Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Um framework para testes de *software* na nuvem

Gustavo Sávio de Oliveira

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

João Pessoa, Paraíba, Brasil.

Universidade Federal da Paraíba
Centro de Informática
Programa de Pós-Graduação em Informática

Um framework para testes de *software* na nuvem

Gustavo Sávio de Oliveira

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Computação Distribuída

Prof. Dr. Alexandre Nóbrega Duarte
(Orientador)

João Pessoa, Paraíba, Brasil.

O48u Oliveira, Gustavo Sávio de.
Um framework para testes de software na nuvem / Gustavo Sávio de Oliveira.-- João Pessoa, 2012.
84f. : il.
Orientador: Alexandre Nóbrega Duarte
Dissertação (Mestrado) - UFPB/CI
1. Informática. 2. Ciência da computação. 3. Teste de software. 3. Computação em nuvem.

UFPB/BC CDU: 004(043)

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de **GUSTAVO SAVIO DE OLIVEIRA**, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 29 de agosto de 2012.

Ao vigésimo nono dia do mês de agosto do ano dois mil e doze, às quatorze horas, na sala do REUNI - da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Computação Distribuída*", o Sr. **GUSTAVO SAVIO DE OLIVEIRA**. A comissão examinadora foi composta pelos professores doutores: ALEXANDRE NOBREGA DUARTE (PPGI-UFPB), Orientador e Presidente, ALISSON VASCONCELOS DE BRITO (PPGI-UFPB), como examinador interno, e DIEGO MOREIRA DE ARAUJO CARVALHO (CEFET-RJ) como examinador externo. Dando início aos trabalhos, o professor ALEXANDRE NOBREGA DUARTE, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*Um framework para testes de software na nuvem*". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, eu, professora Liliane dos Santos Machado, vice-coordenadora deste programa, servindo de secretária, lavrei a presente ata que vai assinada por mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 29 de agosto de 2012.

Liliane dos Santos Machado

Prof. Dr. ALEXANDRE NOBREGA DUARTE
Orientador e Presidente (PPGI-UFPB)

Prof. Dr. ALISSON VASCONCELOS DE BRITO
Examinador Interno (PPGI-UFPB)

Prof. Dr. DIEGO MOREIRA DE ARAUJO CARVALHO
Examinador Externo (CEFET-RJ)

Resumo

Infraestruturas de computação em nuvens podem ser utilizadas para tornar o processo de testes mais eficiente e eficaz, obtendo desde redução dos custos de aquisição de infraestrutura à flexibilidade em se utilizar apenas os recursos realmente necessários para efetuar os testes. Entretanto, explorar tais infraestruturas para auxiliar no processo de testes requer procedimentos de configuração e automatização nem sempre disponíveis. Neste contexto, identificamos a ausência de ferramental que efetivamente automatizasse todo o processo e tornasse uma infraestrutura de computação em nuvem parte integrante do ambiente de desenvolvimento e testes. Este trabalho objetiva apresentar uma solução que distribui e coordena a execução paralela de testes automáticos de *software* em ambientes distribuídos e heterogêneos. Tal solução, denominada *CloudTesting* fornece ao desenvolvedor o suporte necessário para executar testes de unidade de *software* na nuvem. Experimentos realizados com o *CloudTesting* em uma infraestrutura de computação na nuvem demonstraram reduções de mais de 20 horas no tempo gasto de execução de um conjunto sintético de testes de unidade.

Palavras-Chave: Teste de *software;* Computação em Nuvem.

Abstract

Cloud Computing infrastructures can be used to make the testing process more efficient and effective, resulting from lower costs of acquisition of infrastructure to the flexibility of using only the resources actually required to perform the tests. However, exploiting such infrastructures to assist in the testing process requires automation and setup procedures not always available. In this context, we identified the lack of tools that effectively automate the entire process and make a cloud computing infrastructure part of the development and testing environment. This work presents a solution that distributes and coordinates the parallel execution of automated software testing in distributed and heterogeneous environments. This solution, called CloudTesting, provides the necessary support to developers to perform unit tests in the cloud. Experiments performed with CloudTesting in a cloud computing infrastructure showed reductions of more than 20 hours of execution time spent of a condensed set of unit tests.

Keywords: Software testing, Cloud Computing**.**

Ao meu saudoso pai, Dácio Lima Silva de Oliveira (in memoriam).

Agradecimentos

Como em todos os momentos da minha vida, agradeço primeiramente a Deus, por me fornecer todos os subsídios necessários para dar andamento a essa atividade.

Ao meu pai (*in memoriam*), pela sua amizade, companheirismo, esforço e amor. Por ter me ensinado a pensar, a agir, a analisar a vida e valorizar cada instante.

A minha mãe, pela sua coragem, força, amor e incentivo fora do comum. Por ser um exemplo de pessoa e dignidade.

A minha avó, por todo carinho e zelo constante.

A minha pequena sobrinha Thamira, por radiar alegria em momentos difíceis, por despertar o sentimento de pai mesmo antes de ter me tornado um.

A minha irmã, por participar da minha vida.

A minha tia Alaide, por ter me acolhido durante o momento mais difícil da minha vida. Por todo amor e carinho recebido, por toda a sabedoria e demonstração de fé repassada.

Ao amor da minha vida, Rita de Cássia, por todo o incentivo, dedicação, carinho e amor recebido.

Ao professor Alexandre Duarte, pela pessoa que é. Por ter me escutado em momentos críticos, pela sua amizade e competência.

Ao professor Alisson Brito, por desde a época da graduação ter fomentado a sede pela pesquisa e conhecimento científico, pela sua amizade e seus direcionamentos.

Para todos os amigos que contribuíram de modo direto ou indireto.

Ao CNPq e a todos que constituem o PPGI.

Conteúdo

Lista de Símbolos

CAPES Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CPU *Central Processing Unit*

EBS *Elastic Block Store*

EC2 *Elastic Compute Cloud*

IAAS *Infrastructure as a service*

IDE *Integrated Development Environment*

IEEE *Institute of Electrical and Electronic Engineers*

P2P *Peer-to-peer*

PAAS *Platform as a service*

PDE *Plug-in Development Environment*

QEMU *Quick EMUlator*

QOS *Quality of service*

RAM *Random Access Memory*

SAAS *Software as a service*

SBC Sociedade Brasileira de Computação

SLAS *Service Level Agreement*

TI Tecnologia da Informação

TPTP *Test and Performance Tools Platform*

Lista de Figuras

Lista de Equações

Lista de Tabelas

Capítulo 1

Introdução

Historicamente, quando as organizações demandavam maior capacidade de armazenamento de dados ou de poder de processamento para seus sistemas havia duas opções a seguir: adquirir mais *hardware* ou tornar a operação de tecnologia da informação (TI) mais eficiente [Grundy *et al*. 2012]. Ambas as opções implicavam em aumentos significativos dos custos de aquisição, implantação, operação e manutenção do *hardware* e de sua infraestrutura de rede.

A Computação em Nuvem é um modelo que permite de modo conveniente, o acesso a recursos sob demanda por meio de um *pool* compartilhado de recursos computacionais configuráveis que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação com o provedor de serviços [Jadeja e Modi 2012].

Hoje, a Computação em Nuvem surge como uma nova alternativa para as organizações, sendo uma das tecnologias emergentes mais discutidas na indústria de TI nos últimos anos por apresentar um modelo de negócio que fornece recursos computacionais sob demanda com custos reduzidos de administração [Riungu-Kalliosaari *et al.* 2012] [Grundy *et al*. 2012].

Computação em Nuvem é uma tendência que vem sendo bastante explorada, como confirma a pesquisa da *Gartner Research*, que antecipou que até 2012, 20% das empresas buscariam adquirir recursos em nuvens ao invés de adquirirem seus próprios equipamentos. Grandes empresas de TI têm investido no fornecimento desses serviços com destaque para Amazon, Microsoft, Apple e Google. De um modo geral, a Computação em Nuvem vem modificando diversas subáreas, incluindo a forma de entrega e uso de *software*. Atualmente, o licenciamento tradicional de *software* está diminuindo e as contratações de serviços em nuvens estão em ascensão [Riungu-Kalliosaari *et al.* 2012].

Em se tratando de processos de teste de *software*, a Computação em Nuvem representa uma nova alternativa para tornar o processo mais eficiente e eficaz quando comparada a métodos e abordagens tradicionais [Riungu-Kalliosaari *et al.* 2012]. Observando que em um processo de desenvolvimento moderno, testes utilizam em média cerca de 40% da carga de trabalho total no período da elaboração do *software* [Fewster e Graham 1999], avalia-se que testar *software* pode ser tão custoso quanto desenvolvê-lo [Catelani *et al*. 2008]. A adoção de Computação em Nuvens nesse processo possibilita a redução de custos relacionados à aquisição e manutenção de *hardware*, licenciamento de ambientes de testes internos, além de aumentar a velocidade de

execução dos testes por sempre obterem os recursos realmente necessários para a execução de um dado teste.

Apesar destes benefícios, desenvolver e executar testes em nuvem não é uma atividade trivial, requerendo habilidades técnicas para configurar e submeter os casos de teste, muitas vezes implicando em um aumento na complexidade dessa tarefa.

Diante deste contexto, automatizar o processo de utilização de infraestruturas de computação em nuvem como ambiente para execução de testes automáticos de *software* é fundamental para que tais infraestruturas passem realmente a fazer parte do ambiente de desenvolvimento e testes.

Observa-se então uma lacuna por ferramentas que auxiliem no desenvolvimento, distribuição e controle da execução de testes automáticos de *software* em infraestruturas de computação em nuvem. Tais soluções devem abstrair boa parte da complexidade envolvida nesse processo com o objetivo de facilitar, agilizar e suavizar o uso dessa tecnologia. Porém, devem ser flexíveis o suficiente para se adaptarem a diferentes cenários e diferentes processos de teste.

Sendo assim, é indispensável à definição de um *framework* que sirva como referência para que testes possam ser executados na nuvem com um mínimo de esforço de desenvolvimento [Silva 2006] [Price 1997].

1.1 Motivação

Testes automáticos são um importante ingrediente no processo de qualidade de *software* e fundamentais no desenvolvimento de sistemas computacionais confiáveis. Um conjunto de testes automáticos que captura o comportamento esperado do *software* pode, inclusive, ser utilizado para validar a implementação dos seus requisitos funcionais [Duarte *et al*. 2006]. Testes garantem a funcionalidade do *software* e reduzem custos, já que efetuar a correção de um erro de implementação de *software* na etapa de manutenção é de sessenta a cem vezes maior que o custo para corrigi-lo durante o período de desenvolvimento [Pressman 2006].

A literatura relata grandes desastres causados por *software* outrora considerado devidamente testado [Ben-Ari 1999] [Mellor 1994] por conta de defeitos que permaneceram ocultos até o momento em que o *software* foi executado em seu ambiente de produção. Tais acontecimentos levaram a um maior desenvolvimento de soluções visando a diminuição do tempo de duração de cada etapa do processo de testes [Rothermel *et al*. 2001], a automatização de todo o procedimento de testes [Duarte *et al*. 2006] e implicaram na necessidade de métodos mais simples para configuração e controle efetivos do processo de testagem.

Com a adoção de Computação em Nuvem como infraestrutura de suporte para testes automáticos de *software*, espera-se alcançar desempenho e escalabilidade de maneira eficiente e econômica. Com a nuvem é fácil adquirir uma infraestrutura completa e diversificada para testes a uma velocidade e custo que seriam impossíveis com os métodos tradicionais [Riungu-Kalliosaari *et al.* 2012]. Devido à complexidade dos sistemas computacionais atuais cada ambiente computacional é único [Duarte *et al.* 2006]. Isso implica que, por mais que o ambiente de desenvolvimento de *software* e seu ambiente de produção sejam parecidos, ambos divergem significativamente. Esta divergência aliada à possível contaminação do ambiente de desenvolvimento pode dificultar na detecção de defeitos que possam levar a ocorrências de falhas durante a execução do *software*. Desta forma, testar o *software* em ambientes diversos, distintos e isolados do seu ambiente de desenvolvimento é fundamental para aumentar a confiança na cobertura e corretude do conjunto de testes utilizado.

