- Banco de Dados: PostgreSQL 9.4 (Master)

3.1.1.2 CONFIGURAÇÃO SERVIDOR 2

- Ponto de presença: US West (N. California)
- IP: 54.207.57.148
- Tipo de disco: SSD (GB)
- Sistema Operacional: Ubuntu Server 14.04.2 LTS
- Memória RAM: 1GB (Pouco recurso para o cenário)
- Banco de Dados: PostgreSQL 9.4 (Slave)

3.2 CONFIGURANDO NGINX PARA RECEBER AS REQUISIÇÕES

O usuário executará uma requisição a uma determinada página da internet, enviando assim um pedido para o servidor da página web. O primeiro a receber a requisição será o varnish, que devolverá a informação para o usuário se o conteúdo estiver em cache, quando não, o nginx ficará encarregado de processar a página para conteúdo estático.

Uma vez que o mesmo conteúdo da requisição seja dinâmico, o nginx enviará o processamento para o apache para então encaminhar ao usuário.

```
server {
        listen   8181;

        root /var/www/;
        index index.php index.html index.htm;

        listen 443 ssl;

        server_name example.com;
        ssl_certificate /etc/nginx/ssl/nginx.crt;
        ssl_certificate_key /etc/nginx/ssl/nginx.key;

        location / {
        try_files $uri $uri/ /index.php;
        }

        location ~ \.php$ {

        proxy_set_header X-Real-IP  $remote_addr;
        proxy_set_header X-Forwarded-For $remote_addr;
        proxy_set_header Host $host;
        proxy_pass http://127.0.0.1:8080;

         }

         location ~ /\.ht {
                deny all;
        }
}
```

Ilustração 5: Configuração Nginx

Esta configuração cria um sistema onde todas as extensões com um final php são reencaminhados para o servidor apache que está sendo executado na porta 8080, ficando assim o Nginx como servidor de front-end.

3.3 CONFIGURANDO E PREPARANDO O APACHE.

Servidor Apache pode trabalhar com conteúdo estático e fornece uma variedade de módulos multi-Processamento(MPMs) que praticamente ditam como as solicitações de clientes são tratadas. São eles:

mpm_prefork: Esse módulo é muito rápido por gerar processos com uma única thread para lidar com cada solicitação, porém o desempenho cai rapidamente após os pedidos superarem o número de processos.

mpm_event: Este módulo gera processo separado para que cada um possa gerenciar múltiplas threads, sendo uma variante do worker, assim atendendo mais requisições simultâneas.

mpm_worker: Este módulo implementa um multi-processamento, para atender as solicitações e é capaz de servir um grande número de solicitações com menos recursos do sistema.

Foi utilizado o mpm_prefork para tratar as requisições vindas do nginx, sendo assim, o apache tratará todas as páginas dinâmicas.

Configuração apache: **/etc/apache2/sites-available/example**

```
1 <VirtualHost 127.0.0.1:8080>
2 ~
3
4
5 ~
6 ~
```

Ilustração 6: Configuração Apache

3.4 CONFIGURAÇÃO E FUNCIONAMENTO DO VARNISH

O varnish é praticamente e basicamente um proxy HTTP reverso, que armazena todo o conteúdo HTTP requisitado na memória RAM, fazendo assim que o servidor não consulte e processe diversas vezes o mesmo conteúdo. O varnish é eficiente e ágil buscando em cache o objeto da requisição e devolve para o usuário final. Sendo assim um acelerador HTTP.

