UNIVERSIDADE PAULISTA
VICE-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA

RELATÓRIO SEMESTRAL DE PESQUISA

AVALIAÇÃO DE DESEMPENHO DE APLICAÇÃO DE E-COMMERCE: UMA ABORDAGEM ORIENTADA À ESCALABILIDADE

Autor: Kerolláine Lauto de Oliveira
Curso: Ciência da Computação
Campus: Araraquara
Orientador: Prof. M.Sc. Renê de Souza Pinto

Pesquisa financiada pelo Santander,
Programa Santander Universidades de Bolsas de Educação

AVALIAÇÃO DE DESEMPENHO DE APLICAÇÃO DE E-COMMERCE: UMA ABORDAGEM ORIENTADA À ESCALABILIDADE

Kerolláine Lauto de Oliveira

Nos dias atuais presencia-se um desenvolvimento desenfreado na quantidade e complexidade dos serviços disponibilizados na *web*. Devido a característica essencial da globalidade da rede, esses serviços tendem a sofrer mudanças repentinas e significativas na quantidade de usuários que os utilizam. Para garantir que um serviço seja capaz de suportar essas flutuações, é fundamental a elaboração de um plano de capacidade de recursos computacionais. Para que tal plano seja elaborado e mantido, são necessárias informações sobre o desempenho da aplicação, garantia da qualidade do serviço - *QoS*, entre outros. Para isso, o *benchmark* Bench4Q, que é uma extensão da especificação do *benchmark* TPC-W, emula sessões reais de usuários de determinado sistema assim como processa os resultados, gerados através da execução do *benchmark* com carga de trabalho anteriormente definida.

Outros experimentos científicos concluíram que o TPC-W pode ser útil em diversas situações específicas, mas para o objetivo de elaborar um plano de capacidade de recursos computacionais torna-se necessário um controle para garantia da *QoS*. Neste cenário, o Bench4Q atende aos requisitos e otimiza a carga de trabalho em tempo real.

Palavras-chave: Análise de desempenho; Benchmark; Bench4Q; TPC-W; E-commerce; Escalabilidade; QoS.

SUMÁRIO

INTRODUÇÃO

A adoção de ambientes de computação em nuvem é crescente, e eles tem sido utilizados desde sistemas operacionais até aplicações menores, alterando fortemente o paradigma de computação definido. Com esta nova demanda de aplicações surge a necessidade dos serviços serem nativamente escaláveis [3].

Escalabilidade é um desafio em um projeto de aplicação de *e-commerce.* Provedores precisam saber como o desempenho da aplicação varia de acordo com o número de usuários acessando, o número máximo de transações processadas por segundo, atualização da aplicação para suportar grandes volumes de informação ou mudança arquitetural requerida. Quando é necessário escolher um ambiente para implantar determinada aplicação, é necessário conhecer diversas combinações de servidores, *hardware*, entre outros. Para isso, pode-se fazer uso de *benchmarks* que comparam alternativas concorrentes [2].

Por outro lado, observa-se uma forte tendência no desenvolvimento de serviços baseados em arquiteturas dispostas em camadas, cujo objetivo é aumentar a coesão da funcionalidade ofertada [4]. Dentre todos os benefícios e boas práticas existentes na disciplina de engenharia de *software*, destacam-se a melhoria de manutenção e o isolamento de problemas. Com isso, é possível analisar como é feita a demanda para uma camada e implantá-la de forma distribuída - caso em que haveria escalabilidade horizontal. Esta escalabilidade trata o aumento da quantidade de dispositivos computacionais (*hardware* físico ou virtual) que hospedam aquela camada.

Sendo assim, esta pesquisa pretende proporcionar a apreciação da condução pragmática de experimentos para modelar a capacidade de um sistema computacional. Os procedimentos experimentais e todos os artefatos gerados (*scripts*, tipos de testes, planilhas de planejamentos, tipos de gráficos) contribuirão para a condução do mesmo tipo de avaliação focada em outros sistemas.

Esta pesquisa tem como objetivo (1) o estudo de um *benchmark* de aplicação de *e-commerce* e (2) a análise de métricas de desempenho que auxiliem no planejamento de capacidade de recursos computacionais de um serviço compatível com o objeto de estudo.

Pretende-se avaliar o desempenho de aplicações de *e-commerce*, utilizando uma abordagem orientada à escalabilidade. Para isso, é necessário um ambiente e

ferramentas que sejam adequadamente implementados e implantados, para que seja alcançado um resultado satisfatório. Espera-se de resultados os artefatos gerados como *scripts*, gráficos, planilhas de planejamento entre outros. Estes artefatos serão possíveis de ser gerados por utilizarem o conteúdo que será produzido pelos testes abordados, que analisarão vários aspectos tanto da aplicação escalável quanto os aspectos funcionais e comportamentais dos serviços configurados. Com estes fatores resultantes, espera-se a definição e execução de procedimentos experimentais que permitam a reprodução dos testes. Todo o *benchmark* executado e procedimentos experimentais terão o intuito de ser ajustável para outras aplicações, desde que sejam do mesmo cenário, protocolo utilizado e sejam escaláveis.

Para a execução do *benchmark* utiliza-se nesta pesquisa a especificação TPC-W e ferramenta Bench4Q, que compreende aplicações *web* e diversas configurações. Entretanto, devido à alguns pontos negativos mostrados em resultados de experimentos, publicados cientificamente e detalhados na seção DISCUSSÃO, a ferramenta utilizada para análise de desempenho de aplicações de *e-commerce* será o Bench4Q, que é uma extensão do *benchmark* TPC-W para sistemas *e-commerce* e suporta análise de métricas baseadas em sessões, tem gerador de carga baseado em agentes e como *Software Under Test*(SUT) tem uma loja virtual de livros [6].

MÉTODO

O processo de pesquisa foi dividido em quatro etapas, sendo:

- Levantamento sobre ferramentas e avaliação de aplicações escaláveis;
- Implantação ou produção de um conjunto para executar o *benchmark*;
- Análise e categorização de dados obtidos;
- Execução de estudo de caso e extração dos procedimentos experimentais.

Essas quatro etapas foram separadas em camadas, conforme especificadas abaixo, e compõem a estrutura metodológica a ser criada ao longo da pesquisa.

As camadas foram divididas em três segmentos principais. A camada **Benchmark** é responsável por conter o conjunto de ferramentas e métodos que avaliará a aplicação escalável, e é formada desde funções nativas do sistema operacional até programas específicos, e outras ferramentas de extensão, assim como a execução do *benchmark*. Esta camada proverá para a camada **Resultados** definições estatísticas, *profile* da aplicação, utilização de CPU, entre outros. A camada **Aplicação** é composta pelos elementos do serviço de *e-commerce* em teste, uma aplicação HTTP escalável. O objeto de estudo neste caso foi baseado na especificação TPC-W, a qual será detalhada na seção MÉTODO. Esta camada é configurável e integrável, uma vez que os parâmetros dos softwares básicos (sistema operacional, servidor *web* de aplicação, balanceador de carga, ambiente elástico etc.) e da aplicação (mapeamento de implantação, acoplamento entre os componentes, tamanho do *buffer* e filas etc.) poderão ser explorados no intuito de expor a dinâmica de desempenho específica para o experimento. A camada **Resultados** terá as variáveis resultantes de camadas anteriores, como o tempo médio de acesso, número de requisições que foram respondidas com sucesso, o gráfico de utilização de CPU, gráfico de utilização de I/O, gráfico de utilização de rede, descrição dos experimentos, análise de influência de fatores, replicações, intervalo de confiança estatístico etc. Apesar de todos estes resultados serem gerados pela camada **Benchmark**, esta interpreta os dados recebidos e gera os relatórios desejados utilizando as variáveis resultantes como acima descrito, explorando ao máximo os resultados oferecidos pelo sistema operacional. O sistema operacional utilizado no experimento foi o Linux.

Durante este período da pesquisa foram executadas as etapas levantamento sobre ferramentas e avaliação de aplicações escaláveis; Implantação ou produção de um conjunto para executar o *benchmark* e parcialmente análise e categorização de dados obtidos.

O principal foco foi aprimorar os métodos e materiais, anteriormente definidos pelo projeto de pesquisa, a partir de informações retiradas de material científico publicado até o período desta pesquisa sobre o tema. O objetivo principal deste aprimoramento foi definir um conjunto de especificação detalhado, o qual é de extrema relevância para obter-se melhores resultados.O principal objeto de estudo no decorrer do levantamento sobre ferramentas e avaliação de aplicações escaláveis foi o Bench4Q.

O TPC-W é uma especificação de *benchmark* que contém três principais componentes: (1) O sistema que será avaliado, chamado SUT (*System Under Test*), (2) carga de trabalho que será submetida ao SUT - especificando o tipo de requisição e intensidade de carga de trabalho - e (3) uma ou mais métricas de performance obtidas por meio de um tipo de monitoramento e avaliação do SUT. Compreende-se *o* SUT do TPC-W por todos os servidores *web*, servidores de aplicação, servidores de banco de dados, balanceadores de carga, redes internas e a interface de rede requerida para implementar um *e-commerce* simulado, conforme Figura 1. Esta especificação de *benchmark* contém em sua base de dados no mínimo as tabelas cliente, endereços, compra, item da compra, transação de cartão de crédito e autor.

Em um *e-commerce* os clientes interagem com a aplicação através de sessões, que são sequências de solicitações consecutivas para executar funções de negócio durante uma única visita ao *site*. Funções de negócio são compreendidas por pesquisar, adicionar ao carrinho, pagar, entre outras ações que o usuário final teria possibilidade dentro do *e-commerce* avaliado [7].

Uma forma de captar o padrão de navegação dentro de uma sessão é através do gráfico modelo de comportamento do cliente (Figura 2), denominado CBMG, o qual descreve como os usuários navegam pelo site, quais ações eles executam e quantas vezes, e a frequência de transições entre funções de negócio. Pode-se observar vários padrões de comportamento de usuários em um *e-commerce*.

Diferentes padrões de comportamento de clientes geram diferentes cargas sobre os recursos computacionais da aplicação [7].