Teste de *software* baseado na nuvem possibilita intercalar facilmente entre ambientes operacionais diferentes por meio da virtualização, o que resulta em testes mais realistas, permitindo que os desenvolvedores entendam a perspectiva de seus serviços do ponto de vista dos usuários finais. Entretanto, a execução de testes em uma infraestrutura de Computação em Nuvem implica em uma série de atividades inerentes a execução de aplicações distribuídas, por exemplo, balanceamento de carga, latência da rede e *multitenancy*, fazendo com que testadores precisem se preocupar com aspectos da execução que não estão diretamente relacionados às funcionalidades ou requisitos não-funcionais da aplicação a ser testada [Riungu-Kalliosaari *et al.* 2012].

Desta forma, é necessário que uma aplicação para execução de testes de automáticos de *software* na nuvem abstraia estes aspectos do testador, permitindo que ele mantenha o foco na sua atividade principal, que é testar o *software*.

Tal solução deve esconder do testador a complexidade de configuração e execução de testes na nuvem, permitindo que mesmo usuários que não possuem qualquer conhecimento sobre tal tecnologia possam explorar suas características durante o processo de teste. Idealmente, tal solução não deve exigir modificações profundas no processo de testes ou na forma utilizada para descrever os testes automáticos, limitando significativamente os custos necessários para sua adoção.

1.1.2 Pesquisa realizada com desenvolvedores de *software*

Com o intuito de mapear as tendências de mercado, desenvolveu-se um *survey* com a intenção de apanhar evidências anedóticas de quais tecnologias são mais utilizadas na indústria de *software*.

Um questionário contendo dez questões de múltipla escolha foi divulgado para uma comunidade de desenvolvedores de software com o intuito de identificar o estado da prática em relação à execução de testes automáticos de forma distribuída ou paralela. O questionário foi disponibilizado na Web através de um formulário acessível por meio do endereço http://goo.gl/Amsys e anunciado nas listas de discussão apresentada pela Tabela 1.

Tabela 1 - Lista de discussão onde o *survey* foi divulgado.

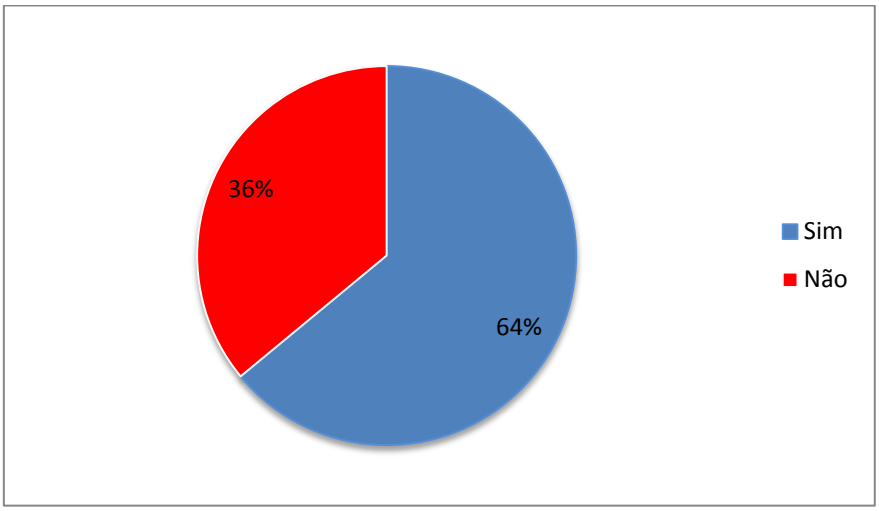
Listas	**Endereços**
SBC-I	https://grupos.ufrgs.br/mailman/listinfo/sbc-l
PBJUG	http://groups.google.com/group/pbjug?lnk=srg&hl=pt
PHPPB	http://groups.google.com/group/php-pb?lnk=srg&hl=pt
DFJUG	http://www.grupos.com.br/grupos/dfjug
LISTA-PHP	https://groups.google.com/group/listaphp?hl=pt
Ceará On Rails	https://groups.google.com/group/cearaonrails?hl=pt
JQuery Brasil	https://groups.google.com/group/jquery-br?hl=pt
Rails-br	https://groups.google.com/group/rails-br?hl=pt
Rails-go	https://groups.google.com/group/rails-go?hl=pt
Android Brasil-Dev	https://groups.google.com/group/androidbrasil-dev?hl=pt
JavaServer Faces Group Brasil	https://groups.google.com/group/javasf?hl=pt
JavaCE	https://groups.google.com/group/javace?hl=pt
JavaRN	https://groups.google.com/group/javarn?hl=pt

FlexPB	https://groups.google.com/group/flexpb?hl=pt
ArchLinux-BR	https://groups.google.com/group/archlinux-br?hl=pt
TCheLinux-RS	https://groups.google.com/group/tchelinux?hl=pt

Foram coletadas duzentas e oitenta e seis respostas em uma única etapa, entre os meses de junho e julho de 2011. Por possuir a característica de múltipla escolha, a soma das percentagens dos resultados das questões pode ultrapassar 100%. Abaixo são discutidas as perguntas que foram relevantes para o trabalho.

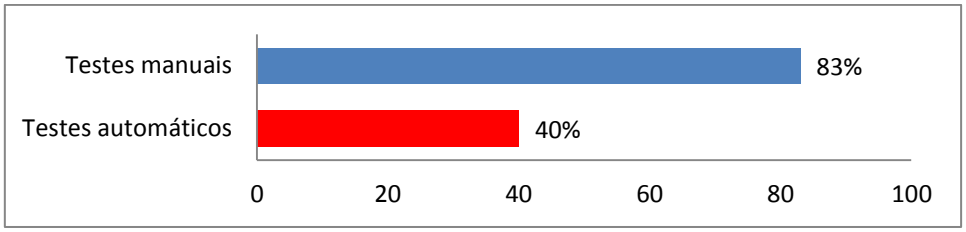
A pergunta "*O processo de desenvolvimento de software utilizado no seu local de trabalho/estudo/pesquisa contempla a realização de testes?"* busca verificar o nível de utilização de testes. Conforme apresenta a Figura 1, 64% das respostas afirmam empregar testes durante a fase de desenvolvimento de *software*.

Figura 1- Utilização de testes no desenvolvimento de *software*.



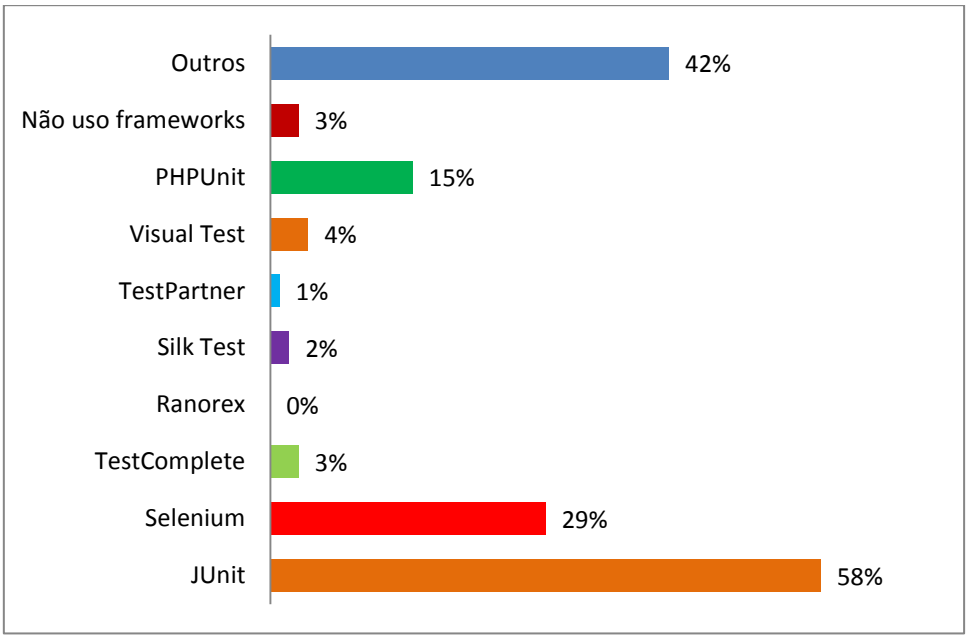
Entretanto, a Figura 2 ilustra os resultados para a pergunta "*Como os testes são realizados*?" Apenas 40% desses testes são efetuados de modo automático versus 83% de testes executados de modo manual. Esse fato é insatisfatório, visto que, para que um teste de *software* seja eficaz, o mesmo deve possuir as características de rapidez e automação [Jeffries *et al*. 2000] [Kapfhammer 2001].

Figura 2 - Modo de execução dos testes.



As pergunta "*Que framework(s) ou ferramenta(s) você utiliza ou já utilizou para desenvolver e executar testes automáticos de software?" map*eou qual tecnologia é utiliza para efetuar os testes, como resultado o *framework JUnit* [Gamma e Back 1999] destacou-se como a mais utilizada para o teste de *software* de unidade com 58%, conforme ilustra a Figura 3. Por meio dessa constatação se torna possível inferir qual tecnologia atacar, podendo-se desenvolver uma solução que abranja a maior parte dos usuários.

Figura 3 – Ferramentas para execução de testes automáticos.



Os resultados obtidos para a pergunta "*Seria proveitoso para seu trabalho ser capaz de executar testes automáticos de forma mais rápida, explorando recursos de computação paralela ou distribuída?"* Motivam o desenvolvimento de uma ferramenta para testes de *software* baseado nas nuvens, 91% dos usuários apontam que obteriam benefícios em suas atividades ao adotarem soluções de computação paralela ou distribuída, apenas 9% das respostas coletadas contrariam essa afirmativa.

Esse resultado se reflete também nas respostas referentes ao tempo máximo gasto para executar um conjunto de testes automáticos, onde 64% dos participantes afirmaram ter gasto mais que uma hora em uma única execução e, dentre estes, 7% gastaram mais que 10 horas. O que se

observa é que o uso de tecnologias de malhas ou nuvens possibilita fazer o que antes não poderia ser feito. Com os resultados obtidos pelo *CloudTesting* **(Capítulo 5)**, conclui-se que os 7% dos participantes que gastaram mais que 10 horas irão poder aumentar a sua produtividade de maneira nunca antes permitida.

1.2 Objetivos

Essa seção discute os objetivos do trabalho. Inicialmente é descrito o objetivo geral (seção 1.2.1) com o desígnio de apresentar a extensão do projeto. Em seguida são abordados os objetivos específicos (seção 1.2.2) que visam tratar das contribuições propostas.

1.2.1 Objetivo Geral

Com o objetivo de maximizar as vantagens do uso de infraestruturas de Computação em Nuvem para execução de testes automáticos de *software*, esse trabalho procura desenvolver um *framework* para automatizar o processo de distribuição, controle de execução, coleta e apresentação dos resultados da execução de testes de unidade de *software*.

1.2.2 Objetivos Específicos

Pretende-se especificamente através dessa pesquisa alcançar os seguintes objetivos específicos:

1. Arquitetar uma abordagem para execução distribuída de conjuntos de testes de *software* oferecendo baixa intrusividade, simplicidade e fácil manutenção e alta flexibilidade;
2. Desenvolver um protótipo de *framework* baseada na arquitetura proposta para execução distribuída de conjuntos de testes de *software* na nuvem;
3. Avaliar experimentalmente a abordagem proposta utilizando o protótipo desenvolvido em uma infraestrutura de computação distribuída;
4. Documentar o trabalho desenvolvido e publicar os resultados obtidos em eventos.