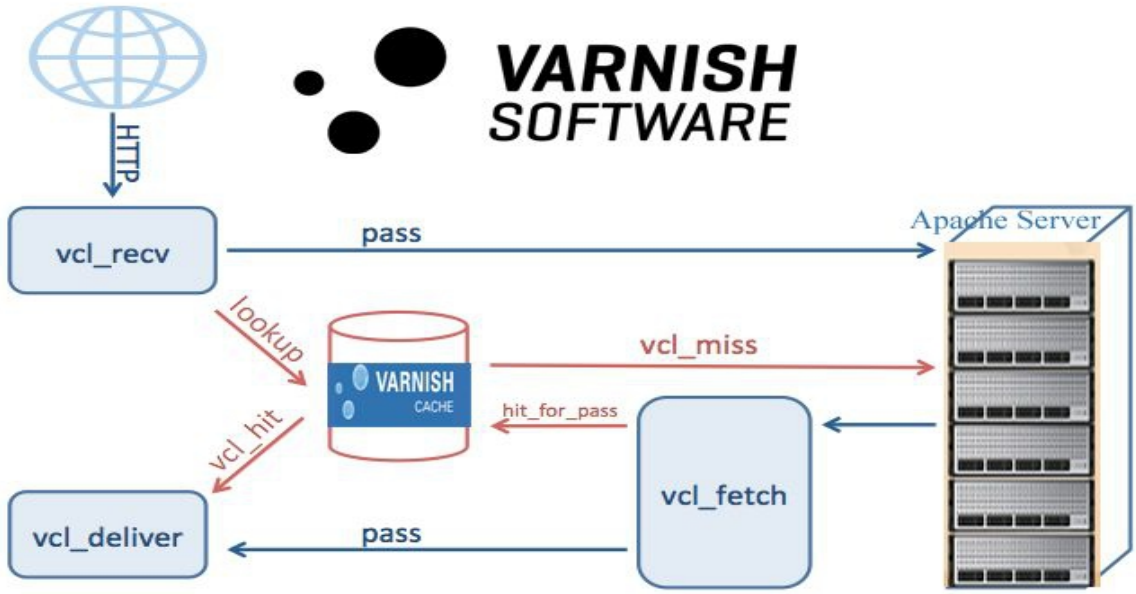
3.4.1 PROCESSO DE REQUISIÇÃO DO VARNISH

O varnish distingue três fases durante o processamento de um pedido. A solicitação é recebida pelo varnish através do navegador (vcl_recv). Nesta etapa, o papel do varnish é chamar a sub-rotina vcl_recv no arquivo de configuração (VCL).

Nesse momento, o cabeçalho do pedido realizado pelo usuário pode ser manipulado como por exemplo, a remoção de cookies, também pode ser decidido se o conteúdo solicitado deve ser pesquisado no cache ou ser enviado para o servidor back-end.

A resposta é recebida a partir do backend (vcl_fetch). Essa função é apenas executada se o conteúdo não for entregue pelo cache. Nesta fase, os cabeçalhos de resposta do servidor também podem ser manipulados e modificados, eles podem ser entregues ou ir para cache. Os atributos da requisição também estão disponíveis e podem ser utilizados para manipular várias configurações.

A resposta é enviada para o navegador (vcl_deliver). Este momento é passado por todas as solicitações e pode ser utilizado para adicionar cabeçalhos como TTL, alterar os cookies, dentre outras. E por fim os parâmetros de solicitação feita está disponível para leitura, finalizando o processo.



Fonte: Kharat, 2015

A ilustração 7[4] acima demonstra o funcionamento interno do varnish conforme as três etapas citadas, o que está em vermelho é relacionado ao cache, ou seja, todo o conteúdo em cache é procurado no cache, e entregue, se não for cache o servidor web será solicitado via vcl_fetch.

```
1  DAEMON_OPTS="-a :80 \
2               -T localhost:6082 \
3               -f /etc/varnish/default.vcl \
4               -S /etc/varnish/secret \
5               -s malloc,256m"
6
7
8  ## Alternative 3, Advanced configuration
```

Ilustração 8: Configuração Varnish /etc/default/varnish

4 Disponível em: http://wpapi.com/install-varnish-with-wordpress-and-apache/. Acesso em setembro. 2015.

```
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}
```

Ilustração 9: Configuração Varnish /etc/varnish/default.vcl

Realizando as configurações acima já é possível ter o varnish, nginx e apache funcionando em conjunto e realizando o balanceamento de carga. No tópico 4 será analisado os testes de configuração das aplicações e garantir que todo o processo está em funcionamento.