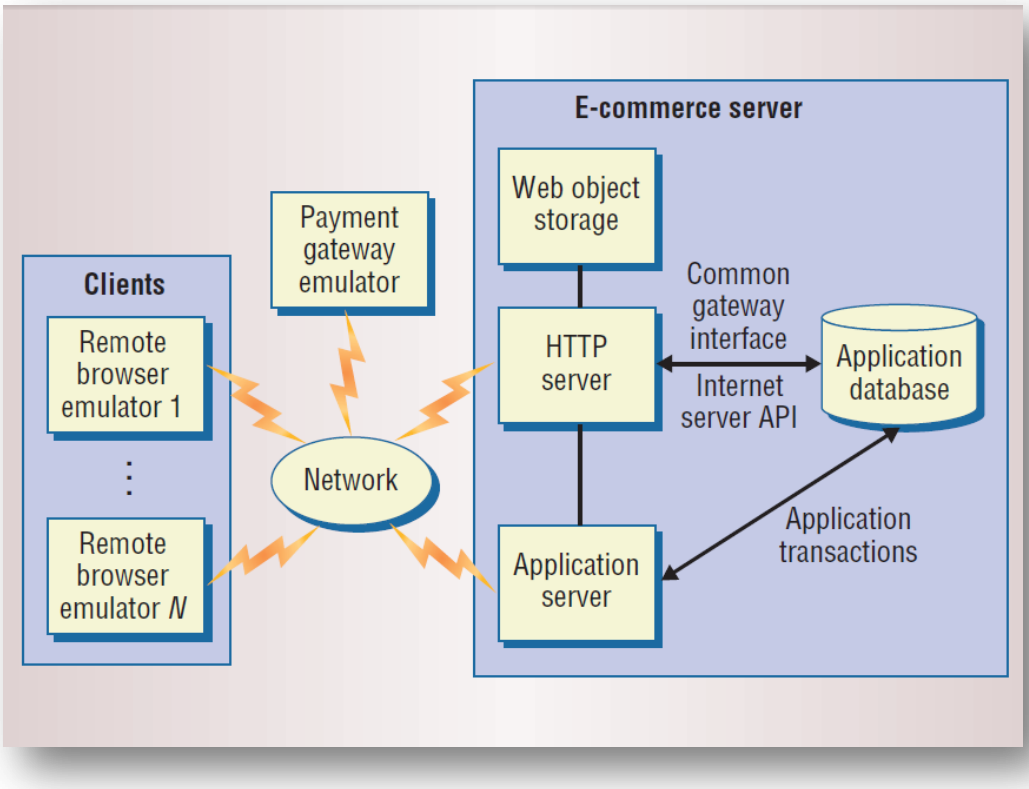

Figura 1: Arquitetura do *benchmark* TPC-W [7]

O TPC-W imita uma loja virtual de uma livraria. As principais características deste *benchmark* incluem:

- Múltiplas sessões de *browsers online* que emulam ações do usuário final;
- Paginas dinâmicas geradas por meio de acesso ao banco de dados da aplicação e alterações nesta base;
- Autenticação via SSL (*Secure Socket Layer*) e TLS (*Transport Layer Security*);
- Simulação de pagamentos;
- Base de dados constituída de múltiplas tabelas com variados tamanhos, atributos e relacionamentos;
- Várias operações de transações e funções de negócio na aplicação.

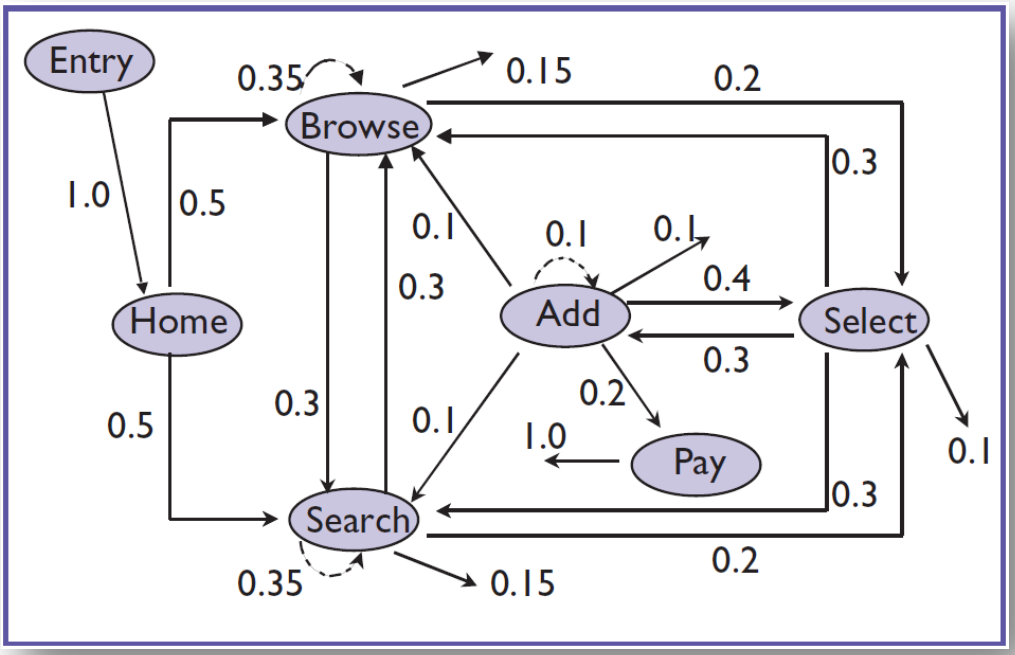

Figura 2: Exemplo de um modelo gráfico do comportamento de um cliente (CBMG) [2]

O TPC-W lida com escalabilidade por estabelecer uma relação chamada fator de escalabilidade entre o número de sessões concorrentes e o tamanho da loja, mensurado em termos de número de itens do inventário [2].

Para a geração de carga de trabalho, este *benchmark* cria requisições para o SUT utilizando um grupo de *browsers* emulados (EB). Esses EBs agem como usuários da aplicação enviando e recebendo conteúdo em HTML via navegador por protocolos HTTP e TCP/IP através de uma conexão de rede. Como descrito, o tamanho do SUT e o fator de escalabilidade determina o número de EBs utilizados para o teste, sendo que cada EB envia uma série de requisições juntamente com o usuário de determinada sessão. A carga de trabalho gerada pelo EB é especificada pelo padrão de navegação com a sessão, representado pelo modelo CBMG conforme Figura 3, e a intensidade da carga de trabalho, especificada pelo número de EBs e tempo de resposta.

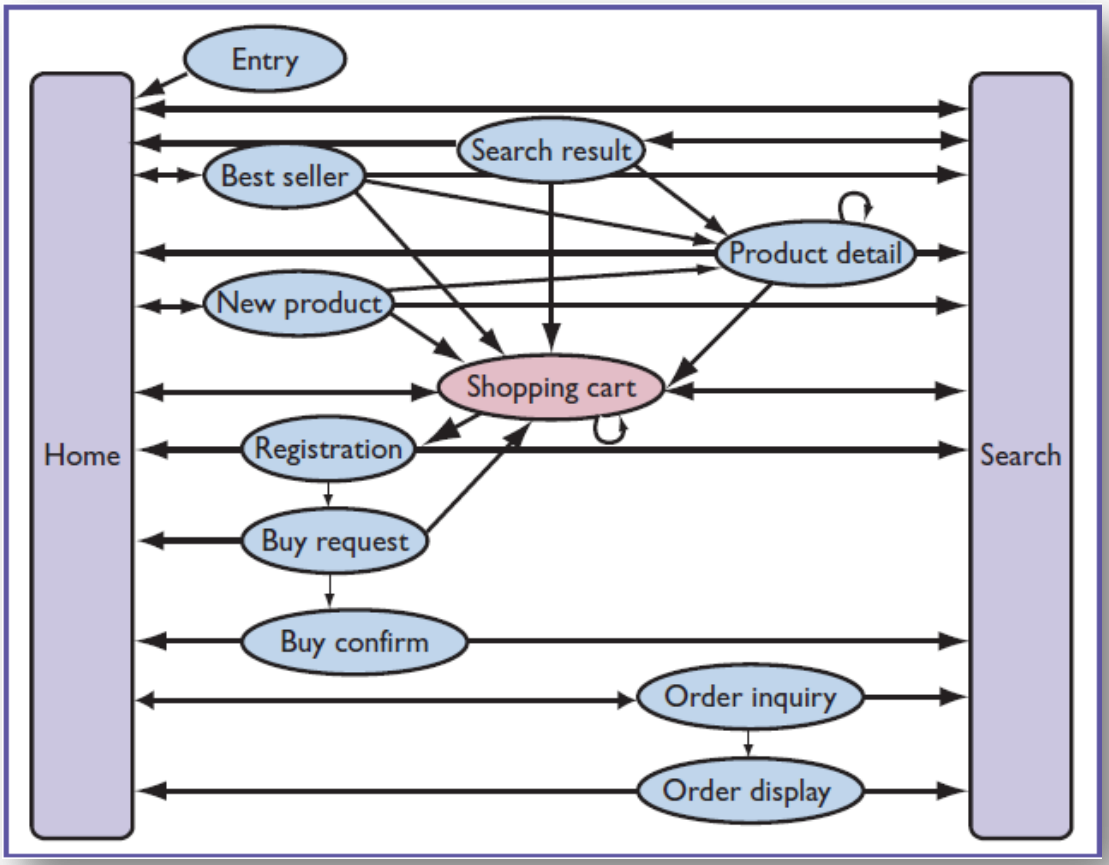

Figura 3: O gráfico mostra possíveis transações entre os estados durante uma sessão de usuário no TPC-W (CBMG) [7].

O TPC-W mensura duas métricas principais: (1) interações *web* por segundo (WIPS) durante uma compra e (2) o custo por WIPS, que é a relação entre o custo total do SUT (incluindo *hardware* e manutenção em três anos, *software* e manutenção) e valor do WIPS.

Além disso, o TPC-W também foi projetado para avaliação de escalabilidade. A ideia principal é manter a proporção entre a carga que está no *site*, o número de EBs simultâneos e o tamanho da base de dados inicial, representado pela cardinalidade de várias tabelas da base. Logo, esta especificação TPC-W suporta um alto número de usuários simultâneos, e a base de dados suporta a quantidade de operações e escala de acordo com a demanda gerada pelos usuários.

Durante a etapa de levantamento sobre *benchmarks* atuais, não foi possível encontrar um que mensure a garantia de QoS (*Quality of Service)*. Entretanto, uma nova metodologia foi definida para *benchmark* de *e-commerce* orientado à QoS,

denominado Bench4Q. Esta pesquisa utiliza como ferramenta de *benchmarking* o Bench4Q, que é uma extensão da especificação TPC-W para sistemas de *e-commerce*, e suporta análise de métricas baseadas em sessões e analisa a QoS da capacidade de carga.

O Bench4Q utiliza o Bench4Q-*Script*, que é uma ferramenta projetada para este *benchmark* e oferece uma maneira conveniente de configurar os testes distribuídos e analisar os resultados. Com esta ferramenta, o usuário pode gravar o *script* de requisições via protocolo HTTP e utilizá-las para teste [5].

O Bench4Q-Script é composto de três partes: (1) *console*, (2) agente e o (3) SUT (*System Under Test*), conforme Figura 4.

O console é o responsável por implementar o agente de toda a carga gerada de acordo com os requisitos dos usuários, controlando o processo de *benchmarking* e exibindo resultados. O agente, implementado e com respectivas cargas geradas, depois de ser inicializado gera mais cargas de acordo com configurações e resultados e informações em tempo real coletadas do *console*. Ao mesmo tempo, o *console* pode monitorar este estado e adquirir isto da base de resultados. Finalmente, o SUT é o sistema a ser analisado, como uma aplicação de *e-commerce.* Tradicionalmente e assim como definido na especificação TPC-W, esta parte é estruturada por recursos computacionais que contém a base de dados, o servidor *web*, servidor de aplicação, entre outros [5].

Como requisitos para executar este *benchmark*, é necessária a instalação de uma *Java Virtual Machine* (JVM), pelo motivo do Bench4Q-Script tem sido escrito na linguagem de programação Java, um sistema operacional baseado em Linux, e o serviço SSH *(Secure Shell),* normalmente nativo na distribuição do sistema operacional escolhido para a implementação.

