Um estudo sobre testes de desempenho com aplicação prática utilizando a ferramenta JMeter

Evandro Grezeli de Barros Neves e Regiane Aparecida Marucci

***Resumo*—Este artigo descreve o resultado de um estudo sobre testes de desempenho aplicado teste de desempenho aplicado em uma arquitetura e-commerce hipotética utilizando uma ferramenta de apoio, chamada JMeter. Com as informações geradas como resultado desta execução, foi utilizada como dados de entrada para o desenvolvimento de um aplicativo que apresenta gráficos gerenciais de tempo de resposta da aplicação testada.**

***Palavras-chave* – Teste de *Software*. Teste Não Funcional. JMeter. Desempenho.**

I. INTRODUÇÃO

"Teste de Software é o processo com o intuito de encontrar erros em um programa ou sistema em execução" [1].

Esta definição citada por Myers reforça o motivo pelo qual cada dia mais empresas se preocupam com a qualidade do seu *software*, investindo em grandes equipes para encontrar os defeitos inseridos em tempo de programação, seja por má interpretação da especificação funcional ou falta de mão de obra qualificada.

Por falta de tempo de cronograma de projeto ou por pouca experiência da equipe de desenvolvimento envolvida, as fábricas de *software* se preocupam em entregar o sistema conforme o esperado, porém não se preocupando em como o produto foi desenvolvido. Por estes motivos, muitos dos sistemas hoje em produção, acabam tendo problemas de desempenho e mesmo de segurança, por não ter tido uma validação destes requisitos ainda no seu ciclo de testes.

Diante do que foi exposto, este artigo expõe uma breve descrição das melhores práticas propostas por autores brasileiros e internacionais que servem de apoio para testes de desempenho [2].

II. TESTES NÃO FUNCIONAIS

Os testes não funcionais são verificações da qual o desenvolvedor ou o testador, tem que avaliar se os requisitos não funcionais especificados no começo do projeto estão de fato sendo atendidos. Existem algumas categorias de testes não funcionais, sendo elas [3]:

a) Teste de Usabilidade;
b) Teste Estrutural;
   a. Teste de Performance (Desempenho);
   b. Teste de Configuração (Ambiente);
   c. Teste de Confiabilidade e Disponibilidade;
   d. Teste de Recuperação;
   e. Teste de Contingência;
c) Teste de Compatibilidade (Versionamento);
d) Teste de Segurança;
e) Teste de Instalação;

Os testes de caixa branca[3], consiste em avaliar o comportamento interno dos componentes do software. Diferentemente do teste de caixa preta, que apenas verifica a saída (*output*) em relação à entrada (*input*), o teste estrutural consiste em destrinchar o código fonte, fazendo com que a entrada passe por todos os caminhos lógicos desenvolvidos no *software*, como pode ser visto na figura 1.

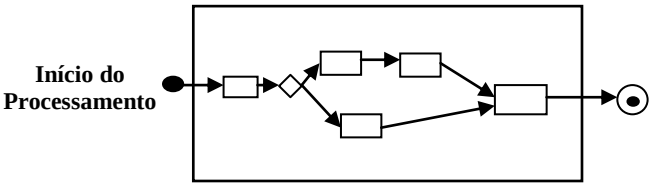

Figura 1. Visão de testes de caixa branca, de Bartié.

Pressman [4], afirma que os defeitos encontrados pelo teste caixa preta não são os mesmos encontrados pelo teste caixa branca. Pelos seguintes motivos:

a) Erros lógicos e pressuposições incorretas são inversamente proporcionais à probabilidade de que um caminho em um programa seja executado. O processamento cotidiano tende a ser bem entendido, enquanto o processamento de um "caso especial" tende a "cair por terra".

b) Muitas vezes acredita-se que um caminho lógico não tem a probabilidade de ser executado quando, de fato, ele pode ser executado regularmente. Ou seja, podem-se cometer erros de projeto que são descobertos somente quando se inicia a atividade de teste do caminho.

c) Erros tipográficos são aleatórios. Quando um programa é traduzido para código-fonte da linguagem de programação, é provável que alguns erros de digitação ocorram. Muitos deles serão descobertos pela verificação de sintaxe do ambiente de desenvolvimento, mas outros passarão sem ser detectados até que os testes se iniciem. É possível que exista um erro tipográfico tanto num caminho lógico obscuro como num caminho da corrente principal.