1.3 Metodologia de Pesquisa

Essa seção descreve as principais fases da metodologia adotada no trabalho. A estratégia utilizada neste projeto define a divisão do esforço em grupos de atividades que estão relacionadas ao mapeamento das soluções existentes para execução distribuída ou paralela de conjuntos de testes de *software*, a validação da abordagem proposta através da concepção de um protótipo e da realização de experimentos práticos utilizando o serviço de computação em nuvem da Amazon EC2. As atividades previstas são descritas a seguir:

1. **Pesquisa bibliográfica**: Responsável por identificar o estado da arte por meio de bases de publicações científicas em busca de trabalhos relacionados. As seguintes bases foram levantadas: *IEEE Xplorer*[1], Periódicos da CAPES[2], Biblioteca Digital da SBC[3], Biblioteca Digital da UNICAMP[4], Domínio Público[5], *Association for Computing Machinery*[6], *The DBLP Computer Science Bibliography*[7], *Scientific Commons*[8], *Citeseerx*[9], *Microsoft Academic Search*[10] e *Google Scholar*[11]. Nesses ambientes, as palavras-chave consistiram em termos relacionados a Teste de *software* baseado na nuvem;
2. **Levantamento de requisitos**: Verificar as necessidades do desenvolvimento do *framework* para execução de testes automáticos na nuvem, tal como a observação das limitações dos trabalhos correlatos em busca de descobrir possibilidades que não foram contempladas por outras ferramentas;
3. **Planejamento**: Determina o escopo do *framework* proposto, com base nos artefatos desenvolvidos durante a fase de levantamento de requisitos;
4. **Desenvolvimento**: Implementação do *framework* proposto;
5. **Avaliação**: Realização de experimentos e análise dos resultados obtidos;
6. **Aprimoramento**: Análise dos resultados de cada estágio, além de reuniões periódicas entre o aluno e o orientador para discutir o andamento do projeto, tal como verificar se o trabalho

[1] IEEE Xplorer: http://ieeexplore.ieee.org
[2] Periódicos da CAPES: http://www.periodicos.capes.gov.br/
[3] Biblioteca Digital da SBC: http://portalsbc.sbc.org.br/
[4] Biblioteca Digital da UNICAMP: http://cutter.unicamp.br/
[5] Domínio Público: http://www.dominiopublico.gov.br
[6] Association for Computing Machinery: http://www.acm.org/
[7] The DBLP Computer Science Bibliography: http://www.informatik.uni-trier.de/~ley/db/
[8] Scientific Comm: http://en.scientificcommons.org/
[9] Citeseerx: http://citeseerx.ist.psu.edu/
[10] Microsoft Academic Search http://academic.research.microsoft.com/
[11] Google Scholar : http://scholar.google.com/

mantém o contexto escolhido inicialmente ou se necessita adequá-lo, além de ponderar melhorias em nível de *software*.

1.4 Publicações relacionadas

Os resultados da pesquisa desenvolvida durante a construção deste trabalho possibilitou, até o momento, a aceitação do seguinte artigo para publicação:

- Oliveira S. G., Duarte N. A. *Framework para Teste de Software Automático nas Nuvens*. In: VI Workshop Brasileiro de Teste de *Software* Sistemático e Automatizado, 2012, Natal.

1.5 Estrutura do trabalho

O restante deste trabalho toma como base todos os conceitos necessários para a descrição e concepção da pesquisa e seus experimentos. Assim, além deste, segue a sequência de capítulos listados abaixo:

- **Capítulo 2 - Fundamentação Teórica**: Neste capítulo são expostos os conceitos e terminologias intrínsecas à proposta do trabalho. São levantadas duas grandes áreas: Teste de *software* e Computação em nuvem;
- **Capítulo 3 - Trabalhos Relacionados**: Neste capítulo são apresentados trabalhos relevantes e relacionados à proposta do projeto. Esses trabalhos são definidos por categoria, analisados e comparados entre eles;
- **Capítulo 4 - *CloudTesting*:** Essa seção apresenta a concepção do *framework* para execução de testes automáticos na nuvem, denominado *CloudTesting*, descrevendo sua definição e especificação;
- **Capítulo 5 - Experimentos e resultados:** Essa seção apresenta os testes elaborados para o processo de análise do *CloudTesting*, descrevendo o modo como foram efetuados e as contribuições alcançadas por meio dos resultados obtidos;
- **Capítulo 6 Conclusões:** Por último, o trabalho se encerra por meio de discussões que apresentam as contribuições da pesquisa e analisam perspectivas para o desenvolvimento de trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo tem por finalidade apresentar a fundamentação teórica necessária para a construção de um *framework* para distribuição da execução de testes automáticos de *software*. Apresentam-se inicialmente considerações sobre teste de *software* de forma geral (seção 2.1) seguida de uma discussão sobre testes de unidade (seção 2.2). Após isso, trata-se de testes automáticos de *software*, computação distribuída, computação em nuvem e *frameworks* (seções 2.3, 2.4, 2.5 e 2.6 respectivamente). O capítulo é encerrado com uma sessão de considerações finais (seção 2.7).

2.1 Teste de *Software*

O desenvolvimento de *software* é uma tarefa complexa, que exige diferentes habilidades técnicas e humanas, e que envolve um grande esforço e consome muitos recursos [Xiaohui *et al*. 2010]. A criação de *software* é um trabalho passível de erro, devido a variáveis, tais como a heterogeneidade e dinamismo dos sistemas computacionais modernos [Duarte *et al*. 2006]. *Software* é utilizado desde a execução de afazeres simples até serviços complexos em diferentes aspectos. É exatamente por essa razão que há a necessidade de garantir a qualidade do produto [Fuggeta 2000]. Para isso, as etapas de desenvolvimento de *software* são apoiadas por atividades de teste [Shahamiri *et al*. 2009].

A finalidade básica do teste é a de identificar erros, sejam eles de implementação ou de requisitos de projeto. Para Myers (2004) teste de *software* é uma metodologia que envolve executar um *software* com a finalidade de localizar erros, classificando-os a partir das seguintes premissas: (1) bom teste, caso exista uma probabilidade elevada de identificação de erros, (2) teste bem sucedido, caso encontre um erro não identificado em outras análises. Shahamiri *et al*. (2009) discute que o teste é adotado para melhorar a qualidade e confiabilidade, utilizado como um processo de detecção de erros e falhas em produtos de *software*, sendo praticamente impossível efetuar um teste que avalie um *software* por completo.

Pressman (2006), Gimenes *et al*. (2005) e Pádua (2003) compartilham da mesma opinião, afirmando que o procedimento de teste visa efetuar uma análise no *software* com o objetivo de descobrir falhas que resistam as revisões. Com isso, buscando alcançar quaisquer atributos de

qualidade, tais como: funcionalidade, confiabilidade e eficiência. Para Whittaker (2000) e Beizer (1995) o teste de *software* atesta a validação, consistindo em um método de execução de um produto para definir se o escopo cobriu as especificações e o *software* realizou suas funções com êxito. Catelani *et al*. (2008) aborda que o teste de *software* é uma atividade crítica no processo de desenvolvimento.

Apesar do teste de *software* possuir características de identificação de erros visando corrigi-los, se aplicado de forma indevida os resultados possivelmente não representarão a realidade. Myers (2004) e Crespo *et al*. (2004) destacam o planejamento e a estratégia do teste como fatores para que se alcance resultados proeminentes. Para McGregor e Sykes (2001) não somente o planejamento deve ser observado, como também o controle de recursos disponibilizados para o teste, visto que, de acordo com Juristo *et al*. (2004), teste de *software* é uma atividade que demanda um bom investimento financeiro. Contudo, conforme citado anteriormente, Pressman (2006) destaca que o custo para efetuar a correção de um erro de implementação de *software* na etapa de manutenção é de sessenta a cem vezes maior que o custo para corrigi-lo durante o período de desenvolvimento.

Segundo Myers (2004) existem duas estratégias principais para testar um *software*: testes de caixa branca e testes de caixa preta. Testes de caixa branca, também conhecido como testes unitários ou testes estruturais, se baseiam em uma análise rigorosa do código fonte da aplicação, sendo o *software* testado internamente, envolvendo aspectos relacionados aos caminhos lógicos e aos diversos relacionamentos entre seus componentes. Testes de caixa branca são bastante utilizados como testes de unidade e testes de integração. Os resultados obtidos por meio desse tipo de teste aprimoram as demais atividades da Engenharia de *Software*, tais como: manutenção, confiabilidade e melhoria do processo [Harrold 2000].

Testes de caixa preta, também conhecido como testes funcionais, são utilizados para avaliar o comportamento externo do sistema, ignorando por completo qualquer detalhe específico da implementação e comportamento interno de um determinado *software*. De acordo com Myers (2004) o desenvolvedor/testador utiliza a especificação do *software* para obter os requisitos ou dados do teste, desconsiderando a implementação.

A característica de ignorar completamente os componentes internos do *software* e de se basear pelas especificações para efetuar a análise de teste, torna a técnica caixa preta portável para testar *software*s desenvolvidos em diversos paradigmas, tais como: procedural, orientado a objetos, orientado a aspectos [Binder 2000] [Offutt e Irvine 1995]. O funcionamento do teste consiste no fornecimento de dados de entrada para os quais a saída já é conhecida, assim existe uma comparação do resultado esperado com a saída produzida pelo programa sendo testado.

2.2 Testes de Unidade

Testes de unidade são utilizados para verificar funcionalidades especificas de um trecho de código pré-determinado, geralmente no nível de métodos, classes ou módulos no caso de linguagens orientadas a objetos [Binder 2000], procurando identificar erros na lógica e na implementação do código-fonte do sistema.

Essa classe de teste geralmente é escrita pelos próprios desenvolvedores do *software*, enquanto escrevem o código da aplicação (testes de caixa branca), ou antes, do código da aplicação ser escrito (testes de caixa preta). Nessa etapa, a descrição do projeto é utilizada como orientação para a execução dos testes. Os testes podem ser executados através de *scripts* automatizados ou executados de modo manual.

É importante ponderar que testes de unidade isoladamente não são capazes de checar funcionalidades de *software* como um todo nem do relacionamento de tal *software* com componentes externos ou com outros componentes de *software*. Eles são utilizados para checar se cada bloco fundamental do programa funciona corretamente quando isolado dos demais blocos [Duarte 2010].

A execução automatizada é a abordagem preferida, sobretudo porque a quantidade de testes de unidade para um *software* de tamanho razoável tende a ser muito grande e tais testes devem ser realizados sempre que qualquer nova funcionalidade é adicionada ou quando um defeito é removido, para checar se as alterações realizadas não produziram efeitos colaterais em alguma outra parte do código [Gupta e Jalote 2007].

Estes conjuntos de testes automáticos são ainda mais benéficos no desenvolvimento de grandes projetos de *software*, o que geralmente envolve vários times de desenvolvedores. Nesse contexto, um conjunto de testes estabelece uma camada de abstração para os times, não sendo necessário que os desenvolvedores conheçam detalhes da implementação de um módulo desenvolvido por outra equipe para saber se ele funciona corretamente [Jeffries *et al*. 2000].

Existem *frameworks* para a concepção de testes automáticos de *software* para praticamente todas as linguagens de programação disponíveis no mercado. Um dos exemplos mais conhecidos e mais utilizados no mercado é o *JUnit* [Gamma e Beck 1999], empregado para descrever testes de unidade para programas escritos na linguagem Java. Abaixo na Figura 4, segue um exemplo da implementação de uma classe escrita em Java que servirá para demonstrar o uso do *framework* JUnit.

Figura 4 – Classe utilizada para exemplificar o uso do *framework* JUnit.

```
1  package br.ufpb.ppgi.cloudtesting.examples;
2
3  public class Sum {
4      private int value;
5
6      public int getValue() {
7          return value;
8      }
9
10     public void setValue(int value) {
11         this.value = value;
12     }
13
14     public void plus() {
15         this.value++;
16     }
17 }
```

Nesse exemplo, a classe Sum possui três métodos públicos. O método *getValue()* que retorna o valor da variável de instância *value*, o método *setValue* que define o valor da variável de instância *value* e o método *plus* que incrementa o valor armazenado pela propriedade *value*.

Figura 5 – Exemplo de um teste de unidade utilizando o *framework* JUnit.

```
1  package br.ufpb.ppgi.cloudtesting.examples;
2
3  import static org.junit.Assert.*;
4  import org.junit.Test;
5
6  public class TestForExample {
7
8      private Sum count;
9
10     public TestForExample() {
11         this.count = new Sum();
12     }
13
14     @Test
15     public void testAdd() {
16
17         this.count.setValue(7);
18
19         this.count.plus();
20
21         assertEquals(8, this.count.getValue());
22     }
23 }
```

Conforme exibe a Figura 5, a classe *TestValue* possui um único método público denominado *testAdd()*, a anotação *@Test* indica que o método é um *TestCase*, esse método examina se o método *plus* da classe *Sum* funciona corretamente, essa verificação é feita por meio do método *assertEquals* do JUnit que valida se o valor esperado é o mesmo valor retornado pelo método *getValue()* da classe *Sum*.