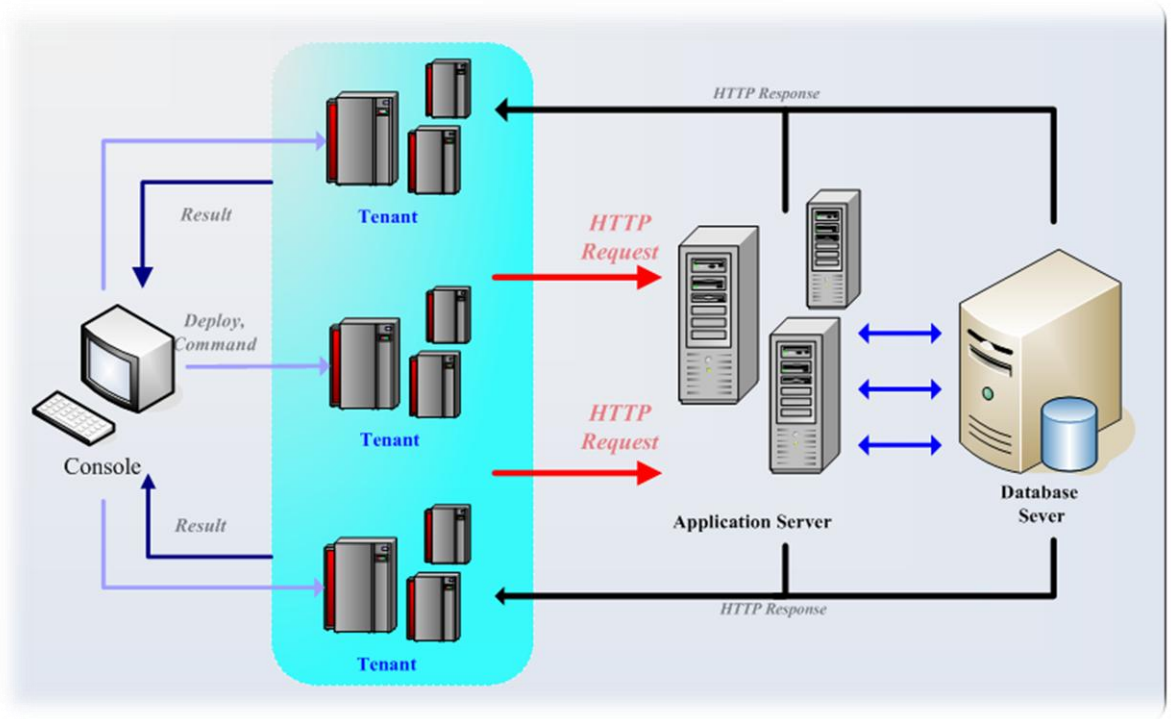

Figura 4: Composição do Bench4Q-Script

RESULTADOS

De acordo com o projeto de pesquisa apresentado, pretende-se avaliar o desempenho de aplicações de *e-commerce*, utilizando uma abordagem orientada à escalabilidade. Para isso, é necessário um ambiente e ferramentas que sejam adequadamente implementados e implantados, para que seja alcançado um resultado satisfatório. Com fatores resultantes, espera-se a definição e execução de procedimentos experimentais para a reprodução dos testes. Todo o processo realizado e procedimentos experimentais terá o intuito de ser ajustável para outras aplicações, desde que sejam do mesmo cenário, protocolo utilizado e que sejam escaláveis.

Durante a pesquisa foi realizada uma investigação sobre as ferramentas e especificações de *benchmark* para análise de desempenho de aplicações de *e-commerce*, definindo um conjunto de ferramentas e resultados favoráveis e realísticos, que resultam em uma metodologia aplicável em outras aplicações a serem avaliadas, assim como execução de experimentos e testes de controle. O aprimoramento dos métodos e revisão bibliográfica específica sobre cada ferramenta foi essencial para que haja agilidade durante a implantação e implementação, assim como conhecimento sobre a especificação TPC-W e suas deficiências, podendo supri-las com outras ferramentas complementares e extensões como o Bench4Q, resultando em procedimentos concisos e análise realística sobre a escalabilidade do SUT.

Como resultados da experimentação e execução de testes de controle, pode-se observar os gráficos gerados que foram organizados como na Figura 5, com quatro gráficos por figura, sendo que nas colunas são definidas os níveis da carga (Médio ou Alta) e nas linhas estão definidos a quantidade de servidores de aplicações (1 ou 2).

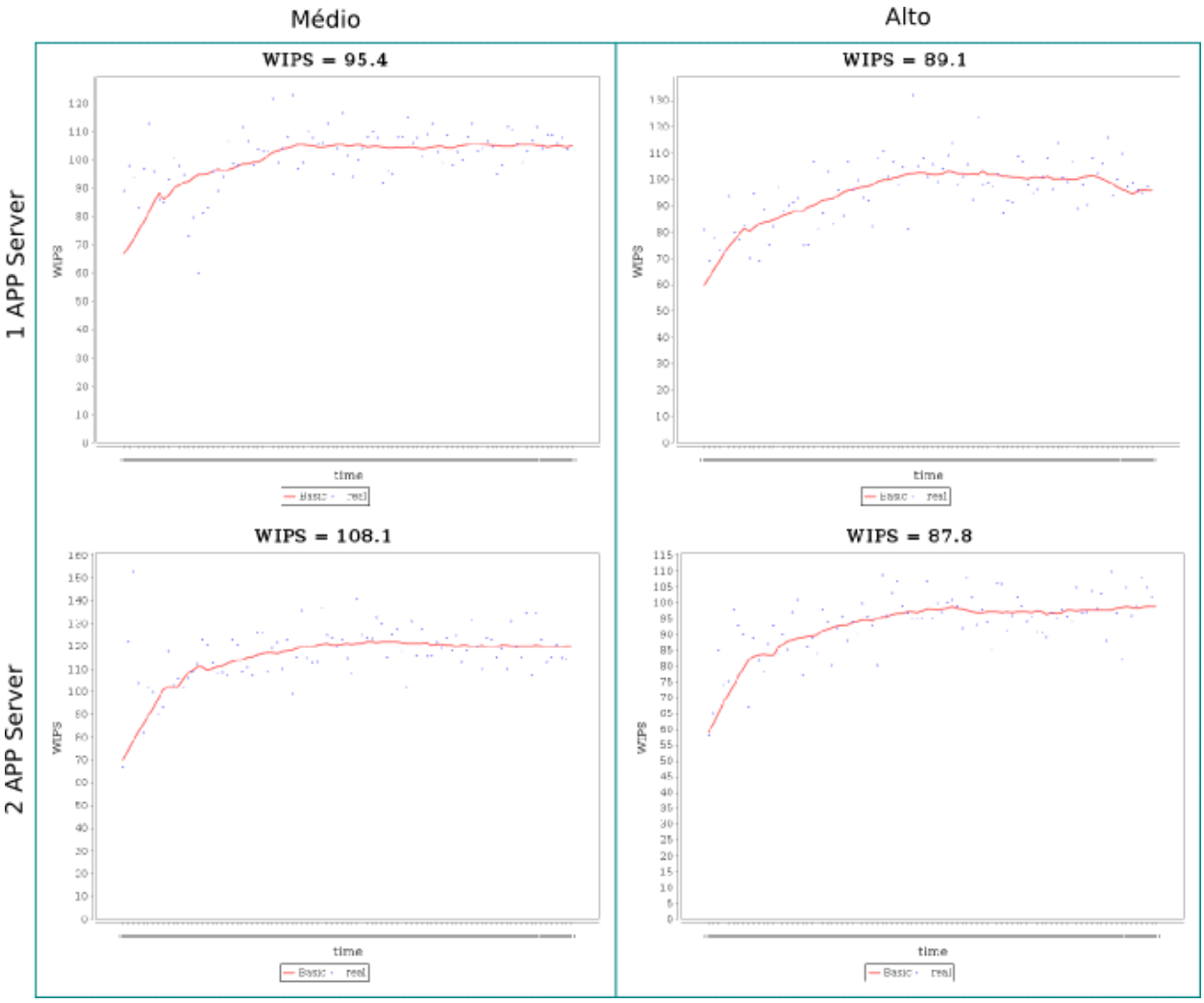

Figura 5: WIPS (Web Interactions Processed per Second) são computadas como o total número de interações Web solicitadas e concluídas com êxito dentro de um intervalo de medição dividido pelo comprimento do referido intervalo em segundos.

A Figura 6 apresenta o tempo de resposta das interações durante o processamento de compras no SUT.

Os menores tempos de resposta foram apresentados na execução com carga Médio, sendo que quando executado com um servidor a media computada foi de 3660,4ms, enquanto que 3497,8ms com dois servidores. Com a carga alta, a média computada para execução em um servidor foi de 6624,2ms e de 7839,9ms para execução com dois servidores de aplicação. No entanto, deve ser observado que na execução com um servidor houveram tempos de resposta de até 40s enquanto que com dois servidores o maior tempo computado foi de aproximadamente 15s.

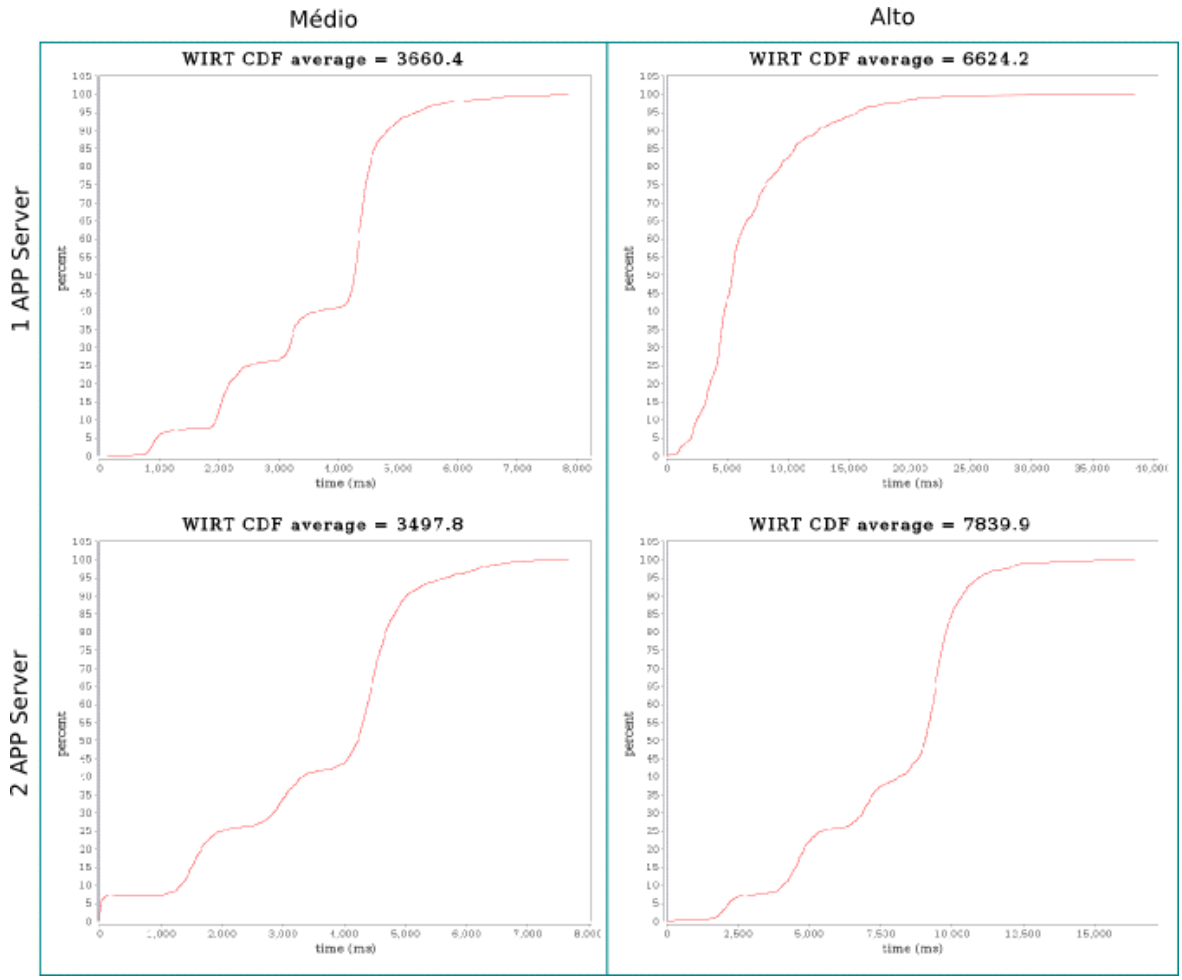

Figura 6: WIRT Order. Tempo de resposta durante o processamento de compras no SUT.