III. TESTES DE DESEMPENHO

Com o desenvolvimento tecnológico atual e soluções propostas para o mundo coorporativo, a infra-estrutura que dá suporte aos serviços na Web compreende muitos recursos de *hardware* diferentes, desde estações de trabalho dos clientes, servidores com seus processadores e subsistemas de armazenamento, além dos recursos não previstos na

Evandro Grezeli de Barros Neves é graduado como bacharel em Sistemas de informação na Universidade Anhembi Morumbi (UAM). E-mail: egrezeli@gmail.com

Regiane Aparecida Marucci é professora mestre do Centro de Ciência Computacionais da Universidade Anhembi Morumbi (UAM). E-mail: remarucci@anhembimorumbi.edu.br

especificação funcional como *Local Area Network* (LAN), *Wireless Area Network* (WAN), balanceadores de carga e roteadores. Ainda existe a complexidade de processamento de *software* com diferentes servidores de aplicação, camadas *middlewares*, sistemas de gerenciamento de banco de dados, sistemas operacionais e diferentes ambientes arquiteturais se comunicando entre si (Alta e Baixa plataforma).

Todos estes ingredientes somados a uma codificação complexa sem se preocupar com boas práticas de programação, tornam ainda mais complexo definir aonde pode existir um problema de desempenho [5].

Uma requisição feita na Web gasta uma parte do seu tempo em processamento dentro da arquitetura da aplicação e outra parte esperando nas filas pelos recursos.

Para uma aplicação Web, os pedidos feitos podem ser decomposto em duas diferentes áreas [5]:

1. Tempos de Serviço: Tempo gasto usando vários recursos, como processadores, discos e redes.

2. Tempos de Espera: Tempo gasto esperando para usar recursos que estão sendo mantidos por outros pedidos paralelamente.

Para o usuário final, existe uma única percepção, o tempo de resposta. Este, por sua vez, pode ser dividido em dois componentes:

1. Tempo de Rede: Todo o tempo gasto pelas mensagens trafegadas entre o navegador do usuário e o Web Site.

2. Tempo no Web Site: Todas requisições processadas internamente pela arquitetura do Web Site.

Dentro destes dois componentes, ainda existem outros conceitos como podem ser visualizados na figura 2, que demonstram como um tempo de resposta pode ser totalizado.

Figura 2. Complexidade da definição do tempo de resposta, de Menascé e Almeida.

Com um desmembramento do custo do tempo da rede no universo do tempo de resposta, a latência é formada pelo número de ida e volta envolvidos na troca de mensagens entre o navegador do cliente e o site, além do tempo de transmissão, que é o tempo necessário para a transmissão de todos os *bytes*.

O tempo gasto por este mesmo pedido do usuário, na arquitetura do *site* pode ser decomposto em dois componentes principais, conforme indicado na figura 2. O tempo de serviço é o período de tempo durante o pedido que está recebendo serviço de um recurso, como *Central Processing Unit* (CPU), disco, LAN. Este mesmo pedido pode visitar um destes recursos, várias vezes, antes que seja concluído. Com esta frequente troca de contexto do processo no processador, estes acabam sendo enfileirados gerando assim o outro componente chamado de tempo de fila.

Dentro deste cenário de complexidade arquitetural a percepção do desempenho pode se tornar algo subjetivo e que deve estar em acordo com o que é esperado de resposta pelo usuário final da aplicação.

A percepção do desempenho não tem um consenso específico que defina o quão rápido um sistema deva ser principalmente para tempos de resposta em serviços Web.

As métricas de desempenho para processos transacionais na alta plataforma sugeridos por Miller [6] suportam a idéia de um tempo que o homem acredita que sua solicitação está sendo processada, caso a mesma retorne em um tempo inferior a 1 segundo, como pode ser visto na tabela 1, enquanto estudos estatísticos voltados para serviços web da decada de 90 [7] apontam uma regra de que o tempo aceitável para resposta de uma página e ou serviço web até a desistência de um usuário é de 8 segundos.

TABELA 1. TEMPO DE RESPOSTA POR TRANSAÇÕES TRANSACIONAIS

Tempo de Resposta	Comentário
0,1 segundo	É o limite de tempo em que o usuário sente que a aplicação está respondendo imediatamente.
1 segundo	É o limite de tempo para que o fluxo de pensamentos do usuário se mantenha contínuo, apesar de que o usuário notará a demora no processamento.
10 segundos	É o limite máximo de espera para manter a atenção do usuário na tela.

Com a crescente demanda de serviços web e com o aumento da inclusão digital de mais usuários, a regra proposta pelo survey da Zona Research inc [7] já não pode ser aplicada exigindo um ganho de desempenho em grandes corporações com o medo de perda de rentabilidade líquida ou desistência destes usuários implicando em visitas a web sites concorrentes.

Um estudo conduzido por Bhatti, Bouch e Kuchinsky [8], observou como os usuários reagiam quando eles eram direcionados a usar um web *site* para configurar e comprar um computador. O estudo identificou que existia frustração quando o tempo de resposta era por volta de 10 segundos. Por outro lado, essa tolerância diminuía de 10 para 4 segundos quando o processo de configuração de compra do computador estava próximo do fim.