2.3 Testes Automáticos de *Software*

O teste pode ser efetuado tanto de modo manual quanto de modo automatizado. A problemática na abordagem do uso de teste manual incide na necessidade de uma maior quantidade de tempo e conhecimento técnico do desenvolvedor/testador para obter resultados, se comparado a um processo de testes automatizado [Mazer e Loring 2008]. Além disso, testes não automatizados podem ser extremamente desfavoráveis, visto que, cada teste pode ser executado diversas vezes durante a fase de desenvolvimento do *software* [Duarte 2010] [Catelani *et al.* 2008]. Assim, em certas circunstâncias, o teste manual se torna inviável ou impraticável [Wu e Sun 2010]. Whittaker (2000) e Taipale *et al.* (2006) afirmam que testes automáticos devem ser concebidos e implementados para solucionar tal problema.

Testes automatizados são compostos por *frameworks* que controlam a execução dos casos de testes de modo automático analisando as funcionalidades do sistema que se deseja testar em busca de possíveis falhas [Eliane e Luana 2010]. Atualmente, a utilização de testes automáticos de *software* é uma das principais abordagens utilizadas para diminuir o custo, aumentar a confiabilidade e a eficiência do *software* [Shahamiri *et al*. 2009] [Catelani *et al*. 2008] [Patton 2002], visto que, todos os testes podem ser repetidos quantas vezes necessários de modo fácil e ágil, permitindo a execução de maiores quantidades de testes em um intervalo de tempo reduzido [Tuschling 2008].

Duarte (2010) destaca a importância do uso de teste automático de *software*, definindo-o como um conjunto de testes favorável ao desenvolvimento de aplicações de grande porte (por exemplo, *softwares* com milhões de linhas de código) por fornecer a capacidade de abstração entre equipes dispersas, além de analisar o comportamento do *software* de modo automático e rápido, exigindo menor esforço para efetuar comparações em uma dada implementação para avaliar se uma funcionalidade é executada de acordo com o previsto.

2.4 Computação Distribuída

A computação distribuída é um paradigma que possibilita a distribuição de processamento entre computadores distintos em uma rede de computadores. Esse modelo divide e executa tarefas de modo colaborativo utilizando os conceitos estabelecidos pela computação paralela, diferenciando-se do modo como os processos e a memória são distribuídos.

Coulouris *et al*. (2007) define computação distribuída como componentes de *hardware* e/ou *software* que atuam em redes de computadores através de mensagens enviadas por meio de

protocolos de comunicação. A computação distribuída permite que tarefas sejam executadas desfrutando do poder computacional de vários computadores ao mesmo tempo, possibilitando que grandes massas de dados sejam processadas com tempo reduzido. Por se tratar de uma sinergia, a computação distribuída não centraliza o processamento em grandes servidores, pelo contrário, a carga é distribuída em vários computadores com poder computacional limitado que enviam o resultado do processamento ao servidor quando a tarefa é finalizada.

Tanembaum (2003) ressalta que existe uma ampla confusão no entendimento do conceito de redes de computadores e sistemas distribuídos. Em um sistema distribuído, um conjunto de computadores independentes se comporta como um único sistema sob a perspectiva do usuário, sendo essa transparência estabelecida comumente por meio de *middlewares*. Para Wu (1999) computação distribuída é uma composição de entidades de computação que trabalham com um objetivo em comum.

Coulouris *et al*. (2007) caracteriza os sistemas distribuídos a partir das propriedades: (1) concorrência: cada máquina da rede processa as suas atividades, além de concorrer pelos recursos disponíveis; (2) inexistência de relógio global: a comunicação é realizada através de mensagens, isso implica na impossibilidade de se assegurar a existência de uma referência temporal para todos os elementos do sistema distribuído; (3) falhas independentes: a falha de um único membro não intervém na execução dos demais. Atualmente, os principais modelos de arquitetura de sistemas distribuídos adotados são: modelo cliente/servidor, modelo *peer-to-peer* (P2P) e o modelo de objetos distribuídos.

2.5 Computação em Nuvem

Computação nas nuvens é uma tecnologia emergente que é constituída por um *pool* de recursos computacionais de grande escala. Esse modelo possibilita que cada categoria de aplicação/sistema utilize somente a computação que necessita, tais como espaço em armazenamento e todos os serviços de *software*. Dessa forma, evitando desperdícios, proporcionando a redução de custos desnecessários [Zhang *et al.* 2010].

Nesse contexto, o *pool* é chamado de nuvem, que por sua vez, é composta de recursos de computação virtual auto gerenciáveis, sem intervenção humana. Comumente, a nuvem é composta por *clusters* de servidores de larga escala, o que abrange, por exemplo, servidores de cálculos. Essas características proporcionam uma menor concentração em detalhes de configuração e focam no que

realmente é importante, o negócio. Assim, sendo a computação nas nuvens adequada à inovação [Zhang *et al*. 2010].

Zhang, Chen, Zhang e Huo (2010) discutem que o princípio fundamental da nuvem é o de distribuir tarefas para múltiplos computadores, de modo que, a computação possa ocorrer remotamente, disponibilizando uma grande quantidade de recursos computacionais da forma mais transparente possível para o usuário. Para Mell e Grace (2009) computação nas nuvens é um paradigma apropriado para disponibilizar acesso sob demanda a um conjunto compartilhado de recursos de computação que podem ser rapidamente agrupados e posteriormente liberados com o mínimo de esforço.

Apesar de ser tratada como novidade, computação nas nuvens é o resultado da evolução contínua da computação distribuída, computação paralela e computação em grades. Além disso, sem o padrão interconexão, o desenvolvimento de protocolos e tecnologias maduras de montagem de *Data Center*, a nuvem não se tornaria realidade [Zhang, Chen, Zhang e Huo 2010] [Gong *et al* 2010].

Vaquero *et al.* (2009) e Gong *et al*. (2010) apresentam a composição da computação nas nuvens, discutindo que a nuvem é decomposta em três categorias: infraestrutura como serviço (IaaS), plataforma como serviço (PaaS) e *software* como serviço (SaaS) e cinco camadas (clientes, aplicações, plataforma, infraestrutura e servidores). IaaS fornece infraestrutura de computação como um serviço, clientes não necessitam adquirir servidores, *firewalls*, roteadores ou qualquer recurso de rede. Ao invés disso, pagam apenas pelo período de utilização [Jadeja e Modi 2012]. O PaaS disponibiliza toda uma plataforma de desenvolvimento utilizando a infraestrutura da nuvem. Por meio dele desenvolvedores obtêm todas as ferramentas necessárias para o ciclo de vida do *software*, o que inclui: desenvolver, testar, implantar e hospedar [Jadeja e Modi 2012]. SaaS oferece *software* em forma de serviço (*software* sob demanda), o cliente não necessita comprá-lo para desfrutar de suas funcionalidades. Basicamente o *software* é executado na infraestrutura da nuvem em uma única instância de execução, utilizando virtualização para atender a todos os clientes. Questões referentes à capacidade de processamento e armazenamento são vistas como utilidade que os clientes pagam apenas pelo necessário [Kalagiakos e Karampelas 2011].

De um modo geral, as propriedades que permeiam a nuvem compreendem a abstração dos detalhes de implementação, o fraco acoplamento, a tolerância a falhas, agilidade, dispositivo de localização e independência, alta confiabilidade, alta escalabilidade, segurança, sustentabilidade (virtualização), otimização e variedade de recursos, adaptação automática, SLAs (Acordo de nível de serviço) e o modelo de negócio econômico (*pay-per-use*) [Gong *et al*. 2010] e [Vaquero *et al.* 2009].

Por se tratar de uma evolução de outras tecnologias, o conceito de nuvem habitualmente é confundido. Buyya *et al.* (2009) distingue computação nas nuvens de *clusters* e computação em grades. Recursos de *cluster* localizam-se em um único local administrativo com entidade única. Recursos de computação em grade são distribuídos e localizados em diferentes domínios administrativos, com várias entidades e políticas de gestão. A computação na nuvem é uma combinação de características de ambas as tecnologias. Contudo, computação nas nuvens, computação em grades, computação de alto desempenho ou supercomputação, todos pertencem à grande área da computação paralela [Gong *et al*. 2010]. Ainda hoje, existe muita confusão entre as tecnologias de grades computacionais e computação nas nuvens. A Tabela 2 exibe as principais características de computação nas nuvens e computação em grades.

Tabela 2 – Diferenças entre Computação nas Nuvens e Grades Computacionais. Adaptado de [Vaquero *et al.* 2009].

Característica	**Computação nas Nuvens**	**Grade Computacional**
Compartilhamento de recursos	Recursos atribuídos não são compartilhados.	Através da colaboração.
Heterogeneidade de recursos	Agregação de recursos heterogêneos.	Agregação de recursos heterogêneos.
Virtualização	*Hardware* e plataformas de *software*.	Dados e recursos de computação.
Segurança	Através do isolamento.	Através de delegação de credencial.
Arquitetura	Definida pelo usuário.	Orientada a serviços.
Fluxo de trabalho do *software*	Fluxo de trabalho não é essencial para a maioria das aplicações.	Aplicações requerem um fluxo de trabalho pré-definido.
Escalabilidade	Nós e *hardware*.	Nós.
Usabilidade	Facilidade na utilização.	Difícil de gerir.
Modelo de negócio	Flexível.	Rígido.
Garantia da qualidade do serviço	Suporte limitado com foco na disponibilidade e *uptime*.	Suporte limitado.

A Tabela 2 apresenta uma comparação entre as propriedades de grades computacionais e computação nas nuvens, a seguir serão abordadas essas características. Em relação ao compartilhamento de recursos, nuvens disponibilizam recursos sob demanda, de modo que seja transmitida a impressão para o usuário de que existe apenas um recurso único e dedicado. As grades aumentam o compartilhamento de recursos entre diversas organizações utilizando o processamento de máquinas ociosas. Ambas, fornecem suporte para a agregação de recursos heterogêneos de *hardware* e de *software* [Vaquero *et al*. 2009].

Tendo em consideração a virtualização, grades virtualizam recursos de dados e de computação. As nuvens herdam essas mesmas propriedades, embora adicionem suporte a virtualização de *hardware*. Virtualização está intrinsicamente relacionada à segurança e arquitetura em nuvens computacionais, constituindo um fator chave. Nuvens utilizam a virtualização que provê um ambiente individual para seus usuários. As grades tratam da segurança por meio de delegação de credencial e requerem em sua arquitetura que suas aplicações sejam *gridificadas*, um critério rígido para os desenvolvedores. [Gong *et al*., 2010].

As nuvens não se preocupam com fluxo de trabalho, diferentemente das grades que são basicamente orientadas a serviço, necessitando realizar a coordenação do fluxo de trabalho e serviços de localização. Comparando questões de escalabilidade entre as duas, observa-se que as grades utilizam a adição de nós. Já as nuvens, oferecem redimensionamento automático de recursos de *hardware* virtualizados [Gong *et al*. 2010] [Vaquero *et al*. 2009].

A fácil usabilidade e o modelo de negócio são grandes diferencias das nuvens para o usuário final se comparada às grades [Gong *et al*. 2010]. Nuvens são fáceis de utilizar, abstraem os detalhes de implementação e utilizam o modelo *pay-per-use*. Já as grades requerem gerenciamento intensivo que em muitos casos é complexo, oferecem um modelo de negócio que geralmente cobram taxas fixas por cada serviço ou organização que compartilhem dos recursos ociosos. Outra diferença marcante se situa na qualidade de entrega de serviço (QoS). Grades não são completamente comprometidas com QoS, provavelmente devido ao modelo de colaboração de recursos compartilhados. Nas grades são as aplicações que devem garantir QoS por si só. Nas nuvens QoS é uma das características inerentes [Vaquero *et al*. 2009].