Os gráficos de VSPS (*Valid Session per Second* – Sessões Válidas por Segundo) são apresentados na Figura 7 . TSPS é o total de sessões computadas por segundo e VSR é a taxa de sessões válidas esse valor é uma aproximação do resultado da divisão de VSPS por TSPS.

Com a carga Médio, foram computadas 99% das sessões como válidas, tanto na execução com um servidor quanto na execução com dois servidores. Já na execução com carga Alto na execução com um servidor, apenas 22% das sessões são válidas. Porém na execução com dois servidores, assim como na execução com carga Médio, é computado 99% das sessões como válidas.

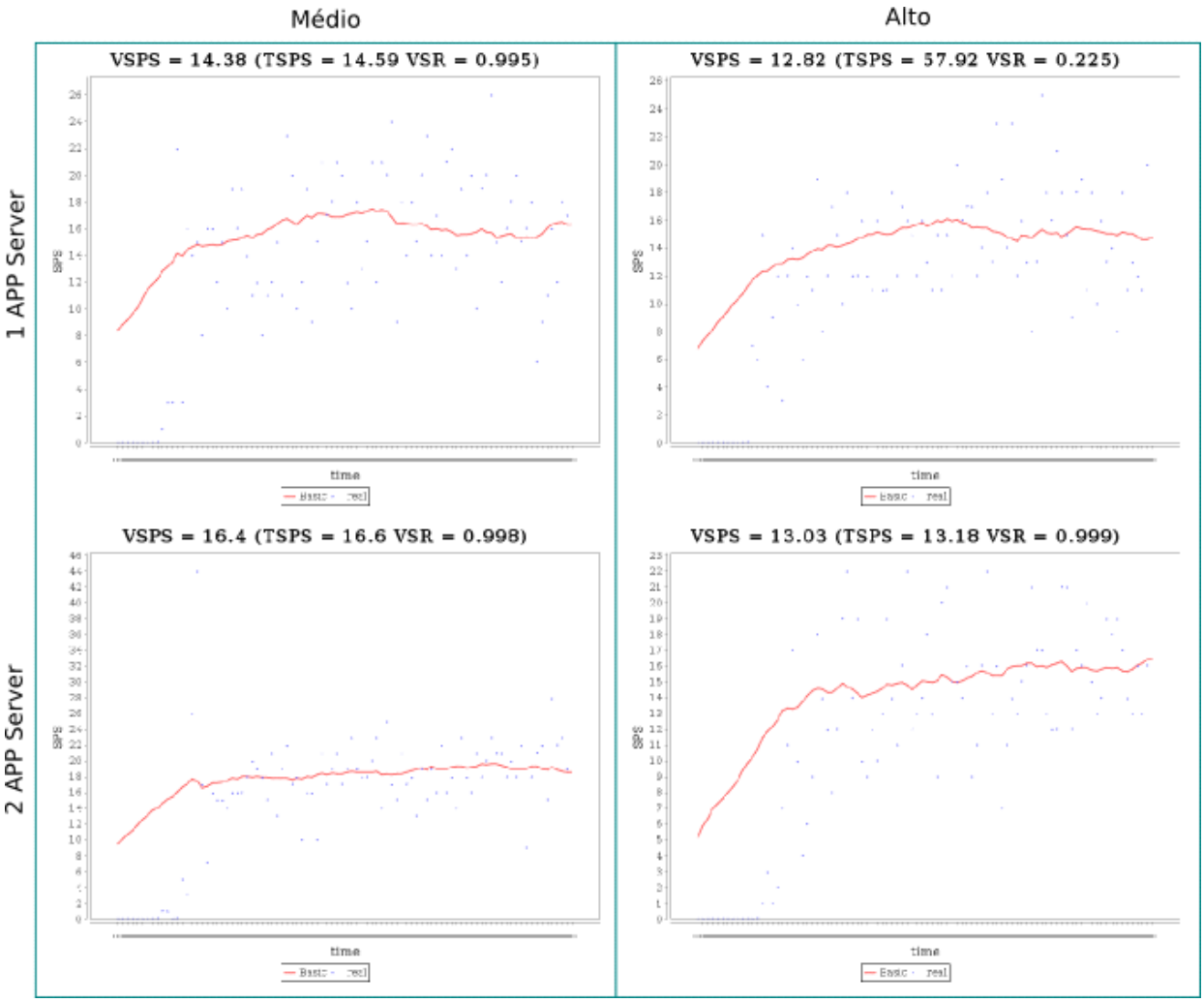

Figura 7: VSPS (*Valid Session per Second*), TSPS é o total de sessões computadas por segundo e VSR é a taxa de sessões válidas.

Na Figura 8 é apresentada a distribuição da duração da sessão, que é dada pelo número da sessão dividido por sua duração. É uma distribuição exponencial com expoente negativo.

Como pode ser percebido em ambos os gráficos, apesar das diferenças dos valores as primeiras colunas apresentam valores mais altos e esses valores vão diminuindo a medida que avança na duração da sessão.

A Figura 9 é composta pelos gráficos dos sumários das seções. A coluna em azul representa o total de sessões completas. A coluna em verde são as sessões que apresentaram ganho - esta coluna não está presente da execução com carga Alto e apenas um servidor de aplicações - . A coluna em vermelho são as sessões que apresentaram erro. É perceptível a presença desta coluna apenas no gráfico

com a carga Alto e com apenas um servidor de aplicações. Neste mesmo gráfico é percebido que os erros ocorreram na página inicial da aplicação (*home*).

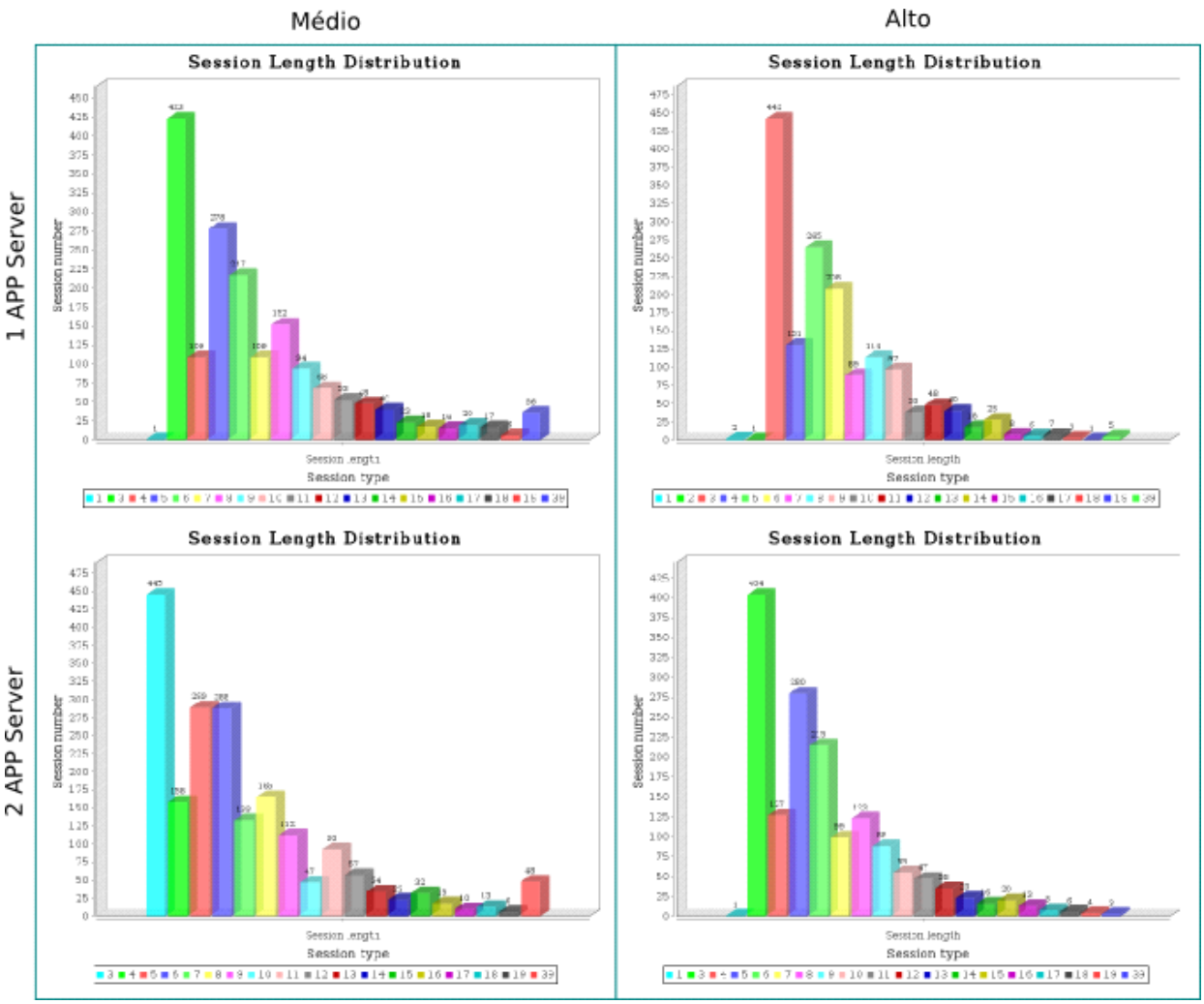

Figura 8: *Session lenght distribuition*.

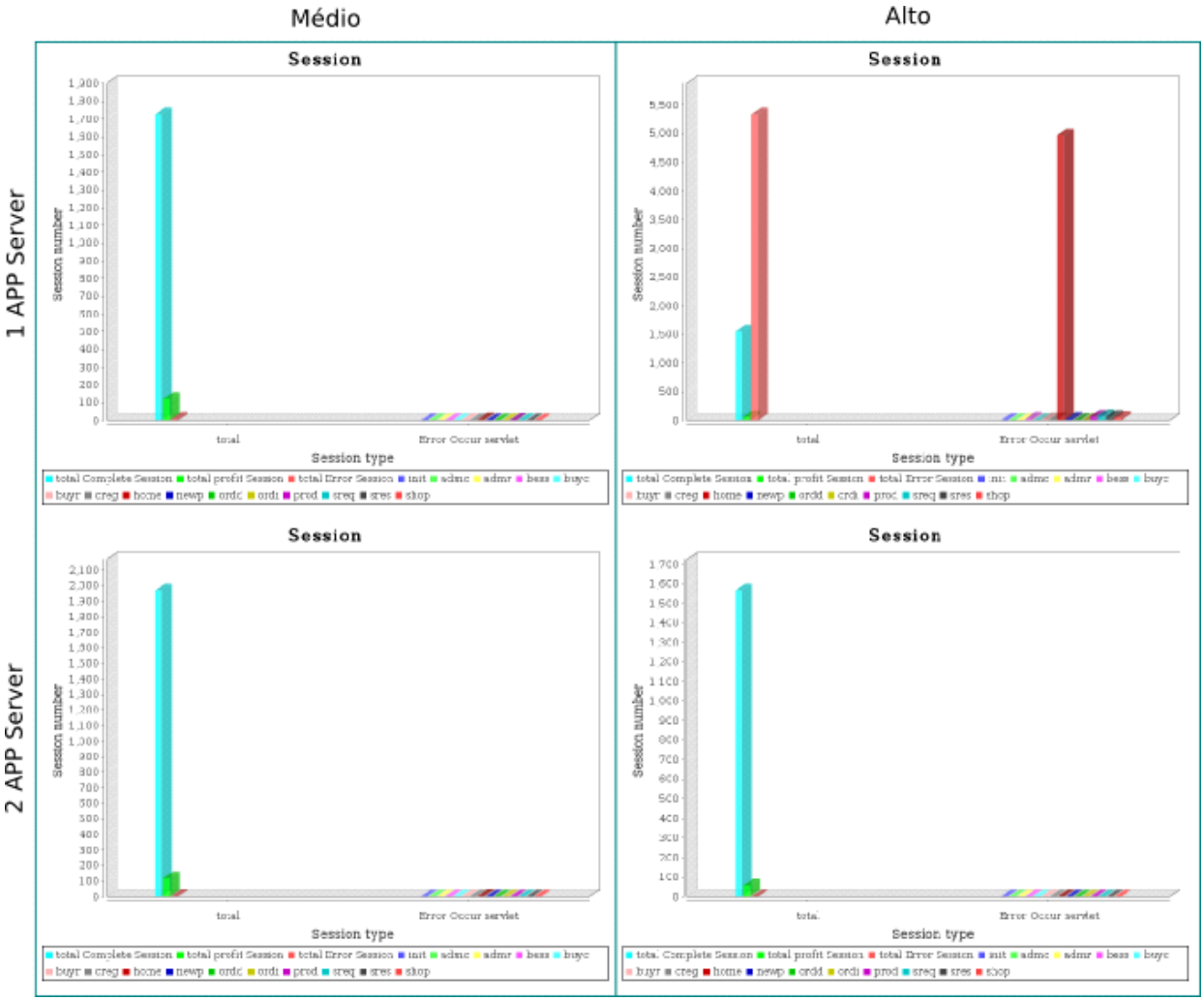

Figura 9: Sumário das seções.