Spool [9] conduziu outro estudo com diversos usuários que deveriam visitar 10 *sites* diferentes e utilizar os recursos dos sites conforme o interesse de cada um. Cada usuário deveria pontuar a velocidade de cada site conforme a sua percepção. Ao final dos testes, percebeu-se que o site considerado o mais devagar de todos pelos usuários era, na verdade, o mais rápido de todos, com um tempo de resposta de 8 segundos. Por outro lado, um site corporativo mundial de compras foi considerado o mais rápido de todos, no entanto, o seu tempo de resposta era de 36 segundos. Concluíram que essa percepção contraditória está ligada aos elementos visuais da página. Em média, os usuários gastam cerca de 4 segundos lendo cada elemento. Este site corporativo oferece vários elementos interessantes do ponto de vista do usuário, dessa forma, após a requisição de uma operação, o usuário gasta vários segundos lendo esses elementos, o que torna a percepção de que o tempo de resposta do site é inferior aos 36 segundos mensurados.

Um estudo mais recente publicado por Barber [10] sugere que acima de 5 a 8 segundos de espera para o usuário já é

considerado frustrante, como pode ser visto na tabela 2, e que o tempo considerado típico para o usuário é de 2 a 5 segundos.

TABELA 2. TEMPO DE RESPOSTA POR SERVIÇOS WEB

Tempo de Resposta	Comentário
Abaixo de 3 segundos	Resposta imediata
De 2 a 5 segundos	Tipico
De 5 a 8 segundos	Lento
De 8 a 15 segundos	Frustrante
Acima de 15 segundos	Inaceitável

Diante de um cenário subjetivo que depende única e exclusivamente da aplicação Web e do que é esperado da mesma em termos de desempenho, o consenso comum para avaliar o desempenho de uma aplicação é através do tempo de resposta e a taxa de processamento da aplicação [11].

Além destas duas métricas principais para medição de desempenho, ainda é possível utilizar-se de outras para verificar o desempenho dos serviços na web, sendo elas:

a) Tempo de Resposta de ponta a ponta: A soma total do tempo da rede ao tempo de processamento.

b) Tempo de Resposta do site: O valor do tempo de espera da requisição do usuário ser completamente atendida.

c) Erros por segundo: Qualquer falha no processo de interação com o servidor, como, por exemplo, um estouro na fila de conexões de usuários ignorando todas as requisições que possam ser feitas.

d) Visitantes por dia: Demonstra a capacidade atual da aplicação sem degradação do tempo, sendo possível ter o cenário esperado por tempo de desempenho dos serviços prestados.

e) *Hits*: Um único pedido de página pode gerar vários *hits*, dependendo dos tipos de arquivos incluídos na página *Hypertext Markup Language* solicitada.

Uma maneira de determinar qual será a experiência do usuário utilizando os serviços web é através dos testes de desempenho. O objetivo principal de um teste de desempenho é entender qual o desempenho do serviço sob condições de carga de trabalho que reflitam a realidade ou uma projeção da mesma.

É importante que estes testes sejam planejados com antecedência e com presença imprescindível da equipe de desenvolvimento, produção e planejamento de capacidade. A chave para obtenção de sucesso nestes tipos de teste é a simulação fiel do ambiente de produção e cenários de carga de trabalho, de modo que os resultados dos testes coincidam com o mundo real da melhor maneira possível e caso a aplicação ainda não esteja disponível ao usuário final já possa ser implantada com a verificação e com o conhecimento dos limites computacionais da mesma.

Os testes a serem aplicados devem incluir situações que representam uma atividade típica do sistema e situações de pico, de modo a determinar o comportamento do serviço em ambas às situações. Basicamente podem ser definidos três tipos de testes de desempenho caracterizados pela intensidade de carga gerada [5]:

a) Teste de Carga: É criada uma carga simulada imitando o cenário de demanda típica do sistema, para saber como a aplicação se comporta.

b) Teste de esforço: Como o sistema se comporta sob condições extremas (situação atípica), utilizando os cenários de pior caso utilizando a carga mais pesada do que o esperado. Este tipo de teste também é conhecido como Teste de *Stress* [3] [12].

c) Teste de pico: Testado em condições muito específicas, quando a carga é muitas vezes maior do que a média e em um pequeno espaço de tempo. Este tipo de teste também é conhecido como *Spike Test* [12].

Além destas definições básicas de testes de desempenho, é possível citar testes mais específicos a serem feitos dependendo do resultado esperado pelo usuário [12].

a) *Capacity Test*: Este tipo de teste é destinado para obtenção de resposta sobre qual a capacidade da arquitetura no seu formato original e o quanto é necessário ela crescer dependendo da demanda desejada de usuários. Uma técnica estratégica voltada a determinar um crescimento de arquitetura.

b) *Component Test*: Teste de desempenho voltado a uma única camada da aplicação, cujo objetivo é definir o quanto a mesma isoladamente atende a demanda solicitada e qual o seu tempo de resposta. Teste isolado utilizado para identificação de gargalos específicos da arquitetura.

c) *Endurance Test*: O objetivo deste teste é analisar como a arquitetura se comporta sobre condições contínuas de cargas normais de usuários em um período de tempo superior ao real.