Na indústria empresas como a Amazon[12], Google[13], Microsoft[14] e IBM[15] oferecem serviços de computação nas nuvens. Na academia, encontramos iniciativas *open source* com características de

[12] Amazon Elastic Compute Cloud (EC2): http://aws.amazon.com
[13] Google Apps: https://www.appengine.google.com/start

pesquisas exploratórias, oferecendo aos pesquisadores a possibilidade de se aprofundarem diretamente com Computação nas Nuvens a um nível mais intenso que as soluções proprietárias permitem. Por exemplo, o *toolkit* Nimbus[16] [Bresnahan *et al.* 2011] e o serviço *Eucaliptus Community*[17] [Nurmi *et al*. 2008], OpenNebula, OpenQRM, TPlataform, XCP, Apache Virtual Computing Lab(VCL), Enomaly Elastic Computing Platform.

2.6 Frameworks

Reutilizar código e design de *software* é um dos princípios fundamentais da Engenharia de *software* [Mattsson e Bosch 1997]. Isso se dá devido à necessidade de desenvolver artefatos em passo acelerado, com qualidade e a um custo reduzido [Prota 2012]. Ainda que alcançar tais objetivos seja um desafio para os engenheiros de *software* [Mian e Natali 2001], desenvolver sistemas por meio de uma abordagem de construção que alia componentes reutilizáveis é uma alternativa viável adotada há muito tempo, possuindo como abordagens iniciais a reutilização somente de pequenos blocos de construção de componentes.

Com a evolução das técnicas de reuso e dos paradigmas de desenvolvimento de aplicações surgem os *frameworks*, que são implementações de um projeto abstrato em relação a um determinado domínio de problema [Mattsson e Bosch 1997]. Silva e Price (1998) discutem que *frameworks* mapeiam uma implementação incompleta de um conjunto de domínio de aplicações, devendo ser adaptado com o intuito de gerar aplicações específicas. O fluxo de ideias de Feng Zhu e Wei-Tek Tsai (1998), Krajnc e Hericko (2003), Campbell *et al.* (1991) e Gamma *et al.* (1995) corroboram que *frameworks* são programas reutilizáveis parcialmente implementados que fornecem uma arquitetura geral para o desenvolvimento de uma aplicação de um domínio específico.

O principal benefício de utilizar *frameworks* é que desenvolvedores de *software* não precisam iniciar uma aplicação a partir do zero [Feng Zhu e Wei-Tek Tsai 1998], pois existe um alto nível de reutilização no desenvolvimento de aplicações, alcançando-se o aumento do nível de qualidade, uma redução do esforço de desenvolvimento do *software*, modularidade, extensibilidade e inversão de controle [Gimenes e Huzita 2005] [Júnior 2006].

Apesar das vantagens, desenvolver um *framework* é uma tarefa que requer esforço técnico considerável, visto que, entre os requisitos deseja-se flexibilidade e extensibilidade [Silva e Price

[14] Cloud Power: http://www.microsoft.com/pt-br/cloud/default.aspx
[15] IBM Cloud Computing: http://www.ibm.com/cloud-computing/us/en/
[16] Nimbus: http://www.nimbusproject.org/
[17] Eucalipstus Community: http://open.eucalyptus.com/

1998]. Logo, durante o planejamento de elaboração de um *framework* alguns pontos devem ser considerados, tais como: curva de aprendizagem, integração, eficiência, manutenção, validação, remoção de falhas e existência de poucos padrões para desenvolvimento de *frameworks* [Júnior 2006]. Silva e Price (1998) discute que o esforço necessário para um usuário aprender a desenvolver aplicações a partir de um *framework* é a principal barreira inicial. Portanto, relatam questões que um usuário deve tomar conhecimento ao utilizar um *framework*:

(1) Quais classes devem ser desenvolvidas para gerar uma aplicação e quais classes concretas podem ser reutilizadas?

(2) Para a criação de uma classe concreta para um aplicativo, quais métodos devem ser criados e quais métodos herdados podem ser reutilizados?

(3) Um usuário deve entender a semântica dos métodos, isto é, quais são as suas responsabilidades. Em que situações esses métodos serão executados e como fazer com que a implementação de um dado método estabeleça colaboração entre diferentes objetos?

Prota (2012) relata duas metodologias de desenvolvimento de *Frameworks*: *Example-driven design* e *Hot spot driven design*. O primeiro método analisa uma maior quantidade de exemplos do domínio com a intenção de visualizar toda a perspectiva do problema. Assim, identificando com maior eficiência as especificações e generalidades de cada aplicação. A segunda técnica procura por pontos de extensão, que se trata de estruturas do *framework* que fornece a possibilidade de adaptações do código para um funcionamento específico. Tais metodologias utilizam o mesmo modelo para desenvolvimento do *framework*: Análise do domínio, modelagem e teste.

Os *frameworks* possuem duas formas de classificação, relacionados à finalidade do uso e a forma de utilização. Podem ser agrupados de acordo com o escopo que atuam: problemas que buscam solucionar (horizontais, verticais e infraestrutura) e o modo no qual foi organizado e utilizado (caixa-branca, caixa-preta e caixa-cinza) [Prota 2012].

Frameworks horizontais se concentram na resolução de parte de um problema da aplicação, já os verticais são capazes de resolver toda ou grande parte do problema de um domínio. Os *frameworks* de infraestrutura destinam-se apenas em resolver problemas na camada de infraestrutura. *Frameworks* de caixa-branca dependem de recursos do paradigma orientado a objetos, como herança e ligação dinâmica para alcançar extensibilidade [Krajnc e Hericko 2003]. *Frameworks* de caixa preta, são instanciados através de composições, por meio da implementação de interfaces. Nessa categoria de *framework*, não existe preocupação em saber como as tarefas individuais são realizadas [Prota 2012] [Júnior 2006]. *Frameworks* de caixa-cinza são híbridos, possuem ambas as características dos *Frameworks* de caixa-branca e caixa-preta [Júnior 2006]. Apesar da energia gasta

durante o período de desenvolvimento, os benefícios ocasionados pelo uso de *frameworks* justifica o empenho da sua produção [Mattsson e Bosch 1997].

2.7 Considerações Finais

Este capítulo teve dois objetivos definidos. Inicialmente, apresentar os conceitos que comumente interessam pesquisadores e desenvolvedores de aplicações e frameworks acerca da temática de testes de *software* baseados na nuvem. As discussões efetuadas no conjunto de conhecimentos expostos nas seções anteriores fundamentaram o contexto e escopo do trabalho. Assim, produzindo intensidade para o levantamento das dificuldades, necessidades e limitações do *framework* a ser desenvolvido. Em seguida, auxiliar os leitores que possuem interesse em introduzir-se na área de Engenharia de *Software* e Computação nas Nuvens e não possuem especialização nesse campo de extensão. Portanto, provendo uma perspectiva geral sobre esse ecossistema.

O próximo capítulo objetiva investigar, relatar e examinar os principais trabalhos correlatos divididos em duas categorias: Testes Automáticos e Distribuídos de *Software* e Teste de *Software* Baseado na Nuvem, com o intuito de estabelecer quais contribuições já foram realizadas e quais problemas ainda perduram na área relacionada.

Capítulo 3

Trabalhos Relacionados

Este capítulo apresenta uma análise sintetizada sobre trabalhos relacionados à automação de Teste de *Software* em plataformas distribuídas, destacando as principais contribuições, limitações e identificando aspectos que possam ser adicionados ao *CloudTesting*. Primeiramente, discute-se sobre Teste de *Software* Automático Distribuído (seção 3.1) detalhando os trabalhos correlatos acerca do tema. Em seguida, trata-se a respeito dos trabalhos sobre Teste de *Software* Baseado na Nuvem (Seção 3.2). Por fim, o capítulo se encerra com as Considerações Finais (Seção 3.3).

3.1 Teste de *Software* Automático Distribuído

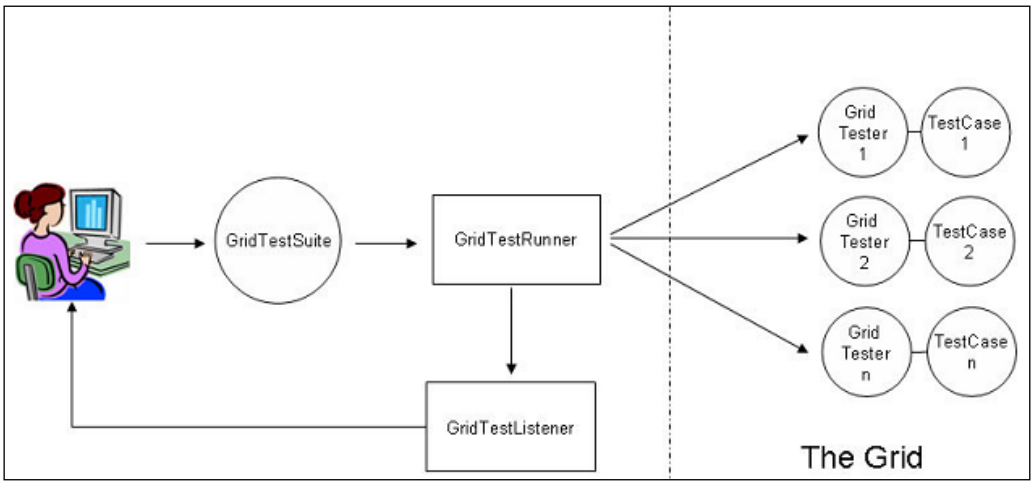
Observa-se que durante os últimos anos foram desenvolvidas várias pesquisas bem fundamentadas que tratam de soluções e meios para automatizar e acelerar o processo de teste de *software* [Gamma e Back 1999] [Kapfhammer 2001] [The Open Group 2003] [Hughes *et al.* 2004]. Apesar disso, à medida que os sistemas computacionais crescem e se tornam mais complexos o teste de *software* requer um maior esforço.

Diversos sistemas de teste de *software* automático distribuído ou de larga escala foram propostos nos últimos anos, visto que, essa metodologia explora as características de amplo paralelismo e vasta heterogeneidade de ambientes como forma de limitar os efeitos do ambiente de desenvolvimento sobre os resultados dos testes [Duarte *et al.* 2006].

3.1.1 GridUnit

A ferramenta *GridUnit* [Duarte *et al.* 2006] investiga o uso de grades computacionais como um ambiente de teste e apresenta uma solução *open source* para a execução de testes de unidade de *software* de modo automatizado na grade. A solução é uma extensão do *framework JUnit* [Gamma e Back 1999], capaz de distribuir a execução de uma suíte de testes *JUnit* na grade sem a necessidade de qualquer modificação no código fonte.

Figura 6 – Arquitetura conceitual do *GridUnit* [Duarte *et al.* 2006].