3.5 PREPARANDO POSTGRESQL PARA OPERAR EM MODO REPLICAÇÃO.

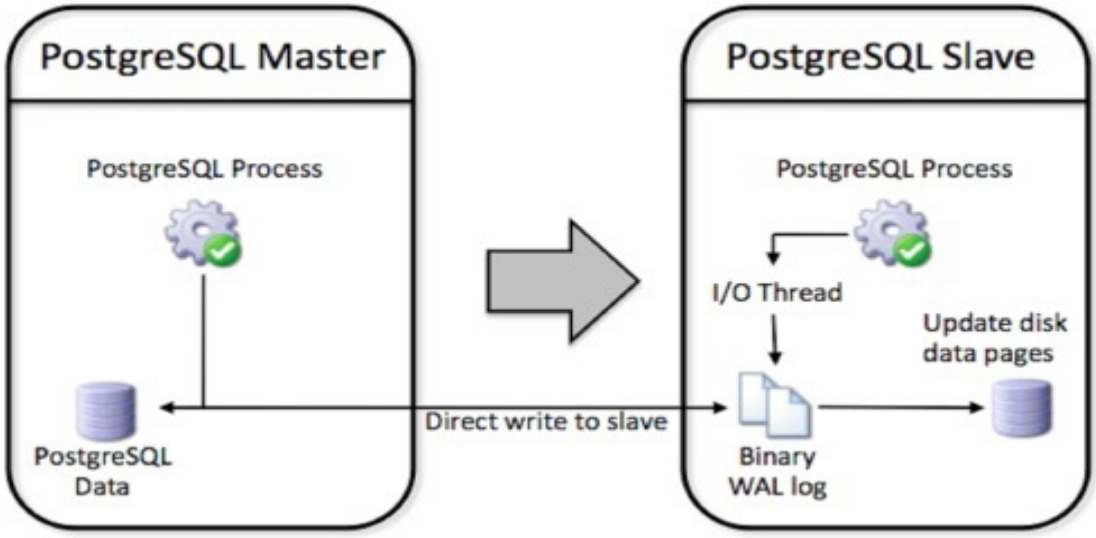
A replicação de banco de dados permitirá a redundância da informação, caso o servidor X falhe, o servidor Y entra e o serviço volta a funcionar sem perda de dados, pois qualquer comando executado no servidor X será replicado e executado no servidor Y.

1. O SGBD master registra alteração vinda pela aplicação em seu log binário.

2. O SGBD slave faz uma cópia dos eventos de log binário do SGBD master para o seu relay log.

3. O SGBD slave repete os eventos no seu relay log, assim aplicando as alterações no SGBD slave.

Na Ilustração 10[5] é possível observar como é realizado o processo de replicação SGBD Postgres.



lustração 10: Processo de replicação do PostreSQL
Fonte: Schumacher, 2015

Para o SGBD *slave* realizar a cópia do log binário, o mesmo inicia uma worker thread (tarefa). A tarefa aberta é chamada de thread I/O, sendo ela responsável em abrir uma conexão do cliente com o SGBD master, a partir desse momento é iniciado um processo de binlog o qual lê o evento a partir de log binário, assim a

5 Disponível em: https://holisticsecurity.wordpress.com/2010/11/24/comparing-mysql-and-postgres-9-0-replication/. Acesso em Julho. 2015.

thread I/O escreve os eventos no relay log do SGBD *slave*.

Depois do SGBD *slave* ter o log binário do SGBD master, o mesmo repete os comandos executados no master, assim tendo todas as alterações feitas no master agora no *slave*. E por fim todos os comandos executados no SGBD master serão replicados no SGBD *slave*.

3.5.1 CONFIGURAÇÃO POSTGRESQL MASTER – SLAVE

Para realizar a configuração do Postgres *Master* e o PostgreSQL S*lave*, é necessário tê-los rodando em dois servidores.

O primeiro passo é instalar o PostgreSQL em sua versão padrão como o seguinte comando em ambos os servidores.

```
root@ip-172-31-34-144:~# apt-get install postgresql
```

Ilustração 11: Instalando o Postgresql

Com o PostgreSQL é possível fazer replicação de duas maneiras (síncrona / assíncrona). Na síncrona, temos maior disponibilidade dos dados e segurança da informação mas para isso se tem um custo de transação, ou seja, o PostgreSQL salvará o registro inserido primeiro no master e depois no *slave*, e só assim liberará a transação. O modo síncrono é indicado para replicação em mesma rede, onde a latência é menor entre os servidores. Será configurado o PostgreSQL de modo assíncrono onde o servidor *slave*, estará sempre atrasado em relação ao *master*, mas não inconsistente, o modo assíncrono é útil para replicação de longa distância onde a latência é maior (DANTAS, 2015).

3.5.1.1 CONFIGURANDO POSTGRESQL MASTER

Após ter instalado o PostgreSQL, o próximo passo é configurar o servidor master, do qual será utilizado o servidor 1 que está alocado em uma instância na AWS, localizado em São Paulo com o IP público 54.207.32.129.

O primeiro arquivo para se configurar é o postgresql.conf que está localizado em /etc/postgresql/9.3/main/postgres.sql, neste arquivo será alterado os seguintes itens.

- listen_addresses
- wal_level
- wal_keep_segments
- max_wal_senders

3.5.1.1.1 LISTEN_ADDRESSES

```
listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
~
~
~
```

Ilustração 12: Configuração listen_addresses

Especifica o endereço TCP/IP em que o servidor está aguardando as configurações, altere o valor para *, com essa configuração qualquer IP com usuário e senha pode conectar-se ao PostgreSQL, deixaremos com * para podermos testar as conexões de uma máquina local, utilizando um software de gerenciamento de SGBD.