A partir do estudo teórico realizado, sobre os tipos de testes de desempenho em suas diferentes nomenclaturas citadas por Menascé [5] e para os testes de desempenho mais especíifico citados por Meier et al. [12], obtevese assim, os seguintes tipos. Teste de Carga, Teste de esforço, Teste de Pico, *Capacity Test*, *Component Test*, *Endurance Test*. Ressalta-se que para demonstrar os resultados do relatório gerencial proposto neste artigo optou-se pelo teste de esforço.

Assim como nos testes funcionais que possuem um ciclo bem definido dentro da sua etapa de teste, os testes de desempenho também têm uma metodologia definida [5] como pode ser visto na figura 3.

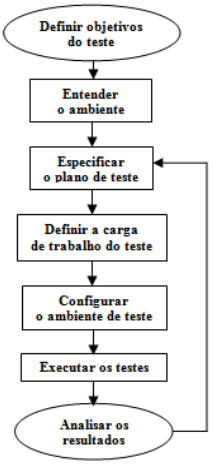

Figura 3. Uma metodologia para o teste de desempenho, de Menascé e Almeida.

Podem existir diversos tipos de objetivos a se alcançar com um teste de desempenho, alguns exemplos possíveis [5]:

a) Determinar a capacidade do servidor Web.

b) Descobrir o número máximo de usuários simultâneos que um serviço na Web aceita dentro dos limites acordados.

c) Determinar a quantidade de requisições simultâneas da camada de aplicação.

d) Identificar enfileiramentos na infra-estrutura do serviço Web.

e) Identificar a degradação do tempo de resposta percebido pelo usuário final ao crescimento das baterias de testes.

f) Descobrir a capacidade do servidor de banco de dados.

g) Identificar as funções Web que mais oneram a aplicação.

Após a definição dos objetivos desejados a atingir com o teste, é necessário ter bem definido qual o ambiente que o mesmo será executado. Nesta etapa do trabalho, busca-se descobrir que tipo de infra-estrutura está rodando a aplicação (informações de servidores, se existe algum serviço terceiro) e quais os softwares envolvidos nesta infra-estrutura (sistemas operacionais, middleware, aplicações, banco de dados).

Assim como no teste funcional, é necessário ter definido um plano de teste coerente que busque simular os passos dos usuários na aplicação. Este processo é o mais importante para se ter um resultado que busque atingir a expectativa dos objetivos, pois um cenário mal definido pode apresentar uma execução que não estimule de maneira correta o ambiente [5].

Diferentemente dos testes de caixa preta, estes testes buscam englobar apenas cenários cujo final resulte em sucesso e os fluxos mais utilizados pelos usuários, procurando uma distribuição da sua carga de trabalho em percentual de uso e freqüência [13], como pode ser visto na figura 4.

Figura 4. Exemplo de definição de cenário e carga de trabalho, de Barber.

Após a construção e definição dos cenários juntamente com a sua carga, é necessário efetuar a instrumentação com ferramentas de medição para monitorar cada camada do ambiente enquanto os testes são executados (Servidores Web, Servidores de Aplicação, Sistemas de banco de dados, Sistemas operacionais). Estes testes podem ser executados utilizando dois métodos [5]:

1) Teste Manual: Exigindo a quantidade de mão de obra desejada para obter a quantidade total de requisições definida no objetivo do teste e também uma alta sincronia entre os envolvidos para simular a carga necessária no tempo necessário.

2) Teste Automático: Que utiliza a simulação de usuários virtuais responsáveis em fazer o papel dos usuários reais e são comandados por uma controladora única.

Por facilidade, é mais comum a utilização do segundo método para execução destes testes de desempenho, pois o mesmo exige uma equipe reduzida tendo como um único pré-requisito que os envolvidos conheçam os cenários de testes e saibam utilizar a ferramenta que fará a simulação destes usuários virtuais.

Ao final da execução dos testes, a última etapa da metodologia é iniciada, com o início da coleta dos dados das ferramentas que monitoram os ambientes e com base nestas informações são feitas as sugestões de melhoria e otimização para todas as camadas do ambiente, onde nem sempre estas sugestões resultam em ampliação de capacidade de processamento de *hardware*, podendo ser sugestões de otimização de comandos de banco de dados e engenharia baseada em modificações de complexidades algorítmicas de códigos da aplicação.

O teste de desempenho é um método comum utilizado para determinar como os usuários experimentarão o desempenho de um serviço na Web, utilizando-se das abordagens específicas para cada tipo de cenário de carga desejado a simular.