DISCUSSÃO

Um *benchmark* pode auxiliar de forma efetiva a resolução de vários problemas relacionados à recursos computacionais, baixo desempenho ou insuficiente escalabilidade de uma aplicação *web*. A infra estrutura de um *e-commerce* é tipicamente composta de três camadas em sua arquitetura: (1) o servidor *web*, (2) o servidor de aplicação e (3) o servidor de dados. Frequentemente são encontrados problemas como sobrecarga, quando o volume de requisições por transações excedem a capacidade do e-commerce, e responsividade, quando a falta de tempo de resposta adequado diminui a utilização do e-commerce, e consequentemente, reduz lucros. [1]

Conforme avaliação realizada no artigo *Controlling the Performance of 3-Tiered Web sites: Modeling, Design and Implementation* [1], estes problemas podem ser resolvidos por meio de análise aplicando o TPC-W, como por exemplo limitar o tempo de resposta e prevenir a sobrecarga, utilizando uma controladora proporcional integral que se auto ajustará. Justamente pelo auto ajuste, o sistema automaticamente se adaptará há variações de carga e precisará configurar apenas alguns parâmetros. Neste caso, nenhuma alteração no servidor é necessária pois a implementação foi realizada em um *proxy*, com uma aplicação padrão utilizando o sistema operacional Linux, com servidor HTTP Tomcat e o gerenciador de banco de dados relacional MySQL.

Ao utilizar o TPC-W, observa-se que o método atinge comportamento estável e responsividade durante a sobrecarga da aplicação. Com isso pode-se assumir que um controlador adequadamente projetado e implementado pode ser utilizado em um ambiente complexo. Por exemplo, ao utilizar o *TinyProxy* e modificá-lo para agir como o controlador, pode-se redirecionar todas as requisições feitas via protocolo HTTP do cliente para este *proxy*, que consequentemente envia para o servidor web, servidor de aplicação, e finalmente, o controlador. Todas as respostas também trafegam pelo *proxy*.

Este experimento empregando a especificação TPC-W utilizou cinco máquinas, sendo três como clientes gerando requisições, uma do servidor *web* e servidor de aplicação, e outra como o banco de dados e a controladora proporcional integral, dentro do *proxy*. Os clientes carregam as cargas, a taxa de transferência aumenta até a carga atingir determinado limiar, depois a taxa de transferência cai e

o tempo de resposta aumenta [1]. Com testes e avaliações, tal experimento provou que o método controlador projetado consegue limitar o tempo de resposta e manter um nível de taxa de transferência com rendimento significativo mesmo com níveis de cargas excessivos, conforme Figura 10.

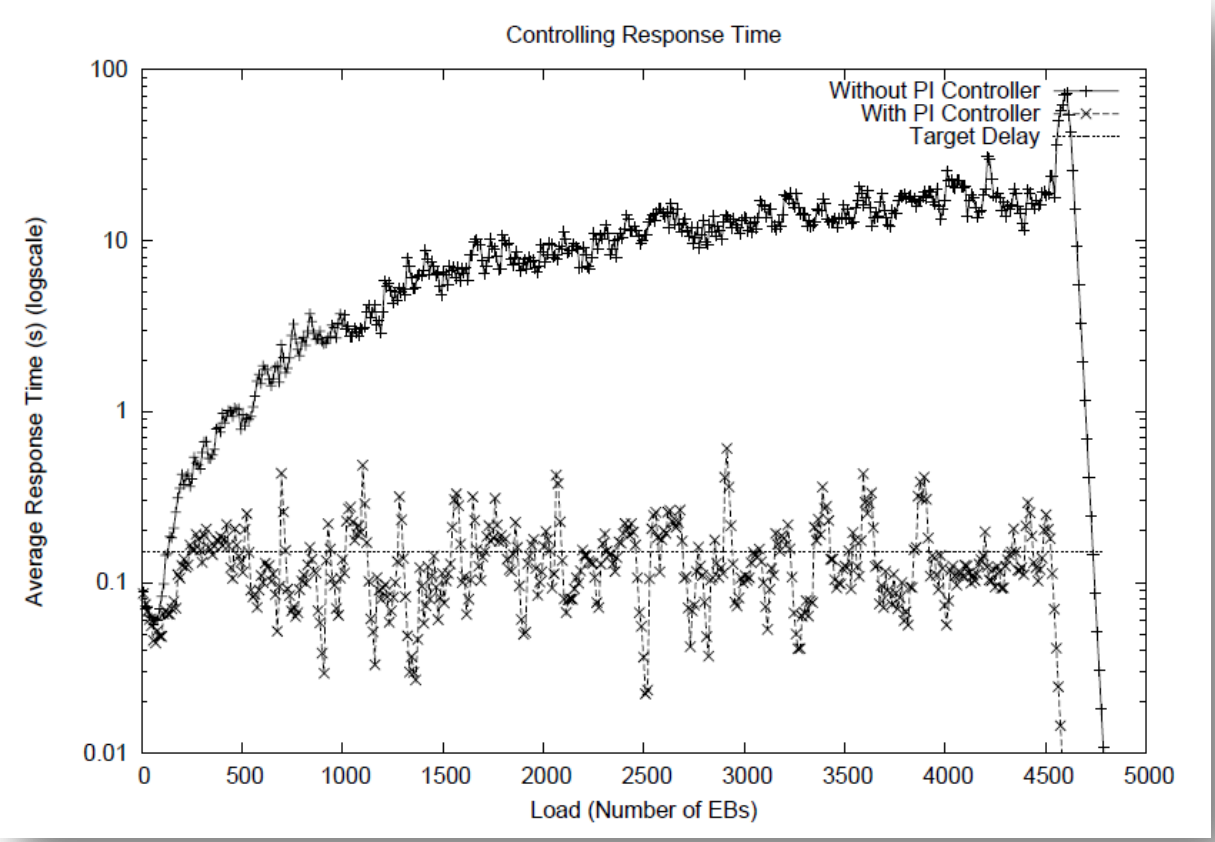

Figura 10: Tempo de resposta do controlador proporcional integral projetado para o experimento [1].

Como qualquer especificação, o TPC-W não se aplica precisamente em outros sistemas de *e-commerce* como aplicações de leilão, que são distintas de uma loja virtual de livros. O TPC-W ignora a presença de robôs em uma carga de trabalho de um e-commerce atual, sendo projetado apenas para requisições feitas por humanos. Os resultados são obtidos em um ambiente controlado, com uma conexão de alta velocidade via cabo. Entretanto, usuários acessam *websites* através de conexões via sinal *wireless* e baixas velocidades, por exemplo [2].

A maioria dos *benchmarks* existentes incluindo TPC-W são insuficientes para avaliar os servidores de *e-commerce* com foco em uma definição da Qualidade de Serviço - QoS. Por este motivo, definiu-se como ferramenta de *benchmark* o Bench4Q. Alguns problemas foram encontrados durante a configuração dos componentes da ferramenta escolhida, entretanto foram todos solucionados conforme Apêndice E - Instalação Bench4Q.

Finalmente, ferramentas como o Bench4Q são importantes para planejamento de capacidade e avaliações dentro do escopo do *benchmark* executado, avaliando não somente a aplicação, mas também a escalabilidade crescente e QoS do *e-commerce* [6].

EXPERIMENTOS E TESTES DE CONTROLE

Durante este período da pesquisa foram realizados experimentos e testes de controle, utilizando as ferramentas de *benchmark* selecionadas durante a revisão bibliográfica. A Figura 11 ilustra a disposição dos componentes do ambiente de experimentação. No *host* físico 1 foi instalado o Console e o Agente. No *host* físico 2, através do Vagrant e do VirtualBox, foram configuradas quatro máquinas virtuais, uma para o balanceador de carga (Load Balancer) com o HAProxy, duas (APP Server 1 e APP Server 2) para hospedar a aplicação *web* através do Tomcat e uma para o banco de dados (DB Server).

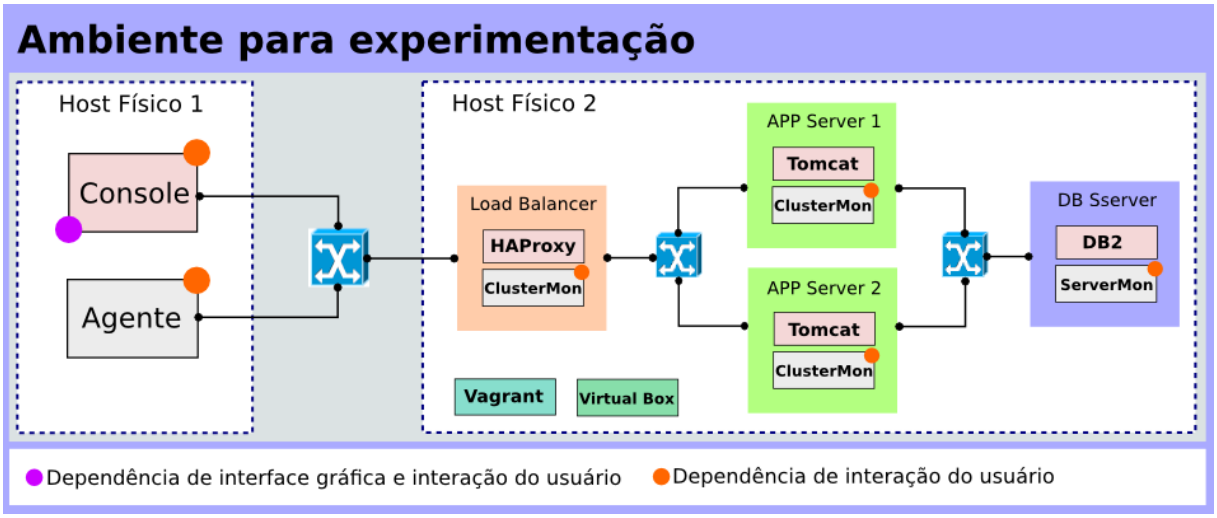

Figura 11: Ambiente para experimentação

Após a instalação e configuração do ambiente, foi realizada a instalação do Bench4Q, a qual está detalhada no Apêndice E.