3.5.1.1.2 WAL_LEVEL

O parâmetro hot_standby é usado pelo PostgreSQL para executar em modo de recuperação de arquivos, nesse modo ele irá apenas executar consultas.

```
wal_level = 'hot_standby'        # minimal, archive, or hot_standby
~
~
```

Ilustração 13: Configuração wal_level

3.5.1.1.3 WAL_KEEP_SEGMENTS

Essa configuração controla o comportamento do built-in streaming de replicação. O wal_keep_segments especifica o número mínimo de segmentos de arquivo de log.

```
wal_keep_segments = 8
~
```

Ilustração 14: Configuração Listen_addresses

3.5.1.1.4 MAX_WAL_SENDERS

Especifica o número máximo de conexões simultâneas de servidores em espera, ou seja, em nosso ambiente teremos apenas um (1) servidor *slave*.

```
1  max_wal_senders = 1
2
```

Ilustração 15: Configuração `max_wal_senders`

```
root@ip-172-31-34-144:~# sudo su postgres
postgres@ip-172-31-34-144:/root$ psql
could not change directory to "/root": Permission denied
psql (9.3.6)
Type "help" for help.
~
postgres=# CREATE USER replicacao REPLICATION LOGIN ENCRYPTED PASSWORD
'teste7';
```

Ilustração 16: Criando usuário no PostgreSQL

Lembrando que o usuário para replicação pode ter qualquer nome, mas deverá ser do tipo *REPLICATION*.

3.5.1.2 CONFIGURANDO POSTGRESQL SLAVE.

Após ter instalado o PostgreSQL no servidor dois, será indispensável parar o PostgreSQL para realizar os ajustes de replicação.

```
root@ip-172-31-34-144:~# service postgresql stop
```

Ilustração 17: Stop PostgreSQL Slave

Com o PostgreSQL parado faz-se necessário utilizar o pg_basebackup, utilizado para fazer backups de bases em execução ou como ponto de partida para servidores de replicação. Para isso se deve executar o comando abaixo.

```
root@ip-172-31-34-144:~# pg_basebackup /var/lib/postgresql/9.2/main/
-h  54.207.32.129 -U replicacao
```

Ilustração 18: Executando pg_basebackup

E por fim basta dar o *start* no PostgreSQL, a partir daqui seu PostgreSQL está configurado como *replication*. O resultado pode ser visto no próximo capítulo de testes.

```
root@ip-172-31-34-144:~# service postgresql start
```

Ilustração 19: Start PostgreSQL Slave

4 TESTES E RESULTADOS

Após a finalização de todas as configurações dos servidores, foram executados diversos testes com o principal objetivo de garantir e verificar que todos os elementos tratados no trabalho estão funcionando corretamente.

Também foram executados vários testes de funcionamento, a fim de justificar o ambiente proposto. O balanceamento de carga e o aumento de performance com pouco recursos de hardware garantem assim redução de custos com equipamentos.

O mesmo acontece com a replicação de dados com PostgreSQL, foi realizado testes de integridade, garantindo 100% de autonomia do recurso.

4.1 TESTE DE CONFIGURAÇÃO VARNISH

Para realizar o teste de funcionamento do Varnish, deverá ser verificado primeiro se o processo está em execução no Linux com comando a seguir.

```
root@ip-172-31-34-144:~# ps aux | grep varnishd
```

Ilustração 20: Comando ps aux

A saída do comando executado será a lista de todos os processos em execução no sistema operacional, incluindo informações como número do processo, consumo de memória e CPU, hora que o processo se iniciou, dentre outras.

```
root@ip-172-31-34-144:~# nobody   12049  0.0  5.1 354140 52840
?        Sl   Jun10  32:25 /usr/sbin/varnishd -P
/var/run/varnishd.pid -a :80 -T localhost:6082 -f
/etc/varnish/default.vcl -S /etc/varnish/secret -s malloc,256m
```

Ilustração 21: Saída de processos em execução

Verificando o funcionamento do varnish, será necessário testar as configurações com o comando curl[6].

```
root@ip-172-31-34-144:~# curl -I http://localhost/corujao
```

Ilustração 22: Testando Varnish com curl

6 http://linux.about.com/od/commands/l/blcmdl1_curl.htm

```
HTTP/1.1 301 Moved Permanently
Server: Apache/2.4.7 (Ubuntu)
Location: http://localhost:8080/corujao/
Content-Type: text/html; charset=iso-8859-1
Content-Length: 314
Accept-Ranges: bytes
Date: Mon, 10 Aug 2015 13:38:42 GMT
X-Varnish: 2138268221
Age: 0
Via: 1.1 varnish
Connection: keep-alive
```