IV. ELABORAÇÃO DE UM TESTE DE ESFORÇO USANDO O JMETER

Neste item é apresentado os passos realizados para a elaboração de um teste de esforço usando o JMeter. Como o JMeter é uma ferramenta cujo objetivo não é analisar o desempenho de aplicações online e sim executar testes de desempenho, neste trabalho propôs-se a criação de um aplicativo para apresentar os resultados de uma execução de testes de desempenho para a plataforma Android.

Para o desenvolvimento deste aplicativo, foi necessária a criação de uma arquitetura hipotética de uma loja do tipo e-commerce para ser elaborado um teste de esforço utilizando o JMeter. Para que assim fosse possível ter insumos suficientes para o funcionamento do relatório gerencial desenvolvido para o Android, o JPlotMeter. Portanto, nas próximas seções está descrita de forma sucinta a arquitetura lógica deste portal..

As aplicações baseadas na internet normalmente são enquadradas em uma arquitetura de três camadas. A primeira camada nesta arquitetura, também chamada de apresentação, incorpora a interface de usuário com os serviços na *online*. É nesta camada onde o usuário final preenche as informações que serão inseridas na aplicação. Na camada seguinte, onde estas informações preenchidas são processadas, é chamada de camada do negócio ou camada de aplicação. É nesta que estão encapsuladas uma coleção de regras para implementar a lógica da aplicação. Finalmente, os dados solicitados por este servidor de aplicação são processados por um servidor de banco de dados, onde são consistidas as informações de negócio. Esta distribuição das camadas pode ser vista na figura 5.

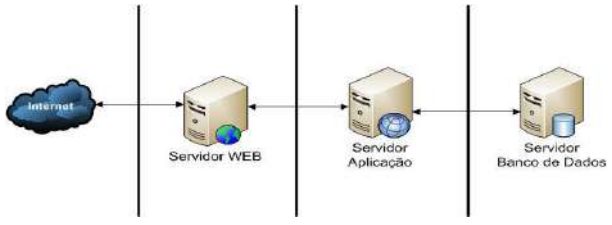

Figura 5. Arquitetura típica de um Site online de múltiplas camadas.

No estudo realizado utilizou-se uma aplicação baseada na arquitetura típica de um e-*commerce*, o JPetStore [14]. O JPetStore é uma loja virtual cujo objetivo é efetuar vendas de animais pela internet, armazenando as informações em um banco de dados. Esta aplicação foi proposta originalmente pela Sun como uma loja virtual que utiliza conceitos do Java *Enterprise Edition*.

Para que fosse possível potencializar a demonstração do relatório de tempo médio de resposta por página desenvolvido para o Android, foi escolhido o teste de esforço, que é o tipo de teste de desempenho que oferece maiores insumos para visualização da degradação de utilização dos recursos computacionais.

Com base neste portal, foram criados seguindo a fundamentação proposta por Myers [1] dois casos de teste que serviram como *script* para o desenvolvimento dos testes de esforço, utilizando duas funcionalidades da aplicação. A primeira refere-se ao fluxo de compra de um animal e a segunda, à pesquisa de um animal.

Para execução deste teste de desempenho no cenário hipotético proposto, tomou-se como premissa que a aplicação suporta em pico, 100 usuários simultâneos, isto significa que a aplicação está em 100% de utilização do site ao atingir esta quantidade de usuários simultâneos. De base nesta informação, foi criada uma pirâmide de usuários, que pode ser vista na figura 6, que demonstra a quantidade de usuários para os ciclos de testes que foram executados. Acima de 100 usuários, o conceito do teste de esforço já está agindo sobre a aplicação, conforme definição deste conceito de teste apresentado anteriormente, chegando à base da pirâmide com até 400% a mais da carga esperada pelo site.

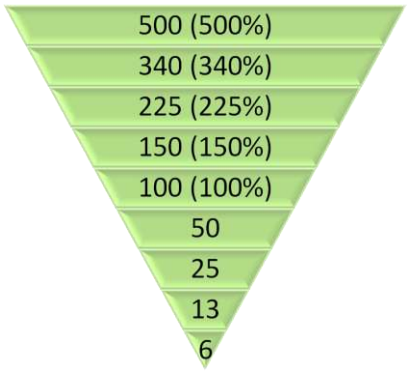

Figura 6. Pirâmide de usuários simultâneos utilizando o JPetStore.

Para cada nível da pirâmide executado, devem ser definidos os parâmetros de tempo de duração da bateria e quantidade de usuários que devem estar concorrendo simultaneamente por ciclo. Na figura 7 estes atributos estão definidos para o teste hipotético em questão, onde cada ciclo tem o tempo de 5 minutos de execução e todos os usuários concorrem simultâneamente desde o instante segundo 0.