Para execução do experimento foram utilizados dois computadores, um com processador Intel(R) Atom(TM) CPU 330 @ 1.60GHz, 512 MB DDR 2 de memória física e um *pendrive* SanDisk - Cruzer Blade com 16 GB com Debian 7.0 "Wheezy" onde ficaram instalados o Console e o Agente e o outro computador com processador AMD Athlon(tm) 64 X2 Dual Core Processor 4400+ 1.00 GHz, 2 GB DDR2 de memória física e HD Seagate - Barracuda 7.200 RPM com 40 GB com sistema operacional Ubuntu Server 14.04 LTS, serviu para hospedar quatro máquinas virtuais onde foram instalados os componentes que compõe o SUT.

A Tabela 1 dá detalhes configuração dos hosts físicos e a Tabela 2 dá detalhes da configuração de cada uma das máquinas virtuais utilizadas durante a execução dos experimentos.

Propriedade	Host 1	Host 2
Processador	Intel(R) Atom(TM) CPU 330 @ 1.60GHz	AMD Athlon(tm) 64 X2 Dual Core Processor 4400+ 1.00GHz
Cache	512 KB	512 KB
Memória	512 MB	2 GB
Swap	600 MB	2 GB
HD	SanDisk - Cruzer Blade com 16 GB	Seagate - Barracuda 7,200 RPM com 40 GB
Sistema operacional	GNU/Linux Debian 7.0 "Wheezy"	GNU/Linux Ubuntu Server 14.04 LTS
Kernel	3.2.0-4-686-pae	3.13.0-32-generic
Sistema de virtualização	--	VirtualBox
Software provisionamento	--	Vagrant

Tabela 1: Configuração de Hosts Físicos

Propriedade	VM1	VM2	VM3	VM4
Processador	1 VCPU (32bits)	1 VCPU (32bits)	1 VCPU (32bits)	1 VCPU (32bits)
Memória	256MB	256MB	256MB	768MB
Sistema operacional	GNU/Linux Ubuntu Server 14.04 LTS	GNU/Linux Ubuntu Server 14.04 LTS	GNU/Linux Ubuntu Server 14.04 LTS	GNU/Linux Ubuntu Server 14.04 LTS
Kernel	3.13.0-44-generic	3.13.0-44-generic	3.13.0-44-generic	3.13.0-44-generic

Tabela 2: Configuração das máquinas virtuais

Em ambos os computadores foi feita a instalação padrão dos sistemas operacionais. Para as máquinas virtuais, foi utilizada a imagem de instalação fornecida com o Vagrant “ubuntu/trusty32”. A máquina virtual 1 foi configurada como balanceador de carga, utilizando o HAProxy; A máquina virtual 2 e a máquina virtual 3 foram configuradas como servidores de aplicação *web*, fazendo uso do Tomcat 7 e a máquina virtual 4 foi configurada como servidor de banco de dados utilizando o DB2 v9.7.0.5. Em ambos os sistemas foi feita a instalação da JDK padrão em cada sistema.

Para execução do experimento, primeiro houve a identificação das métricas e dos fatores. Foram criados quatro configurações de experimento para visualizar o número de seções atendidas pelo SUT em um ambiente com dois servidores de aplicação e um balanceador de carga em relação à diferentes níveis de carga controlada.

Como métricas foram identificadas WIPS (Web Interactions Processed per Second), WIRT (Web Interaction Response Time), taxa de utilização da CPU, consumo de memória, taxa de utilização de disco leitura/escrita e taxa de utilização da rede incoming/outgoing. Como fatores, foram identificados a característica da carga: *browsing* (*read-only*), *shopping* (*read/write*) e *ordering* (*write*); o tempo de execução: valor em segundos; o números de servidores de aplicação: 1 ou 2; comportamento dos EBs (Emulated Browser): *open* (rajada) ou *closed* (utilização constante); número de Agentes; número de EBs por Agente: Médio ou Alto.

A identificação de valores numéricos para Médio e Alto ocorreu base em experimentos. O valor para Alto foi baseado em experimentações no próprio ambiente de teste, utilizando apenas um servidor de aplicações. A primeira experimentação foi com o valor 100 (padrão do Bench4Q), e como não houve erro no atendimento das sessões e a carga de CPU foi moderada , foi realizado um novo teste com valor 1000 onde poucas sessões foram atendidas e a carga da CPU ficou próxima de 100%. Em um novo teste com 500, ou seja, metade do valor do teste anterior, todas as sessões foram atendidas e a carga da CPU ficou em média 85%. No último teste realizado com 600, 15% das requisições foram atendidas a CPU ficou em torno de 94%, e dessa forma Alto ficou definido com o valor 600 EBs. O valor para médio foi dado como metade do valor de alto, sendo que para isso utilizou-se a seguinte fórmula:

$$\frac{Alto}{2}$$

Que resultou no valor 300.

O tempo para execução dos experimentos também foi identificado com base em experimentação. Com o valor padrão do Bench4Q, 60 segundos, foi observado muita variação nas curvas dos gráficos e com uma tendência de assentamento. Foi realizado um novo teste com o dobro do valor padrão 120 segundos e com esse valor após as variações percebidas na execução anterior,

houve o assentamento da curva que se manteve até o final da execução. Em um novo teste com valor 240, ou seja o dobro do valor anterior, o comportamento foi similar ao do teste com 120 com diferença apenas na duração. Por isso foi definido 120 segundos como o valor para execução dos testes.

Os fatores com valor fixo para execução dos experimentos são apresentados na Tabela 3 e a configuração do experimento ficou como apresentado na Tabela 4.

Fator	Valor
Característica da carga	Shopping
Tempo de execução	120 segundos
Comportamento dos EBs	Closed
Número de Agentes	1

Tabela 3: Fatores com valor fixo.

Servidores de Aplicação	#EBs
1	Médio
1	Alto
2	Médio
2	Alto

Tabela 4: Configuração dos experimentos.

Para confirmação dos resultados, foram executadas três réplicas para cada configuração do experimento. Após cada replica, todos os sistemas do ambiente foram reiniciados. Em média o tempo de execução das três réplicas de cada configuração é de 52 minutos.

Para a execução deve-se seguir a seguinte seqüência, resumidamente:

- Iniciar o monitor no servidor de banco de dados
- Iniciar o monitor no servidor de aplicações
- Configurar a carga no console
- Executar o experimento
- Coletar os dados de execução do experimento

Para iniciar o monitor deve-se acessar o diretório *ServerMon-1.0* e em seguida executar o *script run.sh*. Será exibida um lista numerada de endereços IPs presentes na máquina e deverá ser fornecido o número referente ao IP que será utilizado para fazer o monitoramento. Para configurar a carga, primeiramente deve-se configurar o IP onde esta o SUT. Para esta configuração deve-se utilizar os

campos destacados na Figura 12, onde a seta preta no menu lateral aponta o item *"Load Simulator"* que exibe a tela de configuração dos endereços do SUT e do servidor de banco de dados. Os retângulos destacam os campos que devem ser alterados. No campo que contem o retângulo com indicação "1", deve-se adicionar a URL para acessar o SUT. No campo do retângulo com indicação "2", não é preciso altera o valor, bastando apenas habilitar o monitor através do *checkbox*. No último campo destacado com indicação "3", deve-se habilitar o monitor através do *checkbox* e onde está *"Database Server Address:"* deve ser preenchido com o IP do servidor de banco de dados.

Depois de configurar os endereços IPs do SUT e do servidor de banco de dados, deve-se configurar a carga de acordo com a quantidade de EB e também o tempo que o teste será executado. A Figura 13 destaca os campos que devem ser alterados, e os retângulos estão dispostos de acordo com a ordem que deve ser conduzido.

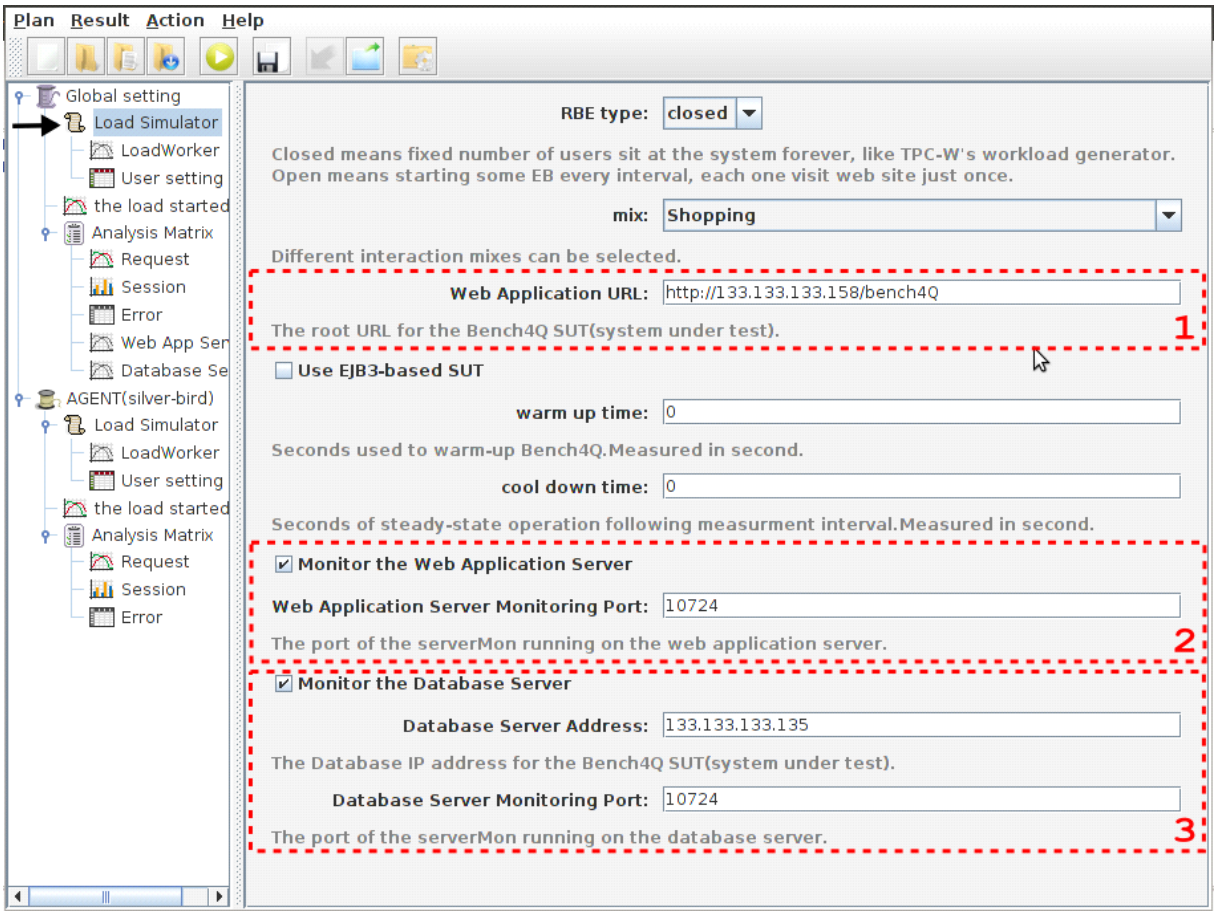

Figura 12: Campos para configuração do IP do SUT e do servidor de banco de dados.