A Figura 6 representa a arquitetura conceitual do *GridUnit*, demonstrando a comunicação dos módulos e interfaces. A seguir serão discutidas as suas funcionalidades:

(1) **Distribuição transparente e automática**: cada teste *JUnit* é tido como uma tarefa independente e o agendamento da execução da tarefa na grade é feita sem intervenção humana;

(2) **Teste de prevenção de contaminação**: cada teste é executado adotando a virtualização oferecida pela grade, impedindo que o resultado do teste A1 altere o resultado do teste A2;

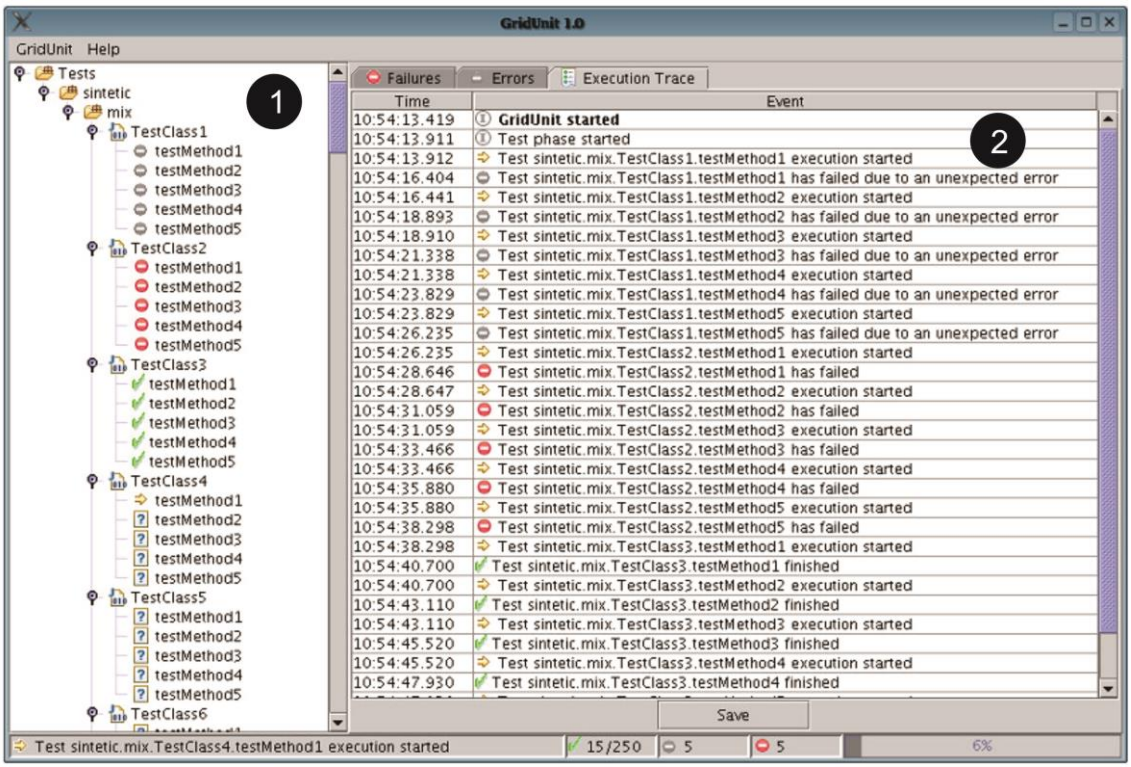
(3) **Distribuição de carga de teste**: a ferramenta possui o *Grid Scheduler* que gerencia a distribuição de carga;

(4) **Integridade da suíte de teste**: como citado anteriormente, cada teste *JUnit* é executado como uma tarefa independente. Para cada novo teste, o *GridUnit* cria uma nova instância da classe *Test*, efetua chamadas aos métodos *setUp(), testMethod(), tearDown()* e em seguida elimina a instância;

(5) **Controle de execução de teste**: As interfaces *GridTestRunner* e *GridTestListener* fornecem meios para estabelecer a execução e monitoramento de testes de modo centralizado.

Conforme pode ser visto na Figura 7, a interface gráfica do usuário disponibiliza opções para iniciar e parar a execução dos testes de uma suíte de teste específico (Área 1). Além disso, fornece monitoramento da execução dos testes, apresentando o resultado da execução de cada teste quando disponível (Área 2).

Figura 7 – Interface gráfica do *GridUnit* (Área 1 – Gerenciamento de Execução dos testes de unidade; Área 2 – Monitoramento da execução dos testes de unidade) [Duarte *et al.* 2006].



3.1.2 D-Cloud

O projeto *D-Cloud* [Banzai *et al.* 2010] propõe uma solução para teste de sistemas paralelos ou distribuídos de larga escala que necessitem das características de sistemas altamente confiáveis, mantendo o foco em teste de tolerância a falhas em nível de *hardware*. A pesquisa introduz a infraestrutura de computação nas nuvens para o teste de *software*, sendo composto por múltiplos nodos de máquina virtual, que executam sistemas operacionais hospedados com injeção de falhas, um nó controlador gerencia todos os sistemas operacionais hospedados, e um *frontend*, que controla as configurações de *hardware*, *software* e os cenários de teste.

A Figura 8 apresenta a arquitetura conceitual da ferramenta, no qual se destaca as seguintes propriedades:

(1) **Máquina virtual com injeção de falhas**: possui a *FaultVM* que é baseada na QEMU como o *software* de virtualização;

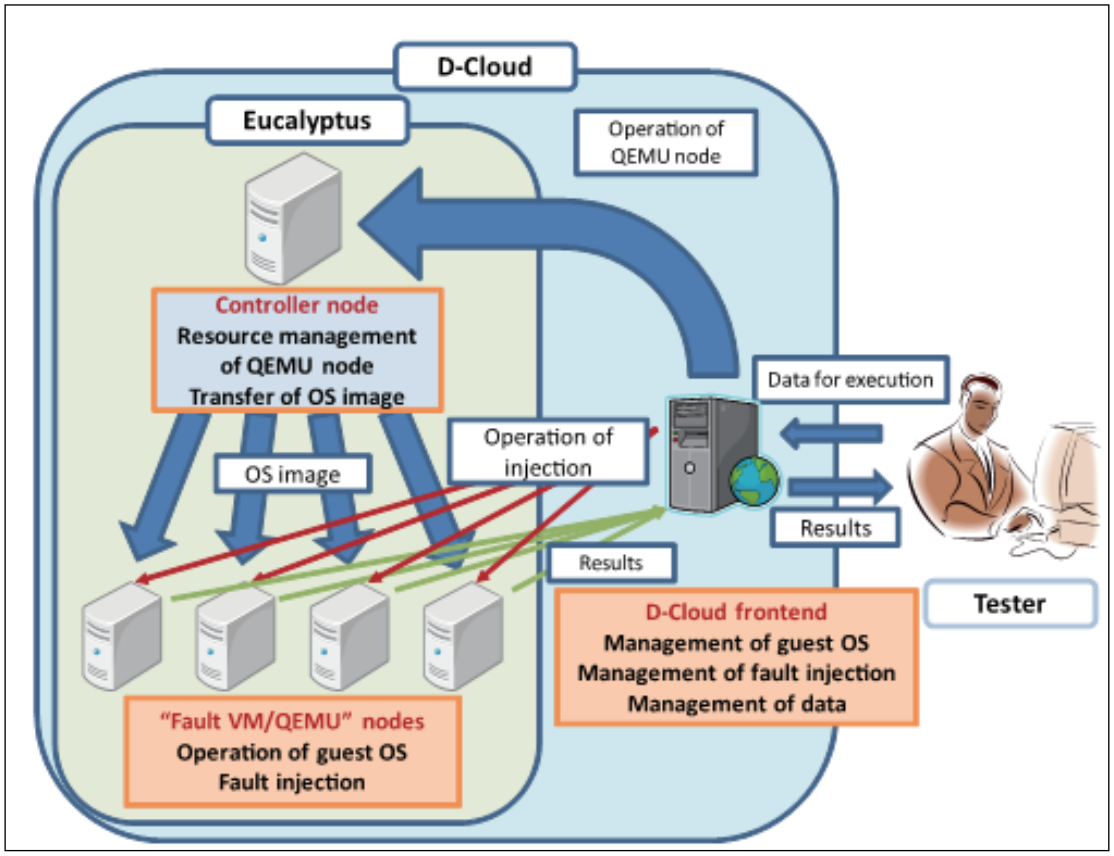
(2) **Gerenciamento de recursos computacionais usando *Eucalyptus***: para gerenciar a grande quantidade de recursos o *software Eucalyptus* é utilizado para efetuar o gerenciamento da

nuvem. Assim, eximindo a responsabilidade do testador de gerenciar a alocação de recursos computacionais, esse processo é feito de modo automático;

(3) **Configuração e testes automatizados do sistema**: a ferramenta automatiza a configuração do sistema e o processo de teste, compreendendo a injeção de falhas, baseando-se em cenários descritos por um testador;

(4) **Preparação dos cenários de teste**: é estabelecida por meio de um arquivo escrito em XML, ao fornecer vários arquivos de cenário, vários sistemas podem ser testados paralelamente. A Figura 8 ilustra as propriedades supramencionadas.

Figura 8 – Arquitetura Conceitual do *D-Cloud* [Banzai *et al.* 2010].

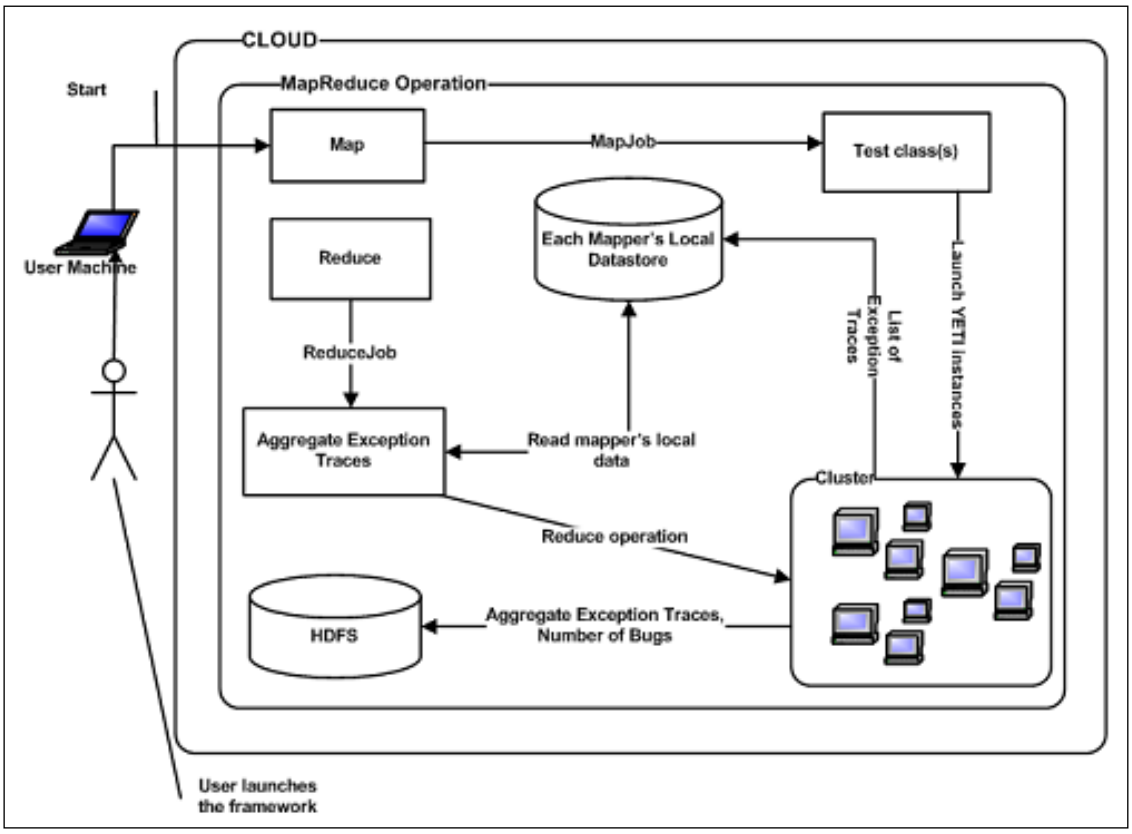


3.1.3 YETI On The Cloud

YETI on the Cloud [Oriol e Ullah 2010] é um projeto que migra as funcionalidades oferecidas pela ferramenta *stand alone* YETI para a nuvem, fornecendo testes aleatórios exclusivamente para as aplicações desenvolvidas na linguagem de programação Java.

A adoção de uma infraestrutura de computação nas nuvens resolve problemas de desempenho (introduz o paralelismo) e segurança (isolamento por meio de uso de máquinas virtuais).

Figura 9 – Arquitetura conceitual do *YETI on the Cloud* [Oriol e Ullah 2010].



Como pode ser visto na Figura 9, a implementação do projeto toma por base o *Hadoop*[18] para distribuir sessões de teste sobre máquinas remotas e recombiná-las ao final de um tempo de teste desejado. A distribuição se baseia em arquivos de textos gerados por meio do YETI *stand alone*. Desse modo, os comandos armazenados nesses arquivos são mapeados para máquinas na nuvem. Ao término dos testes, todas as exceções lançadas são escritas em arquivos e enviadas para o sistema de arquivo distribuído e disponibilizadas para a máquina do usuário, tornando-se disponível para avaliação do testador.