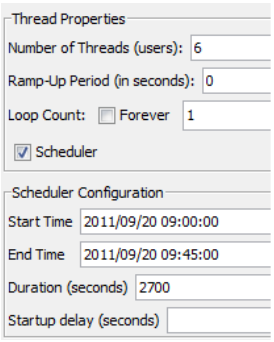

Figura 7. Atributos definidos para o caso de teste no JMeter.

Como serão dois cenários envolvidos nesta execução, cada cenário tem um peso específico para a quantidade de usuários utilizando a aplicação, sendo necessário definir a carga de usuário conforme definido anteriormente. Na figura 8 pode ser visto o percentual da execução no portal JPetStore para cada um dos níveis da pirâmide de usuários.

Figura 8. Distribuição de usuários simultâneos utilizando o JPetStore.

Tendo definido os casos de testes, a criação do *script* que simulará um usuário real, a quantidade de usuário que executará este script e o percentual de carga por funcionalidade, pode-se considerar finalizada a parte da criação de um teste de desempenho.

A previsão de duração do tempo da execução do ciclo de teste completo era de 45 minutos, já que o objetivo proposto era de executar os 9 níveis presentes na pirâmide de usuários, conforme na figura 6.

Porém no nível 7 da pirâmide, cujo total de usuário simultâneos era de 225 o que representa 125% a mais de carga esperada pela aplicação a execução foi encerrada, pois o servidor apresentou um alto consumo de CPU.

Como o objetivo deste trabalho não é mensurar o desempenho do hardware e sim demonstrar a utilização do aplicativo JPlotMeter desenvolvido para o Android, foi decidido que com a execução realizada já se possuia informações suficientes.

V. JPLOTMETER: UM RELATÓRIO GERENCIAL PARA O ANDROID

Os resultados das execuções dos testes feitos no JMeter armazenados em uma tabela temporária criada para o JPlotMeter são processados e expostos por um *webservice* para que o aplicativo no Android pudesse receber estas informações e gerar o gráfico. Um desenho técnico da aplicação pode ser visto na figura 9.

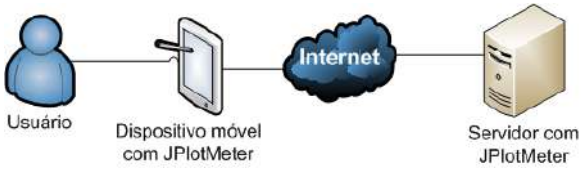

Figura 9. Desenho técnico da solução JPlotMeter.

No servidor com o JPlotmeter foram desenvolvidas quatro classes, dentre elas a AppService, HandleReturn, ConnectionMySql e a JaxWsImpl, como poder ser visto no diagrama de classes apresentado na figura 10.

A classe AppService é responsável em fornecer a informação já tratada para o aplicativo no Android, onde o método consulta através de uma conexão *Java Data Base Connection* (JDBC) uma tabela criada no MySQL, e trata as informações coletadas retornando insumo suficiente para atender a necessidade de gerar o gráfico referente ao tempo médio de resposta por ciclo de teste executado.

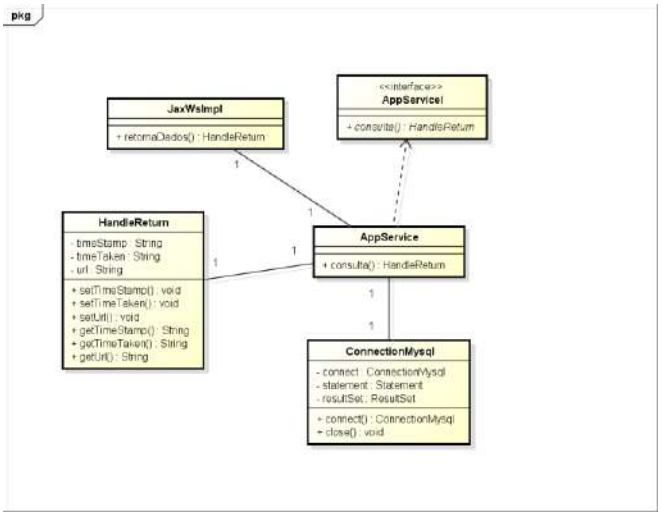

Figura 10. Diagrama de classe servidor de aplicações.

Para um melhor entendimento em relação à interação do usuário com o aplicativo, o diagrama de casos de uso foi elaborado, como pode ser visto na figura 11. O usuário precisa ter o aplicativo JPlotMeter instalado no seu dispositivo móvel com Android. Em seguida, abrir o aplicativo que o mesmo se encarregará de mostrar o gráfico gerencial de tempo médio de resposta na tela do dispositivo.

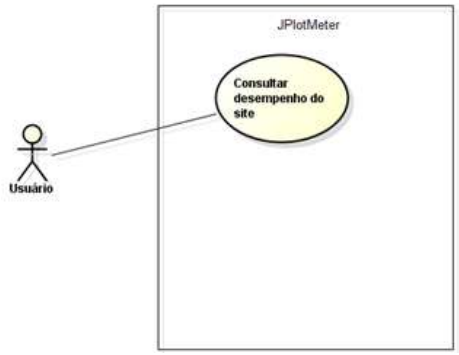

Figura 11. Diagrama de caso de uso.