Na Figura 12 a seta preta aponta o item de menu “*LoadWorker*” que exibe a tela de configuração. O retângulo com número “1” destaca o campo da quantidade de EBs que devem ser utilizados na carga. No retângulo com numero “2”, deve ser configurado o tempo de execução do experimento. Após a configuração da carga, pressione o botão para iniciar o teste, destacado pelo retângulo de número “3”.

Ao término do período estipulado será habilitado um botão para coleta dos dados. É aconselhável aguardar cinco segundos (tempo mínimo para a estabilização do sistema), e após esse intervalo coletar os dados.

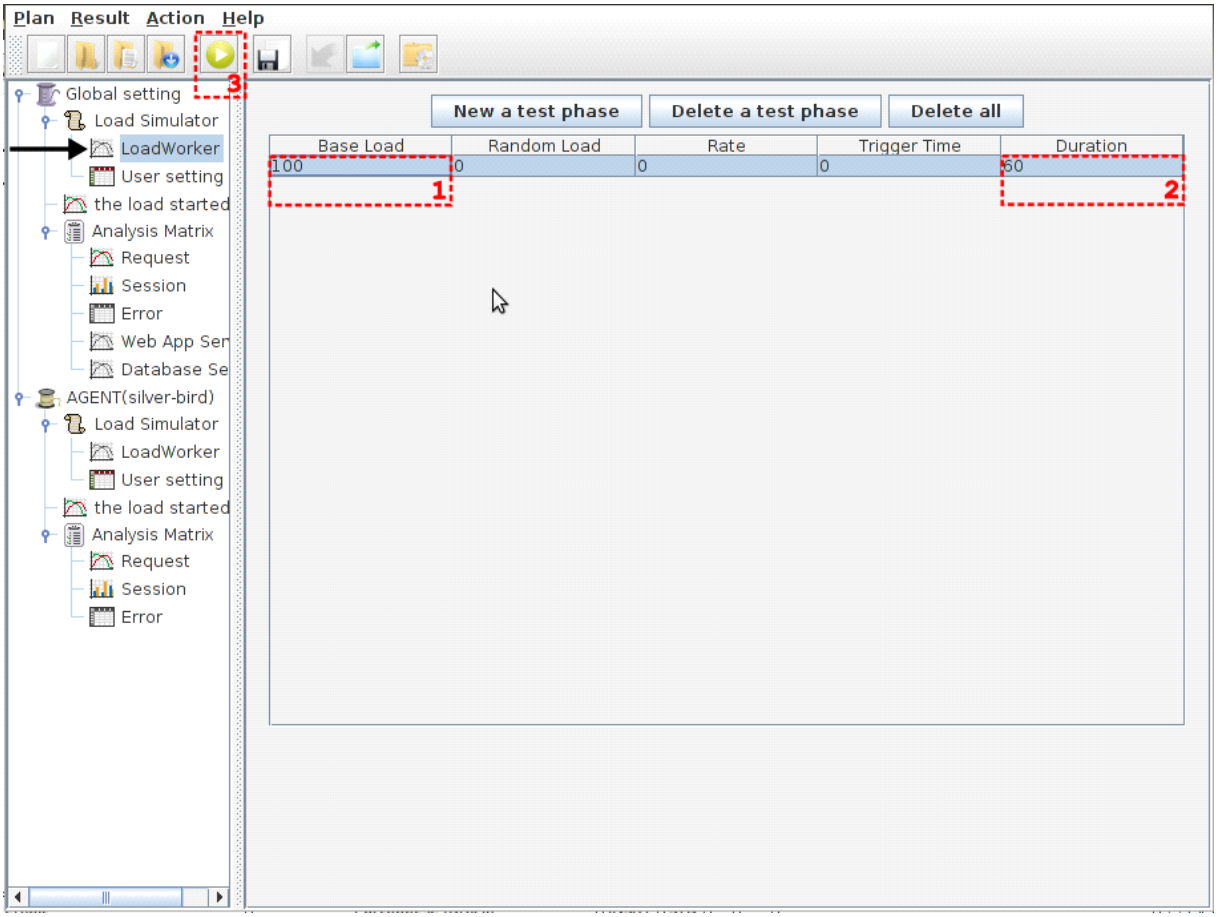

Figura 13: Configuração da carga e execução do experimento.

Para não haver divergência na condução dos experimentos, foi definido o seguinte protocolo de execução:

1. Ligar Host Físico 1
2. Iniciar VMs através do Vagrant
   2.1. DB Server

2.2. APP Server 1
2.3. APP Server 2
2.4. Load Balancer

3. Iniciar Host Físico 2
   3.1. Iniciar Console
   3.2. Iniciar Agente
   3.3. Configurar no Console URL do SUT
   3.4. Configurar no Console IP do DB Server
   3.5. Configurar no Console a carga
4. Iniciar ServerMon no DB Server
5. Iniciar ClusterMon no LoadBalancer
   5.1. Especificar ClusterMon como *leader*
6. Iniciar ClusterMon no APP Server 1
   6.1. Especificar como não *leader*
7. Iniciar ClusterMon no APP Server 2
   7.1. Especificar como não *leader*
8. Iniciar experimento através do console
9. Coletar os dados da execução (esperar 5 segundos para o ambiente se estabilizar)
10. Reiniciar o ambiente

CONCLUSÃO

Estes experimentos foram executados apenas com o intuito de conhecer a ferramenta Bench4Q e os resultados produzidos são insuficientes para concluir como um ambiente de produção deve ser configurado. Foi percebido que a execução dos testes consomem um longo período de tempo e que pelo motivo dos componentes do Bench4Q serem interativos, não é possível automatizar os testes.

Tendo em vista que ele possui o código aberto, uma proposta de trabalho futuro poderia ser a alteração dessa característica dos componentes para serem iniciados com os parâmetros de execução do experimento, o que permitiria automatizar sua execução.

Todo material produzido está disponível para estudo e aperfeiçoamento no endereço http://github.com/kerollaine/bench4q.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Abhinav Kamra, Vishal Misra, Erich Nahum. **Controlling the Performance of 3-Tiered Web sites: Modeling, Design and Implementation.**

[2] IEEE INTERNET COMPUTING. Daniel A. Menascé - George Mason University. **TPC-W A Benchmark for E-Commerce.** Junho de 2002.

[3] **Grid computing**. Disponível em: < http://www.ibge.gov.br/confest_ e_confege/pesquisatrabalhos/CD/palestras/368-1.pdf>. Acesso em: 20.dez.2014

[4] Ian Sommerville. Software Engineering (9th Ed.). Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2010.

[5] OW2. **Bench4Q-Script 1.0 User Manual**. Agosto de 2011. Disponível em: <http://forge.ow2.org/project/showfiles.php? group_id=345>. Acesso em: 27.dez.2014

[6] IEEE INTERNET COMPUTING. Wenbo Zhang. **Bench4Q: A QoS-Oriented E-Commerce Benchmark**. 2011.

[7] IEEE INTERNET COMPUTING. Daniel F. Garcia. **TPC-W E-Commerce Benchmark Evaluation**. 2003.

APÊNDICE A - INSTALAÇÃO INICIAL VAGRANT

Para a experimentação foi necessário definir um utilitário que atendesse aos requisitos de gerenciamento e *deployment* de aplicações multicamadas em ambientes virtuais. Para isso, foi realizada uma análise sobre algumas ferramentas correlatas do mercado.

Docker é uma ferramenta utilizada para criar instâncias de aplicações com o conceito de *container,* que é mais leve que uma máquina virtual. Sua função é empacotar a aplicação e fazer o isolamento no ambiente de produção. O intuito do Docker é agilizar o processo de entrega de *software*. Com ele, desenvolvedores e administradores de sistemas podem empacotar, distribuir e executar suas aplicações sem se preocupar com a infraestrutura. O desenvolvedor adiciona a aplicação que desenvolveu dentro de um *container*, sem se preocupar em qual infraestrutura ele irá executa. O administrador de sistemas recebe e o executa, fazendo o *deploy* da aplicação, sem se preocupar com o ambiente onde ele foi criado. Isso possibilita a separação das funções do desenvolvedor e do administrador de sistemas. O Docker possui o Docker Hub, um serviço de nuvem utilizado para compartilhar aplicações.

Puppet é um sistema de gerenciamento de configuração para infraestrutura de tecnologia da informação que possibilita definir uma configuração para infraestrutura, a qual posteriormente será automaticamente aplicada. Ele trabalha com *fact*, informação de configuração de hardware, sistema operacional e versão de pacotes de cada nó da infraestrutura. Os *facts* são catalogados em um ponto central (Pupet Master), quando há mudança na configuração de um fact ele é enviado para o nó, onde a configuração é aplicada de forma automática. O Puppet também permite automatizar as etapas do processo de entrega de *software provision* de máquinas, teste, produção e liberação de atualização. É possível a criação de módulos (pacotes autônomos de código e dados) para gerencia de aplicações ou de *facts* que podem ser distribuídos através da Puppet Forge (repositório de módulos).

Vagrant é uma ferramenta para automatizar o provisionamento de máquinas virtuais. Ele provê uma forma facilitada de configuração de um ambiente assim como também a sua reprodutibilidade e portabilidade. O Vagrant trabalha com o *script* Vagrantfile que é escrito utilizado a sintaxe do Ruby, nesse script é inserida a configuração do ambiente e com o comando *vagrant up*. Por meio desse *script* em pouco tempo as máquinas virtuais definidas nele estarão prontas para uso. Ele

permite a integração com ferramentas padrões do mercado (*shell scripts*, Chef, Puppet ou Docker) para automatizar a instalação de software. Por padrão ele trabalha com o VirtualBox, porém pode ser utilizado com o VMware e AWS entre outros. Com o Vagrant podem ser criadas Boxes, imagens de um sistema configurado que podem ser distribuídas através do catálogo público de *box* do Vagrant (*public Vagrant box catalog*).

Entre a Docker, Puppet, Juju e Vagrant optou-se por utilizar o última por questão de afinidade, assim como pela simplicidade da declaração de nós do ambiente, possivelmente relacionado à sintaxe da linguagem (Ruby). Algumas de suas principais características são: automatização da criação de ambientes, redução do tempo para criação de ambientes e utilização do Virtualbox por padrão como virtualizador padrão, bem como o gerenciamento de seu ciclo de vida.

Vagrant é uma ferramenta *open source* utilizada para auxiliar o gerenciamento de ambientes virtuais. Algumas de suas principais características são: automatização da criação de ambientes, redução do tempo para criação de ambientes e utilização do Virtualbox por padrão como virtualizador padrão, bem como o gerenciamento de seu ciclo de vida. Nesta pequisa o Vagrant será utilizado para automatizar a criação de um ambiente para execução de aplicações *web* multicamadas. Esse ambiente é composto por seis VM (*virtual machine* – máquinas virtuais) conforme apresentado na Tabela 5.

VM	Função
loadbalancer	Balanceador de carga. Recebe as requisições e repassa para um dos servidores de aplicação que estiver mais livre.
appserver1	Servidor de aplicação. Hospeda a aplicação de interesse.
appserver2	Servidor de aplicação. Hospeda a aplicação de interesse.
dbmysql1	Servidor de banco de dados.
dbmysql2	Servidor de banco de dados.
Storage	Provê recurso de disco.

Tabela 5: Máquinas virtuais implementadas

Uma vez definido os componentes do ambiente e suas funções, a primeira etapa é instalar o Vagrant e o Virtualbox. Após, deve-se criar um diretório para os projetos do Vagrant. Esse diretório é importante porque armazena configurações e recursos como *scripts* para execução nas VMs.

```
mkdir -p ~/vagrant/projects/test
```

cd ~/vagrant/projects/test

O último passo antes de executar o teste é iniciar o projeto. Para isso, dentro do diretório 'test' que foi criado no passo anterior, deve-se executar o seguinte comando:

vagrant init ubuntu/trusty32

Esse comando criará um arquivo chamado Vagrantfile, nesse arquivo ficam as definições das VMs para este projeto. Para o primeiro teste basta apenas executar o comando para iniciar a máquina virtual.

vagrant up

Após este procedimento, o Vagrant permite a criação e acesso de várias VMs de acordo com configurações do Vagrantfile. Utiliza-se o HAProxy, instalação explicada no apêndice D.