Ilustração 24: Testando Varnish com curl

4.2 TESTE DE CONFIGURAÇÃO NGINX

Como citado na ilustração 20, o primeiro passo é verificar se o processo do nginx está ativo. Após isso é executado o seguinte comando.

```
root@ip-172-31-34-144:~# curl -I http://localhost:8181/corujao
```

Ilustração 25: Testando Nginx com curl

Com o comando da ilustração 25, já se pode testar se o nginx, se o mesmo responderá e estará ativo na porta 8181 como definido na configuração. A saída do comando pode ser visualizada na ilustração 26.

O nginx está ativo e respondendo na porta 8181, ou seja, toda requisição que o varnish não ter cache o nginx ficará responsável em receber e interpretar.

```
HTTP/1.1 301 Moved Permanently
Server: nginx/1.4.6 (Ubuntu)
Date: Mon, 10 Aug 2015 13:42:47 GMT
Content-Type: text/html
Content-Length: 193
Location: http://localhost:8181/corujao/
Connection: keep-alive
```

Ilustração 26: Saída curl Nginx

4.3 TESTE DE CONFIGURAÇÃO APACHE

Como visto na ilustração 20 e no tópico 4.2, será necessário primeiro verificar se o processo do apache está ativo, para isso se deve executar o seguinte comando.

```
root@ip-172-31-34-144:~# curl -I http://localhost:8080/corujao
```

Ilustração 27: Testando o Apache com curl

Com o comando da ilustração 27, gerará a saída similar a ilustração 28, onde é possível constatar o funcionamento do apache da porta 8080, ou seja, toda requisição que o nginx não conseguir interpretar, o apache ficará responsável por interpretar e devolver ao usuário.

```
HTTP/1.1 301 Moved Permanently
Date: Mon, 10 Aug 2015 13:38:50 GMT
Server: Apache/2.4.7 (Ubuntu)
Location: http://localhost:8080/corujao/
Content-Type: text/html; charset=iso-8859-1
```

Ilustração 28: Saída do comando curl

4.4 CONCLUSÃO TESTES DE SERVIDOR

Com os testes citados nos tópicos 4.1, 4.2 e 4.3 pode se verificar o funcionamento individual de cada serviço, que no final resultado no funcionamento composto, por fim finalizando o cenário proposto.

Pode se concluir com êxito os testes de funcionamento dos serviços no servidor alocado na AWS.

4.5 RESULTADO DOS TESTES DE REPLICAÇÃO SGBD POSTGRESQL

Para garantir que ambos servidores estão em pleno funcionamento, deve-se executar o comando a seguir.

```
root@ip-172-31-34-145:~# ps aux | grep postgres
```

Ilustração 29: Comando para listar processos em execução no PostgreSQL Master

No servidor um denominado como master, o resultado mostra todos processos em execução do PostgreSQL, pode-se observar um em especial que

mostra a instância de replicação do PostgreSQL *slave* (*postgres 16932 0.0 0.4 247612 4256 ? Ss Jun22 0:00 postgres: wal sender process replicacao 54.207.57.148(56582) streaming 0/F003C78*), mostrando assim que o PostgreSQL *master* está com um processo ativo, transmitindo para o IP 54.207.57.148 todos os processos que são registrados.

```
ubuntu    2047  0.0  0.0  10460   940 pts/1    S+   01:40   0:00
grep --color=auto postgres
postgres 14661  0.0  1.5 246480 16180 ?        S    Jun22   0:00
/usr/lib/postgresql/9.3/bin/postgres -D /var/lib/postgresql/9.3/main
-c config_file=/etc/postgresql/9.3/main/postgresql.conf
postgres 14664  0.0  0.3 246612  3176 ?        Ss   Jun22   0:00
postgres: checkpointer process
postgres 14665  0.1  0.2 246480  2676 ?        Ss   Jun22   0:24
postgres: writer process
postgres 14666  0.0  0.1 246480  1664 ?        Ss   Jun22   0:00
postgres: wal writer process
postgres 14667  0.0  0.2 247340  3028 ?        Ss   Jun22   0:00
postgres: autovacuum launcher process
postgres 14668  0.0  0.1 102124  1660 ?        Ss   Jun22   0:00
postgres: archiver process   last was 000000010000000000000E
postgres 14669  0.0  0.1 102124  1768 ?        Ss   Jun22   0:00
postgres: stats collector process
postgres 16932  0.0  0.4 247612  4256 ?        Ss   Jun22   0:00
postgres: wal sender process replicacao 54.207.57.148(56582)
streaming 0/F003C78
```