Cada usuário simulado pelo JMeter gera uma série de informações sobre a sua execução. Para o protótipo de relatório gerencial JPlotMeter, três destas métricas rudimentares foram necessárias:

1. *Time Taken*: Representa o tempo de espera desde uma solicitação do usuário ao servidor até o retorno;
2. *Time Stamp*: O carimbo contendo data e horário da solicitação do usuário;
3. CSIUri: É o rótulo da *Uniform Resource Locator* (URL) solicitada pelo usuário.

Estas métricas foram armazenadas em uma tabela histórica e transitadas pela aplicação do lado do servidor para aplicação presente na plataforma Android, gerando assim a criação do gráfico representativo do tempo médio de resposta de cada uma das páginas.

Para que fosse possível ter informações suficientes para a demonstração do funcionamento do relatório, foram programadas para que a cada 5 minutos de uma bateria fosse executada a pirâmide de usuários hipotética, totalizando um período de 45 minutos de coletas armazenadas no banco de dados.

Para que fosse possível representar as informações coletadas a cada ciclo de execução no JMeter de maneira visual, foi desenvolvido o aplicativo JPlotMeter para o Android. Este aplicativo consiste em duas classes, que podem ser vistas no diagrama de classes na figura 12. A classe LerArquivo é a responsável por consumir as informações do aplicativo no lado do servidor e a classe JPlotMeter é a responsável em tratar as informações e gerar o gráfico em tempo real.

Para que o gráfico fosse montado de maneira padronizada, optou-se por utilizar uma *Application Programming Interface* (API) disponível para criação de gráficos, chamada jjoe64 [15]. Esta API possui métodos já implementados, que recebem valores númericos e textuais para construção do gráfico, seja ele do tipo linha ou do tipo barra. Para uma melhor representação do tempo de resposta das páginas, foi utilizado o gráfico de tipo linha.

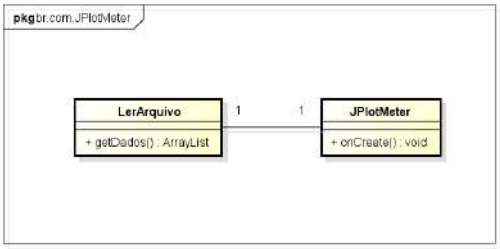

Figura 12. Diagrama de classes do aplicativo JPlotMeter.

Ao consumir as informações do lado do servidor, o aplicativo trata estas como um arquivo *Comma Separated Values* (CSV), que nada mais é do que um texto cujo os valores são separados por algum delimitador, no caso o delimitador é a vírgula. A partir destas informações recebidas, o *Time Taken* é o responsável em gerar o eixo vertical que é a média de tempo de resposta em segundos e o eixo horizontal é gerado através do *Time Stamp*, que é a linha temporal da execução, como pode ser visto no gráfico apresentado na figura 13.

Figura 13. Demonstração dos eixos do JPlotMeter.

Cada uma das estátiscas de acesso realizada pela execução no JMeter é armazenada no banco de dados da aplicação pelo lado do servidor com seu respectivo identificador, representado na tabela pela coluna CSIUri. Este campo no gráfico gerado é a legenda, como pode ser visto no gráfico apresentado na figura 14, sendo possível assim verificar a variação do tempo de resposta ao longo do tempo da execução, sendo que o somatório do tempo médio de resposta das páginas na última bateria executada se aproximou a 150 segundos de duração.

Figura 14. Demonstração das legendas no JPlotMeter.

O resultado identificado na execução do teste de esforço aplicado no portal e-commerce durante o período das 09:00 até às 09:24, no qual se identificou uma limitação física do servidor na bateria de 225 usuários, é refletido com um gradativo aumento do tempo médio de resposta por página representado pelo gráfico apresentado na figura 32, comportamento este que está diretamente ligado com o consumo excessivo de CPU do servidor, monitorado de maneira visual durante a execução do ciclo de teste.

Como o cenário hipotético inicial proposto para aplicação nas configurações físicas específicas o servidor suportava, em condições de pico, 100 usuários simultâneos. Foi possível constatar em uma execução de teste de esforço que a aplicação atende a demanda sem apresentar falhas ou quedas de até 225 usuários simultâneos. Neste caso, observou-se que o limitante para esta quantidade de usuários foi o consumo de CPU do servidor.

A quantidade de informações obtidas foi suficiente para que o gráfico de tempo de resposta médio das páginas fosse gerado, não sendo necessário uma outra execução ou até mesmo rever as informações coletadas.