3.1.4 Análise Comparativa

Dentre as ferramentas abordadas nessa seção, o *GridUnit* é a que mais se assemelha a proposta do *CloudTesting*. Isso ocorre devido ao compartilhamento da natureza do contexto de ambas as pesquisas. Os dois projetos visam distribuir a execução de suítes de testes *JUnit* de modo transparente e automático. Contudo, o *CloudTesting* difere no que diz respeito a infraestrutura e abstração de complexidades.

[18] Hadoop: http://hadoop.apache.org/

O *GridUnit* utiliza grades computacionais e o *CloudTesting* apoia o uso de diferentes plataformas de execução distribuída para executar suítes de testes automáticos de uma aplicação utilizando uma variedade de ambientes de execução, como por exemplo, a nuvem. Há um ganho em utilizar nuvens nesse cenário (como pode ser visto na seção 0). Diferentemente das grades a nuvem oferece redimensionamento automático de recursos de *hardware* virtualizados, segurança por meio de virtualização, dispensa da preocupação com fluxo de trabalho, fácil gerenciamento, usabilidade e modelo de negócio flexível. Em relação à abstração de complexidade, diferente do *GridUnit*, o *CloudTesting* pretende integrar-se de forma simbiótica a ambientes de desenvolvimento integrados, omitindo configurações complicadas para o seu uso. A Tabela 3 exibe de modo analítico as diferenças comentadas entre o *GridUnit* e o *CloudTesting*.

Tabela 3 – Comparação entre as ferramentas *GridUnit* e *CloudTesting*.

Características / Recursos	*CloudTesting*	*GridUnit*
Execução de suítes de testes *JUnit*	O	O
Processamento Paralelo	O	O
Variedade de ambientes de execução	O	X
Integração com IDE'S	O	X
Arquitetura flexível	O	X

O *D-Cloud* possui um contexto e uma abordagem diferente para a execução de teste de *software* automático se confrontado com a proposta do *CloudTesting*. Basicamente, o direcionamento do teste de *software* automático diverge. O primeiro projeto orienta os seus testes para a tolerância a falhas em nível de *hardware*, a segunda pesquisa aponta para a execução de um conjunto de testes de unidade fazendo uso do *framework JUnit*. Além disso, o *D-Cloud* foi elaborado para funcionar exclusivamente para a infraestrutura de computação nas nuvens, ignorando outras plataformas e modos de execução. Por ser um *framework*, o *CloudTesting* pode ser adaptado para necessidades específicas. A Tabela 4 exibe as principais diferenças de ambos os projetos.

Tabela 4 – Comparação entre a ferramenta *D-Cloud* e o *CloudTesting*.

Características / Recursos	*CloudTesting*	*D-Cloud*
Execução de suítes de testes *JUnit*	O	X
Execução de Injeção de falhas de *hardware*	X	O
Processamento Paralelo	O	O
Virtualização de *hardware*	O	O

Redimensionamento automático de recursos de *hardware* virtualizados	O	O
Modelo de negócio flexível	O	O
Integração com IDE'S	O	X
Variedade de ambientes de execução	O	X

O *YETI on the Cloud* é uma solução proposta para eliminar as limitações encontradas no YETI *stand alone*. Se comparado ao *CloudTesting*, observar-se as seguintes diferenças:

(1) *YETI on the Cloud* efetua teste de *software* randômico automático e o *CloudTesting* efetua teste de *software* de unidade automático distribuído;

(2) *YETI on the Cloud* suporta apenas uma infraestrutura, por se tratar de um *framework* o *CloudTesting* pode ser customizado para suportar várias;

(3) *YETI on the Cloud* gera um arquivo que atua como uma espécie de log para avaliação do usuário, *CloudTesting* pode ser utilizado juntamente com ambientes de desenvolvimento.

A Tabela 5 apresenta uma comparação analítica.

Tabela 5 – Comparação das ferramentas *YETI on the Cloud* e *CloudTesting*.

Características / Recursos	*CloudTesting*	*YETI on the Cloud*
Execução de suítes de testes *JUnit*	O	X
Execução de teste randômico	X	O
Processamento Paralelo	O	O
Virtualização de *hardware*	O	O
Redimensionamento automático de recursos de *hardware* virtualizados	O	O
Modelo de negócio flexível	O	O
Integração com IDE'S	O	X
Variedade de ambientes de execução	O	X

Após análises individuais, a Tabela 6 efetua um exame sobre o apanhado de funcionalidades de cada ferramenta descrita nas seções anteriores. Utiliza-se um critério baseado em pesos, atribuindo valores de 0 (zero) e 1. O valor zero significa a inexistência ou incerteza do recurso, o valor 1 (um) representa a existência do recurso.

Tabela 6 – Análise das características das ferramentas *CloudTesting*, *GridUnit*, *D-Cloud* e *YETI on the Cloud*.

	CloudTesting	***GridUnit***	***D-Cloud***	***YETI on the Cloud***
Execução de suítes de testes *JUnit*	1	1	0	0
Processamento Paralelo	1	1	1	1
Variedade de ambientes de execução	1	1	0	0
Integração com IDE'S	1	0	0	0
Arquitetura flexível	1	0	0	0
Execução de Injeção de falhas de *hardware*	0	0	1	0
Virtualização de *hardware*	1	0	1	1
Redimensionamento automático de recursos de *hardware* virtualizados	1	0	1	1
Modelo de negócio flexível	1	0	1	1
Execução de teste randômico	0	0	0	1
Somatório dos pontos	**8**	**3**	**5**	**5**

3.2 Teste de *Software* Baseado na Nuvem

Essa seção apresenta soluções que implementam o conceito *Test as a Service*. Ferramentas que utilizam essa metodologia não só utilizam a nuvem para fornecer serviços, como também às hospedam.

Testes baseados em computação na nuvem encontram relativamente poucas barreiras para a sua aceitação. Isso se dá devido ao seu modelo de negócio que possibilita a redução de custos, sendo desnecessária a compra e a manutenção de ambientes de testes internos [Grundy *et al.* 2012].

Atualmente, várias ferramentas de Teste de *Software* estão disponíveis para uso com muitas finalidades, tais como: testes de desempenho, testes de carga, testes de aplicativos baseados na Web, bem como os testes de ambientes hospedados na nuvem.

3.2.1 Soasta

Soasta é uma ferramenta comercial que fornece teste na nuvem com foco em soluções para aplicações Web (HTML5, Flash,/Flex e Silverlight, SOAP e REST) e aplicações móveis. Possui uma plataforma integrada para testes funcionais baseados na aplicação Selenium fornecendo um ambiente visual de criação e gravação de teste e execução de teste distribuído. Além disso, também disponibiliza testes de desempenho para equipes de qualquer tamanho buscando eliminar possíveis barreiras para utilização de uma abordagem de testes de desempenho realizados regularmente, como parte de uma fase inicial do processo de compilação e continuo durante todo o ciclo de vida de entrega do *software* [Riungu-Kalliosaari *et al.* 2012].

A Figura 10 apresenta os módulos do Soasta:

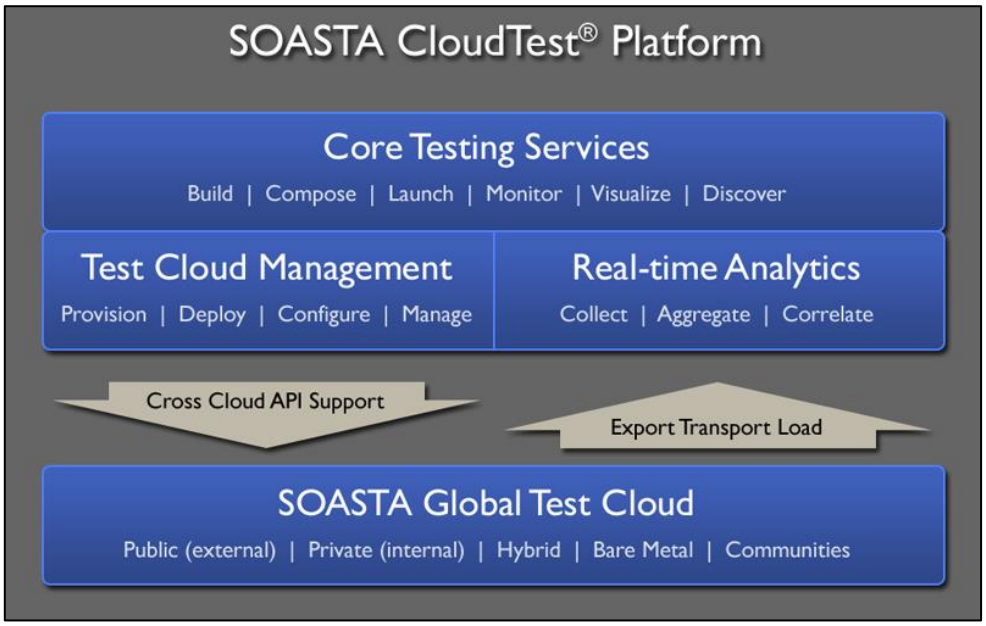
(1) ***Core Testing Services*** é responsável por carregar construir, executar, exibir e descobrir problemas durante o período de testes funcionais e de desempenho;

(2) ***Test Cloud Management*** configura e implanta o *hardware* da nuvem para a condução dos testes;

(3) ***Real-Time Analytics*** é o mecanismo de processamento analítico construído para entrega instantânea de ponta a ponta de métricas durante os testes;

(4) ***The Global Test Cloud*** fornece acesso integrado as principais plataformas de nuvens públicas e privadas.

Figura 10 – Arquitetura da ferramenta Soasta [Soasta 2012].

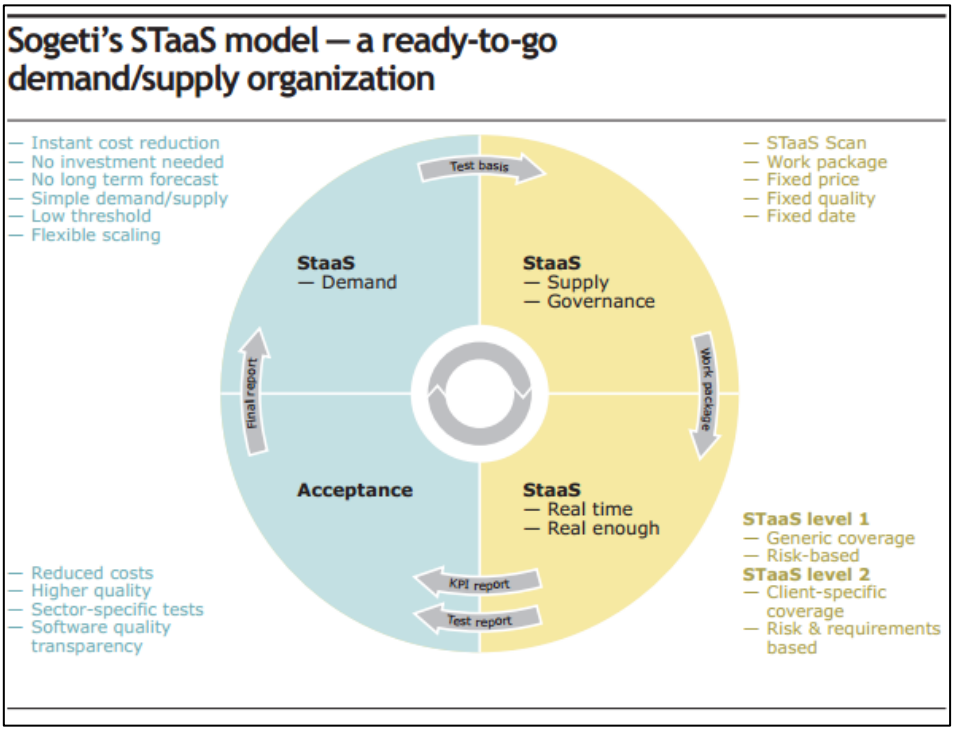


3.2.2 Sogeti StaaS

Sogeti StaaS é uma ferramenta comercial que oferece teste de software como serviço, buscando ser altamente acessível, flexível e de baixo custo para a execução de teste de software. A solução é baseada claramente na oferta e procura, onde modelos de testes são fornecidos em uma base de serviços sob demanda. Necessariamente, o usuário disponibiliza todo o *software* e sua documentação para serem testados, a partir desse ponto os especialistas da Sogeti efetuam os testes definidos e fornecem um relatório de qualidade [Robinson e Ragusa 2011] [Riungu, Taipale e Smolander 2010] [Riungu-Kalliosaari *et al.* 2012] .