Ilustração 30: Processos em execução Postgres Master

No segundo servidor, executando o mesmo comando pode-se observar que o PostgreSQL *slave* está em modo *receiver* (*postgres 5344 0.0 0.4 257708 4648 ? Ss Jun22 0:06 postgres: wal receiver process streaming 0/F003EF8*), recebendo os processos executados no master, também são executados no *slave*.

```
root@ip-172-31-34-144:~# ps aux | grep postgres
```

Ilustração 31: Comando para listar processos em execução no PostgreSQL Slave

```
root@ip-172-31-34-144:~# ps aux | grep postgres
postgres  5342  0.0  1.5 246484 16176 ?        S    Jun22   0:00
/usr/lib/postgresql/9.3/bin/postgres -D /var/lib/postgresql/9.3/main
-c config_file=/etc/postgresql/9.3/main/postgresql.conf
postgres  5343  0.0  0.3 246644  3480 ?        Ss   Jun22   0:00
postgres: startup process   recovering 000000010000000000000000F
postgres  5344  0.0  0.4 257708  4648 ?        Ss   Jun22   0:06
postgres: wal receiver process   streaming 0/F003EF8
postgres  5345  0.0  0.2 246624  2916 ?        Ss   Jun22   0:00
postgres: checkpointer process
postgres  5346  0.0  0.2 246484  2676 ?        Ss   Jun22   0:00
postgres: writer process
postgres  5347  0.0  0.1 102128  1676 ?        Ss   Jun22   0:00
postgres: stats collector process
root      6060  0.0  0.0  10460   940 pts/0    S+   02:03   0:00
grep --color=auto postgres
```

Ilustração 32: Processos em execução Postgres Slave

E por fim será realizada a criação de um conjunto de tabelas que simbolizam a estrutura de uma base de dados para cadastro de país, estado, cidade, bairro e rua de todo o Brasil, além da criação das tabelas, será importado uma dump, com todas os estados, cidades, bairros e ruas do Brasil, as tabelas geradas são, end_endereco, end_cep, end_bairro, end_cidade, end_estado e end_pais, o banco de dados será endereços.

Abaixo é possível ver os scripts de geração das tabelas citadas acima, será executado no servidor master, e após a geração é possível ver as tabelas replicadas no *slave*. Em anexo a modelagem de dados das tabelas citadas.

4.6 SCRIPT SQL DA MODELAGEM FÍSICA.

```
1  CREATE TABLE public.end_endereco(
2          endereco_id serial NOT NULL,
3          bairro_id integer,
4          cep numeric,
5          logradouro_tipo character varying,
6          rua character varying,
7          numero character varying,
8          complemento character varying,
9          observacao character varying,
10         data_adicionado timestamp DEFAULT now(),
11         CONSTRAINT endereco_id PRIMARY KEY (endereco_id)
12
13 );
14
15 ALTER TABLE public.end_endereco OWNER TO ovelha;
16
17 CREATE TABLE public.end_cep(
18         cep_id serial NOT NULL,
19         bairro_id integer,
20         rua varchar,
21         cep numeric,
22         logradouro_tipo varchar,
23         CONSTRAINT cep_id PRIMARY KEY (cep_id)
24
25 );
26
   ALTER TABLE public.end_cep OWNER TO ovelha;
```

Ilustração 33:Script create table end_endereco, end_cep

```
CREATE TABLE public.end_cidade(
        cidade_id serial NOT NULL,
        estado_id integer,
        nome character varying(200) NOT NULL,
        codigo_ibge integer,
        cmun character varying,
        area_territorial numeric,
        data_adicionado timestamp NOT NULL DEFAULT now(),
        CONSTRAINT cidade_id PRIMARY KEY (cidade_id)

);

ALTER TABLE public.end_cidade OWNER TO ovelha;

CREATE TABLE public.end_estado(
        estado_id serial NOT NULL,
        pais_id integer,
        nome character varying(200) NOT NULL,
        sigla character varying(255) NOT NULL,
        cuf character varying,
        data_adicionado timestamp NOT NULL DEFAULT now(),
        CONSTRAINT estado_id PRIMARY KEY (estado_id)

);

ALTER TABLE public.end_estado OWNER TO ovelha;


CREATE TABLE public.end_pais(
        pais_id serial NOT NULL,
        nome character varying,
        sigla character varying,
        cpais varchar,
        data_adicionado timestamp NOT NULL DEFAULT now(),
        CONSTRAINT pais_id PRIMARY KEY (pais_id)

);

ALTER TABLE public.end_pais OWNER TO ovelha;

CREATE TABLE public.end_bairro(
        bairro_id serial NOT NULL,
        cidade_id integer,
        nome character varying,
        data_adicionado timestamp,
        CONSTRAINT bairro_id PRIMARY KEY (bairro_id)

);

ALTER TABLE public.end_bairro OWNER TO ovelha;