APÊNDICE B - INSTALAÇÃO TOMCAT

Pode-se usar qualquer versão, porém para estes experimentos foi utilizada a versão 7.0.59.

Após o *download*, extrair o pacote “tar.gz” baixado. Para o Tomcat iniciar automaticamente adicionar a seguinte linha ao *script* rc.local. <path> deve ser substituído pelo diretório de trabalho, por exemplo a *home* do usuário que está instalando o Tomcat. Exemplo:

/home/tux/apache-tomcat-7.0.59/bin/startup.sh

Dentro do diretório do Tomcat, existem quatro diretórios que são manipulados em quase todas a instalações. A manipulação pode variar de acordo com cada aplicação. Os diretórios são bin/, conf/, lib/ e webapps/ que são respectivamente os diretórios que contêm os arquivos para execução do Tomcat, os arquivos para configuração, bibliotecas para serem utilizadas pelo Tomcat e por aplicações e por fim as aplicações WEB após o *deploy*.

APÊNDICE C - INSTALAÇÃO DB2

Este apêndice descreve os passos necessários para instalar o DB2, requisito da ferramenta de *benchmark* Bench4Q. Para instalar o DB2 no Ubuntu através do gerenciador de pacotes *apt-get*, deve-se executar a seguinte sequências de passos:

*sed -i 's/^# \(.*partner$\)/\1/g' /etc/apt/sources.list*

echo "deb http://archive.canonical.com/ubuntu lucid partner" >> /etc/apt/sources.list

apt-get update

apt-get install -y db2exc

O usuário padrão é o 'db2inst1'. Para conectar no servidor com um cliente, basta configurar o parâmetro *userid* como 'db2inst1' e o parâmetro *password*.

Acessar o *console* do DB2 com o usuário padrão e criar um banco de dados com o nome 'bench4q'.

create database bench4q

A saída para criação do banco com sucesso devera ser como a seguinte:

DB20000I The CREATE DATABASE command completed successfully.

Para ativar o banco de dados recém criado e ativar todos os serviços necessários, executar:

activate db bench4q

A saída para execução desse comando deverá ser:

DB20000I The ACTIVATE DATABASE command completed successfully.

APÊNDICE D - INSTALAÇÃO HAPROXY

Neste apêndice será explicado como instalar o HAProxy no Ubuntu 14.04 e como configurá-lo para balancear carga entre dois servidores de aplicação.

Primeiro passo, é instalá-lo a partir do repositório. Para isso:

sudo apt-get update

sudo apt-get install haproxy

Após a instalação é preciso configurá-lo para iniciar durante o boot do sistema. Para isso deve-se editar o arquivo "/etc/default/haproxy" alterando o valor da variável *ENABLED* para 1.

Depois deve-se fazer as configurações para o HAProxy no arquivo "/etc/haproxy/haproxy.cfg". Porém antes de modificá-lo é aconselhável fazer uma cópia de segurança. Para isso, como super usuário:

sudo cp /etc/haproxy/haproxy.cfg /etc/haproxy/haproxy.cfg.orig

Após fazer a cópia, primeiramente faça a alteração para trabalhar em camada. Alterar o arquivo para as seguintes configurações:

mode tcp

option tcplog

Em seguida deve-se fazer a configuração do *frontend*. Para isso ao final do arquivo, deve-se cria uma nova seção como a seguinte:

frontend www

bind 192.168.100.210:80

default_backend bench4q

Após o *frontend*, deve-se fazer a configuração do *backend*. Para isso adiciona-se após o *frontend* a seguinte seção:

backend bench4q

balance leastconn

mode tcp

server appserver1 192.168.30.220:8080 check

server appserver2 192.168.30.221:8080 check

Depois de configurado o *frontend* e o *backend*, pode-se configurar a interface administrativa. Este passo não é obrigatório, porém ele é recomendado para ter acesso a dados estatísticos. Ao final do arquivo adiciona-se uma seção como a próxima:

listen admin

mode http

*bind *:8383*

stats enable

stats uri /

Por último habilita-se o log do HAProxy. Para isso edite o arquivo */etc/rsyslog.conf.*

Localize as linhas '#$ModLoad imudp' e '#$UDPServerRun 514'.

Remova o comentário e adicione mais uma linha como abaixo:

$ModLoad imudp

$UDPServerRun 514

$UDPServerAddress 127.0.0.1

Depois de configurando, reinicie os serviços:

/etc/init.d/rsyslog restart

/etc/init.d/haproxy restart

APÊNDICE E - INSTALAÇÃO BENCH4Q

Para executar o Bench4Q 1.3 é necessário ter uma JVM (*Java Virtual Machine*) devidamente instalada e configurada em cada um dos servidores que compõe o ambiente de teste. A execução do Bench4Q consistem em uma série de passos para instalação dos componente, configuração e execução na ordem correta.

O primeiro passo é distribuir seus componentes no ambiente que está sendo testado. Neste projeto será feita a distribuição e instalação a partir do servidor de banco de dados, criado através da instalação e configuração do DB2 - Apêndice C.

Copia-se os pacotes “DB2Generator-1.3.0.zip” e “ServerMon-1.0.zip” para um diretório onde se tenha permissão de escrita e execução. Depois de copiado deve-se extrair o conteúdo deles. Para isso:

unzip DB2Generator-1.3.0.zip

unzip ServerMon-1.0.zip

Depois de extrair o conteúdo, deve-se popular o banco de dados. Para isso, deve-se acessar o diretório 'DB2 Generator', que foi criado na extração do conteúdo do "DB2Generator-1.3.0.zip". Neste diretório há o arquivo *db.properties* que contém o parâmetros de configuração para acesso ao banco de dados. Deve-se fazer uma configuração como especificada abaixo, sendo que onde está “senha” deve ser substituído pela senha do banco de dados. Os demais parâmetros, não precisam ser alterados.

dbname = jdbc:db2://127.0.0.1:50000/BENCH4Q

username = db2inst1

password = “senha”

Após configurar os parâmetros, basta executar o “DBPopulate.jar” no terminal.

java -jar DBPopulate.jar

Essa operação consome um longo período de tempo e dependendo da configuração do servidor, pode levar algumas horas. Após o término da execução do

programa o banco de dados já estará configurado e pronto para receber conexão da aplicação.

O próximo passo é fazer o *deploy* da aplicação *web*. Deve-se copiar o pacote “SUT-1.3.0.zip” e “ServerMon-1.0.zip”, e ambos também deverão ser copiados para um diretório com permissão de escrita e execução. Como mencionado anteriormente, recomenda-se o *home*.

Dentro do diretório SUT criado com a extração do pacote “SUT-1.3.0.zip”, há o pacote “bench4Q.zip” e novamente deve-se extrair seu conteúdo:

```
unzip  bench4Q.zip
```

Será criado o diretório *bench4Q*, e esse diretório deve ser copiado para o diretório *webapps* do Tomcat.

Em sequência deve-se configurar uma conexão global no Tomcat. Para isso no arquivo *context.xml (/<path-tomcat>/conf/context.xml)* dentro da *tag* Context, após *WatchedResource*, adicionar um novo contexto deixando o arquivo como especificado abaixo:

```
<Context>
  <WatchedResource>WEB-INF/web.xml</WatchedResource>
  <ResourceLink name="jdbc/DB21W" global="jdbc/DB21W"
        type="javax.sql.DataSource" />
</Context>
```

No arquivo arquivo *server.xml* (/<path-tomcat>/conf/server.xml) dentro da *tag GlobalNamingResources*, após o último Resource configurado deixar a configuração como:

```
<GlobalNamingResources>
  <Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
```

```
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />
    <Resource auth="Container"
        driverClassName="com.ibm.db2.jcc.DB2Driver"
        name="jdbc/DB21W"
        password="senha"
        username="db2inst1"
        type="javax.sql.DataSource"
        url="jdbc:db2://127.0.0.1:50000/BENCH4Q"/>
  </GlobalNamingResources>
```

Adicione o *driver* do db2jcc.jar dentro do diretório *lib*. Por fim, basta reiniciar o Tomcat.

Para testar a aplicação, basta acessar a URL *http://<host>:8080/bench4Q/home.* Uma página com algumas imagens e algumas categorias deverá ser carregada. Em seguida, basta clicar em uma categoria para ver o conteúdo dela listado em uma tabela.

O último passo é instalar o Console e o Agente. O Console possui interface gráfica e por isso deve ser instalado em um computador onde o sistema operacional tenha interface gráfica. Eles podem ser instalados em computadores diferentes. A instalação consiste em copiar o pacote “Bench4Q-1.3.0.zip” para o diretório com permissões de escrita e execução e extrair seu conteúdo.

```
unzip  Bench4Q-1.3.0.zip
```

Será criado o diretório Bench4Q-1.3, e dentro dele, entre outros, há os arquivos *bench4Q.properties* e *lib/bench4Q.jar*. O arquivo *bench4Q.properties* define para o Agente as configurações de conexão com o Console. Caso estejam em computadores diferentes, deve-se configurar nele o IP do computador onde o Console será executado. No caso de execução local a configuração ficará similar a:

bench4Q.consoleHost=127.0.0.1

bench4Q.consolePort=6372

Por fim basta executar o Console e o Agente, na respectiva ordem.

Durante a instalação dos componentes do Bench4Q, ocorreram situações inesperadas, onde os componentes não funcionaram. Na documentação oficial não há relatos das situações e como conseqüência, não há documentação das soluções.

Segue uma lista das situações:

- Problemas com o *parser* do IP do monitor do servidor de aplicações. Quando o servidor de aplicações foi configurado de forma em que os dígitos dos octetos não estavam todos preenchidos, o console disparava exceções de host não conhecido. A resolução para execução dos testes foi reconfigurar o IP do servidor de aplicações para um IP com todos os dígitos dos octetos preenchidos. Como exemplo, IP 192.168.1.50 não funcionou mas o IP 192.168.100.200 funcionou corretamente.
- Problemas com o '*Locale'*. Como o computador onde o Console estava instalado foi configurado com o idioma Português do Brasil, ao tentar fazer a coleta dos dados de execução era disparada uma exceção de *floating point.* Para solucionar o problema, foi passado de forma explícita no terminal a configuração do *locale* na execução tanto do Console quanto do Agente. Execução do Console:

  java -Duser.country=US -Duser.language=en -cp lib/bench4Q.jar org.bench4Q.Console.

  Execução do Agent:

  java -Duser.country=US -Duser.language=en -cp lib/bench4Q.jar org.bench4Q.Agent bench4Q.properties.

- Nos gráficos não é impresso os valores registrados pelos monitores durante a primeira execução do experimento. Para obter os gráficos do dados registrados durante a primeira execução deve-se fazer uma nova execução. Assim é registrado os dados dos monitores durante a primeira e segunda execução e impresso no gráfico.

- O monitor considera a taxa de utilização da CPU em modo usuário, o que apresenta uma taxa baixa. Quando é feita a análise através utilitários como o *top*, a somatória da taxa dos modos de operação da CPU (*user, nice, system, idle, iowait, etc*.) apresenta alta taxa de utilização de CPU no servidor de aplicações.

Apesar das situações listadas todas foram contornadas de forma relativamente simples o que não inviabilizou a execução dos experimentos.