VI. CONCLUSÕES E TRABALHOS FUTUROS

Com o levantamento teórico bibliográfico realizado neste trabalho foi possível convergir os diferentes conceitos sobre testes de desempenho e demonstrar um planejamento para se realizar a execução de um destes tipos de teste, o teste de esforço.

Para a demonstração do teste de esforço, foi desenvolvida uma arquitetura hipotética controlada, contendo as camadas de banco de dados, de aplicação e de web. Esta arquitetura possibilitou efetuar o teste escolhido e armazenar as métricas para futura utilização.

Uma vez implementada a arquitetura do JPetStore, foram escolhidas duas funcionalidades e elaborados dois casos de teste de esforço. A partir da execução destes casos de teste, foi possivel coletar algumas métricas para demonstração do aplicativo JPlotMeter para a plataforma Android. Sua funcionalidade principal é a demonstração de um relatório gerencial apresentando em forma de um gráfico, o tempo médio de resposta por páginas da aplicação.

O aplicativo JPlotMeter não depende exclusivamente dos dados gerados da execução do teste de esforço no portal JPetStore, sendo possível reaproveita-lo em outros portais. Para isto, é necessário que os dados coletados de outra execução de um teste de desempenho qualquer sejam inseridos na tabela do JPlotMeter de maneira que o *webservice* da aplicação possa expor as mesmas para geração do gráfico no Android.

A partir do tipo de informações coletadas pelas execuções feitas na aplicação *e-commerce* e apresentada pela aplicação JPlotMeter, projeta-se que seja possível expandir para mais um gráfico as informações visuais gerando um gráfico de *throughput*, que consiste em apresentar a média de requisições por usuário em um determinado período.

Além deste gráfico, existe a possibilidade de melhoria do JPlotMeter adicionando novas funcionalidades ou reformulação da sua funcionalidade principal desenvolvida. Pelo fato da aplicação que coleta a informação do lado do servidor ser estática, também projeta-se a possibilidade de migrar o cliente para outros sistemas operacionais disponíveis para arquitetura móveis, como o IOS da Apple e o Microsoft Windows *Phone* da Microsoft.

Finalmente, também projeta-se que para uma próxima versão do aplicativo seja possível integrar a saída das informações geradas para cada usuário executado no JMeter diretamente com a base de dados do JPlotMeter, não sendo necessário assim tratar esta informação antes de importá-la para o banco de dados.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Myers, Glenford J.. The Art of Software Testing. 2ª ed. New Jersey: John Wiley & Sons Inc, pp. 20-149, 2004.

[2] Guerra, Ana Cervigni; Colombo, Regina Maria Thienne. Tecnologia da Informação: qualidade de produto de software. Brasilia: PBQP, pp. 429, 2009.

[3] Bartié, Alexandre. Garantia de Qualidade de Software. 3ª ed. São Paulo: Campus, pp. 131-327, 2002.

[4] Pressman, Roger S.. Engenharia de Software. 6ª ed. São Paulo: Pearson Makron Books, pp. 323-522, 2005.

[5] Menascé, Daniel A., Almeida, Virgilio A. F.. Planejamento de Capacidade para serviços na Web: Métricas, modelos e métodos. São Paulo: Campus Elsevier, pp. 54-298, 2003.
[6] Miller, Robert B.. Response time in man-computer conversational transactions. New York: International Business Machine Corporation, in press.
[7] Zona, Research inc.. User trends: The 8-second rule of I-commerce. A Zona Research survey on web-page download speeds and electronic commerce. Info World, 1999.
[8] Bhatti, Nina, Bouch, Anna, Kuchinsky, Allan (04/2011). Integrating User-Perceived Quality into web Server design. 9th International World Wide Web Conference [Online]. Disponível em: http://portal.acm.org/citation.cfm?id=346245
[9] Spool, Jared M.. An interview with Jared Spool of User Interface Engineering, conducted by John Rhodes, WebWorld, 2001.
[10] Barber, Scott (04/2011). How fast does a website need to be? [Online]. Disponível em: http://www.perftestplus.com/resources/how_fast.pdf
[11] Almeida, Virgilio, Menascé, Daniel A., Dowdy, Larry W.. Performance by Design: Computer Capacity Planning by Example. New Jersey: Prentice Hall, pp. 15-197, 2004.
[12] Meier, J. D. et al. Performance Testing Guidance for Web Applications. Redmond: Microsoft Press, pp. 13-143, 2007.
[13] Barber, Scott (11/2012). User experience, not metrics part 3: Modeling individual user patterns [Online]. Disponível em: http://www.perftestplus.com/resources/UENM3.pdf
[14] JPetStore – Site Oficial (11/2012) [Online]. Disponível em: http://www.oracle.com/technetwork/java/petstore1-3-1-02-139690.html
[15] Jjoe64 – Site Oficial (09/2011) [Online]. Disponível em: http://www.jjoe64.com/p/graphview-library.html