```

Ilustração 34: Script create table end_cidade, end_estado, end_pais, end_bairro

```
1  ALTER TABLE public.end_cep ADD CONSTRAINT end_bairro FOREIGN KEY
2  (bairro_id)
3  REFERENCES public.end_bairro (bairro_id) MATCH FULL
4  ON DELETE NO ACTION ON UPDATE NO ACTION;
5
6  ALTER TABLE public.end_cidade ADD CONSTRAINT end_estado FOREIGN KEY
7  (estado_id)
8  REFERENCES public.end_estado (estado_id) MATCH FULL
9  ON DELETE NO ACTION ON UPDATE NO ACTION;
10
11 ALTER TABLE public.end_estado ADD CONSTRAINT end_pais FOREIGN KEY
12 (pais_id)
13 REFERENCES public.end_pais (pais_id) MATCH FULL
14 ON DELETE NO ACTION ON UPDATE NO ACTION;
15
16 ALTER TABLE public.end_bairro ADD CONSTRAINT end_cidade FOREIGN KEY
17 (cidade_id)
18 REFERENCES public.end_cidade (cidade_id) MATCH FULL
19 ON DELETE NO ACTION ON UPDATE NO ACTION;
20
21 ALTER TABLE public.end_endereco ADD CONSTRAINT end_bairro FOREIGN
22 KEY (bairro_id)
23 REFERENCES public.end_bairro (bairro_id) MATCH FULL
   ON DELETE NO ACTION ON UPDATE NO ACTION;
```

Ilustração 35:Script add foreign

5 CONCLUSÃO

Evidencia-se que atualmente as empresas de tecnologia passam por constantes atualizações de hardware e software, criando assim gastos exorbitantes e talvez desnecessários. Com a solução apresentada neste trabalho é possível ver e analisar como melhorar a performance de um servidor web.

Constatou-se que realizando as configurações adequadas nos serviços Varnish, Nginx e Apache é possível diminuir custos com hardware e aumentar o desempenho do servidor. No presente trabalho foi feito o levantamento necessário para se colocar em prática o ambiente proposto e com principal objetivo, mostrar, experimentar, projetar e avaliar um ambiente replicado e com balanceamento de carga com Apache, Nginx e Varnish.

De maneira bastante satisfatória se pode observar o funcionamento da replicação de dados com o PostgreSQL onde obteve-se 100% de êxito em seu funcionamento, criando assim uma solução de baixo custo e com grande importância. Proporcionando também uma cópia em tempo real de uma instância com PostgreSQL no Brasil para uma instância AWS nos Estados Unidos.

A replicação de dados, do cenário apresentado permitiu implementar e demonstrar seu funcionamento. Revelando de maneira clara o fluxo e o funcionamento do balanceamento de carga entre as aplicações Varnish, Nginx e Apache, através dos seus arquivos de logs.

Através dos testes realizados conclui-se que é possível executar, sem grandes dificuldades, um ambiente de balanceamento de carga e replicação de dados com o PostgreSQL, garantindo uma performance muito maior em relação aos serviços separados e não configurados adequadamente.

Salienta-se que esse trabalho não apresenta final conclusivo, visando apenas contribuir para empresas de tecnologia e abrir novos caminhos para desenvolvedores e empresários das áreas de softwares. Cabe ao desenvolvedor implementar estratégias para melhorar a perfomance dos seus aplicativos atuais.

5.1 TRABALHOS FUTUROS

Este trabalho avaliou o desempenho e a performance do Apache, Nginx e Varnish em conjunto, assim alcançando o seu principal objetivo redução de custos com hardware. Uma proposta para um trabalho futuro é desenvolver um software gerencial para empresas 100% otimizado para ser aplicado ao servidor otimizado.

Podendo assim realizar novos testes de performance do lado do servidor e do lado da aplicação, extraindo ainda mais dados para serem analisados, para se alcançar o menor custo possível para um projeto de médio porte na web.

REFERÊNCIAS

STATISTIC BRAIN. Google Annual Search Statistics. 2015. Disponível em: <http:// www.statisticbrain.com/google-searches/>. Citado na página 8.

ABREU, Thiago W. M. Sba: Controle & Automação Sociedade Brasileira de Automatica – 15/05/2006. Disponível em: <http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592012000100004&lng=pt&nrm=iso> Acessado em: 2 de maio de 2015. Citado na página 11.

REIS, Lucas. O que é Nginx? - 30/08/2013. Disponível em: <http://www.bravulink.com.br/o-que-e-nginx/> Acessado em: 2 de maio de 2015. Citado na página 11.

DIAS, Neto. Varnish Cache - O que é e como implementá-lo? - 19/02/2012. Disponível em: <http://blog.hostdime.com.br/materias/tecnologia/varnish-cache-o-que-e-e-como-implementa-lo/> Acessado em: 2 de maio de 2015. citado na página 12.

4LINUX. O que é Postgresql? Disponível em: <http://www.4linux.com.br/o-que-e-postgresql> Acessado em : 2 de maio de 2015. citado na página 12.

TECHNET. Como funciona o Balanceamento de Carga de Rede. Disponível em: <https://technet.microsoft.com/pt-br/library/cc738894%28v=ws.10%29.aspx> Acessado em : 2 de maio de 2015. citado na página 12.

SMANIOTTO, Carlos Eduardo. Artigo da SQL Magazine 24 - Replicação e alta disponibilidade no PostgreSQL. Disponível em: <http://www.devmedia.com.br/artigo-da-sql-magazine-24-replicacao-e-alta-disponibilidade-no-postgresql/6140> Acessado em : 2 de maio de 2015. citado na página 13.

ALECRIM, Emerson. Artigo da SQL Magazine 24 – 22/03/2013. Cluster: conceito e características. Disponível em: <http://www.infowester.com/cluster.php> Acessado em : 2 de maio de 2015. citado na página 13.

SMANIOTO, C. E. Replicação e alta disponibilidade no PostgreSQL. Revista SQLMagazine, 2008. Disponível em:<http://www.devmedia.com.br/post-6140-Artigo-da-SQL-Magazine-24-Replicacao-e-alta-disponibilidade-no--PostgreSQL.html>. Acesso em: 13 junho. 2015.

LANDEMAINE, Charles André. O que é Conteúdo Estático? Disponível em: <http://www.auriance.net/faq/index.php/sid=1&lang=pt&action=artikel&cat=1&id=15&a

rtlang=pt> Acessado em : 14 de junho de 2015. citado na página 15.

TECNOLOGIA CULTURA MIX. O que é Conteúdo Dinâmico? Disponível em: <http://tecnologia.culturamix.com/noticias/o-que-e-conteudo-dinamico> Acessado em : 4 de junho de 2015. citado na página 16.

RAMOS, Wagner Corrêa. Artigo SQL Magazine 15 - Projetando sistemas descentralizados usando replicação de dados "multi-master". Disponível em: <http://www.devmedia.com.br/artigo-sql-magazine-15-projetando-sistemas-descentralizados-usando-replicacao-de-dados-multi-master/5712#ixzz3i8XIBX4F> Acessado em : 7 de agosto de 2015. citado na página 13.

HEEN, Tollef Fog. Varnish Book. Disponível em: <https://www.varnish-software.com/book/3/> Acessado em : 7 de agosto de 2015. citado na página 19.

DANTAS, Cleber. Latência, largura de banda e a velocidade da luz. Disponível em: <http://tableless.com.br/latencia-largura-de-banda-e-a-velocidade-da-luz/> Acessado em : 7 de agosto de 2015. citado na página 26.

ALECRIM, Emerson. Conhecendo o Servidor Apache (HTTP Server Project) – 15/05/2006. Disponível em: <http://www.infowester.com/servapach.php> Acessado em: 2 de maio de 2015.

APACHE. Http Server – Version 2.4. Disponível em: <https://httpd.apache.org/> Acessado em: 05/09/2015.

VARNISH CACHE. Installation on Ubuntu – Quick Install Ubuntu. Disponível em: <https://www.varnish-cache.org/docs> Acessado em: 9/09/2015.

NGINX. Installing Nginx Plus on Amazon EC2. Disponível em: <https://www.nginx.com/resources/admin-guide/setting-nginx-plus-environment-amazon-ec2/> Acessado em: 15/09/2015.

AWS. Features of Amazon EC2. Disponível em: <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html> Acessado em: 15/09/2015.

PostgreSQL. High Availability, Load Balancing, and Replication. Disponível em: <http://www.postgresql.org/docs/9.1/static/high-availability.html> Acessado em: 12/09/2015.

ANEXOS

ANEXO A – MODELAGEM DE DADOS