A Figura 11 ilustra o modelo de negócio adotado pela Sogeti STaaS. Primeiramente define-se o âmbito e conteúdo do teste com o cliente visando identificar os riscos mais importantes. Após isso, uma varredura é realizada por um gerente de teste a partir da documentação entregue pelo cliente. Em seguida, o teste é executado em tempo real usando uma conexão entre o cliente e a Sogeti. Contudo, existem dois tipos de execução, que podem ser selecionadas de acordo com a quantidade e clareza da documentação fornecida do aplicativo a ser testado.

Figura 11 – Fluxo padronizado do processo StaaS [STaaS 2009].



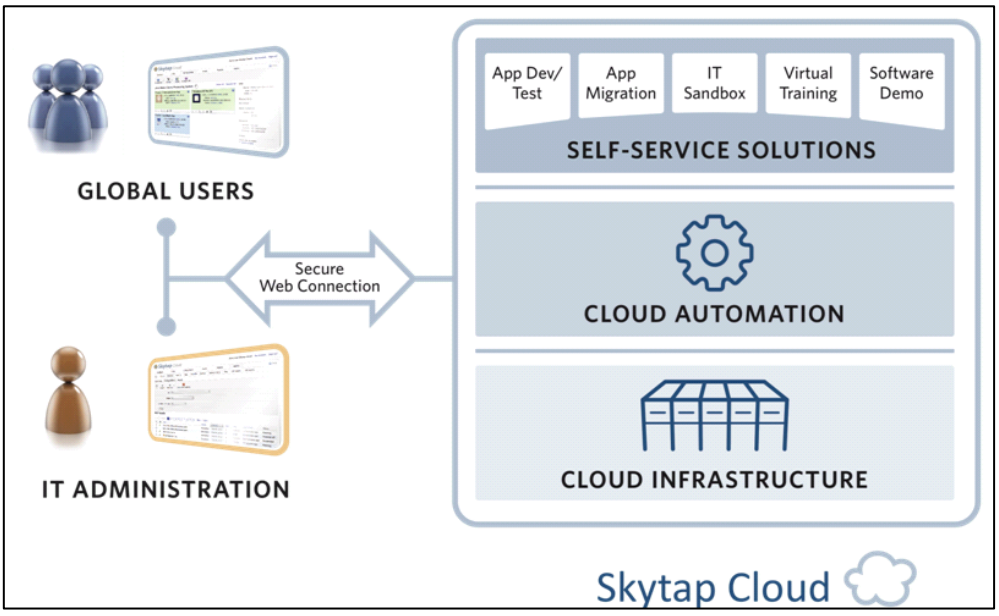
StaaS nível 1 é utilizada em situações onde a documentação do aplicativo é mínima ou inexistente. Assim, a Sogeti aplica um conjunto de testes padrão para a atividade especificada, fornecendo apenas informações sobre a qualidade básica da aplicação. Já a StaaS nível 2 é adotada

quando os requisitos e a documentação da aplicação existem e são bem definidos. Desse modo, um teste personalizado é executado com base na estratégia acordada com o cliente, usando a rede de conjuntos de testes da Sogeti, permitindo uma resposta rápida e a capacidade de oferecer acesso a recursos flexíveis.

3.2.3 SkyTap

Skytap disponibiliza recursos para a automação de computação nas nuvens com foco no desenvolvimento, teste, migração e avaliação de *software* [Riungu-Kalliosaari *et al.* 2012]. Para isso, a ferramenta fornece uma *interface self-service* para seus usuários, que permite que os mesmos colaborem globalmente, conforme ilustra a Figura 12.

Figura 12 – Soluções self-service do SkyTap [SkyTap 2012].



Skytap Development and Test Cloud torna possível à criação e configuração de múltiplos data centers virtuais (VDC), possibilitando a customização de *hardware* e *software*, além da execução em paralelo. A ferramenta em questão pode ser adotada em diversos cenários de teste de *software*, tais como: Testes unitários, testes de integração, testes de sistema, teste de aceitação e teste de desempenho.

3.3.3 Análise Comparativa

As ferramentas abordadas na seção 3.2 possuem um foco diferenciado em relação à proposta do *CloudTesting*. Basicamente, as soluções analisadas oferecem serviços de teste na nuvem, ao invés de um *framework* de desenvolvimento. Contudo, o SkyTap afirma que suporta testes de unidade, distinguindo-se da Soasta que emprega esforços para testes funcionais e de desempenho para tecnologias Web e móveis listadas na seção 3.2.1 e individualizando-se da Sogeti StaaS descrita na seção 3.2.2 que adota um modelo de negócio baseado em acordos entre seus clientes. A Tabela 7 exibe uma avaliação de cada solução analisada com base no escopo do *CloudTesting*. A análise utiliza um critério baseado em pesos, atribuindo valores de 0 (zero) e 1. O valor zero corresponde a inexistência ou incerteza do recurso, o valor 1 (um) representa a existência do recurso.

Tabela 7 – Análise comparativa das soluções para Teste de *software* baseado na nuvem.

	CloudTesting	***Soasta***	***Sogeti StaaS***	***SkyTap***
Framework de Desenvolvimento	1	0	0	0
Open Source	1	0	0	0
Adaptação e/ou customização do serviço.	1	0	1	0
Integração com IDE'S	1	0	0	0
Teste Unitário	1	0	0	1
Teste de Desempenho	0	1	0	1
Teste de Integração	0	0	0	1
Teste de Sistema	0	0	0	1
Teste de Aceitação	0	0	0	1
Tipo de Teste definido pelo usuário (Unidade, funcional, Integração, etc.)	0	0	1	0
Relatório de Qualidade/erros	1	1	1	1
Somatório dos pontos	**6**	**2**	**3**	**6**

3.4 Considerações Finais

A análise dos trabalhos correlatos auxiliou no mapeamento das características e necessidades no panorama dos limites de Teste de *Software* Automático Distribuído e Teste de *Software* Baseado na Nuvem. Com o resultado do apanhado das informações levantadas, verificam-se vantagens na concepção do projeto proposto. Além disso, por meio desse exame esquematizou-se o âmbito das possíveis contribuições e dificuldades que persistem em cada área.

O próximo capítulo objetiva apresentar de modo detalhado a estrutura da ferramenta *CloudTesting*, que trata-se de um *framework* para teste de unidade de *software* baseados na nuvem, que origina contribuições principalmente nas duas grandes áreas analisadas nesse capítulo.

Capítulo 4

CloudTesting

Este capítulo apresenta detalhes da concepção do *framework* para testes automáticos distribuídos *CloudTesting*. Primeiro, é realizada uma definição do projeto (Seção 4.1). Em seguida, são detalhados os requisitos e a identificação de entidades (seção 4.2). Após isso, é apresentado o uso do *framework* (seção 4.3). Por último (seção 4.4), são discutidas as considerações finais.

4.1 Definição

O framework CloudTesting, é uma solução que visa auxiliar o desenvolvimento de teste de unidade de *software* nas nuvens, agilizando o processo de testes automáticos, disponibilizando uma abordagem que permite a distribuição de conjuntos de testes em ambientes de nuvem para execução paralela. A proposta visa atacar a realidade encontrada em grandes projetos de *software* que possuem uma ampla quantidade de testes, o que, em muitos casos, torna-se um fator decisivo para inviabilizar qualquer processo de desenvolvimento que se baseie fortemente no uso de testes automáticos [Wu e Sun 2010], devido à necessidade do consumo de uma grande quantidade de tempo de execução contínua para finalizar cada rodada de testes [Duarte *et al.* 2006].

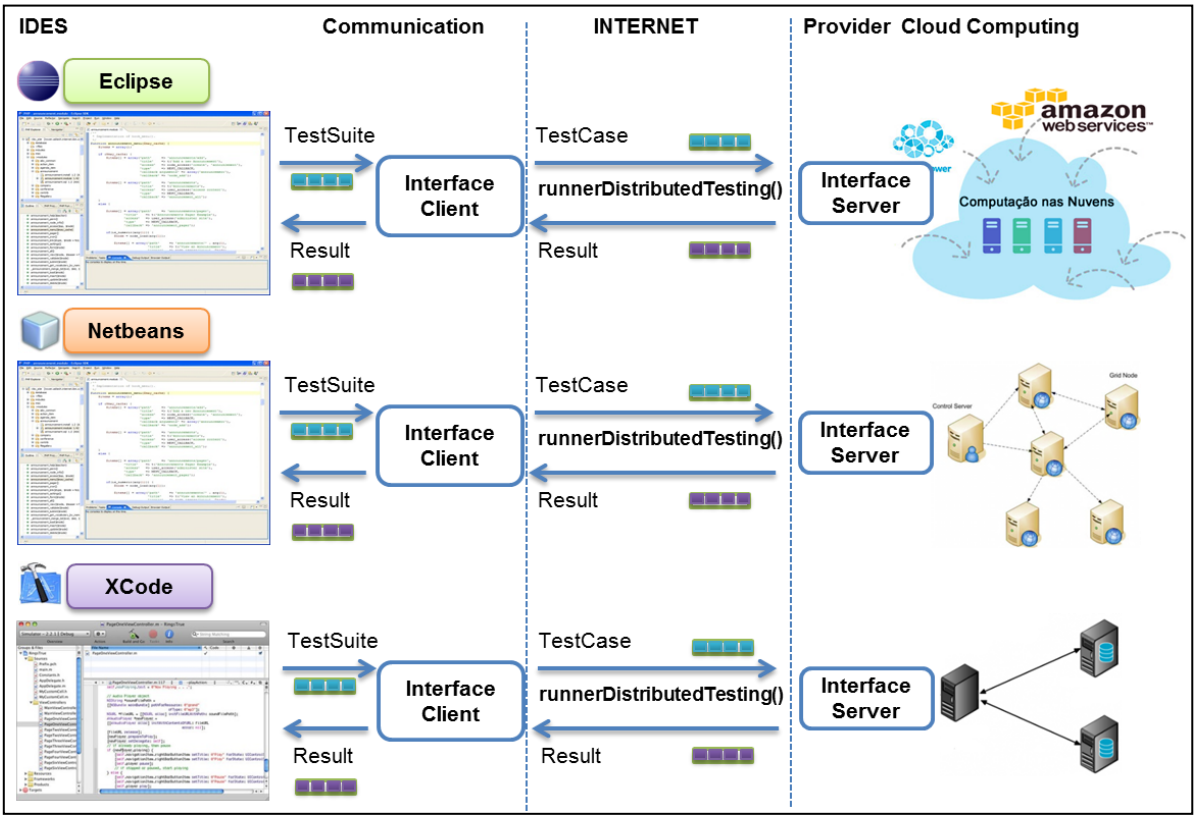
É natural que cada teste de unidade consuma um determinado tempo para cada ciclo de execução, esse período varia de acordo com os fatores relacionados a tamanho e complexidade do *software*, além da capacidade computacional do *hardware* em que o teste é efetuado. Nesse contexto, o único modo de decrementar o tempo gasto no processo é paralelizar massivamente a sua execução [Banzai *et al.*, 2010].

Diferentes estudos apoiam a iniciativa de aumentar o desempenho dos testes tradicionais adicionando características de execução distribuída e concorrente [Duarte *et al.* 2006] [Oriol e Ullah 2010] [Banzai *et al.* 2010]. Apesar disso, divergem quanto à finalidade ou tecnologias empregadas neste trabalho. A alternativa elaborada no *CloudTesting* é fornecer uma solução que possibilite acrescentar desempenho por meio de processamento paralelo utilizando Computação nas Nuvens, buscando abstrair essa metodologia, encapsulando a maior parte da complexidade envolvida no processo de execução e controle dos testes na nuvem.

Isso se torna possível devido à adição de uma nova camada durante a etapa de teste utilizando o *CloudTetsing*, obtendo de forma dinâmica os recursos computacionais necessários para a execução dos testes, sem exigir qualquer modificação no código fonte dos testes para sua utilização. Quando se decide distribuir e paralelizar a execução dos testes, obtém-se uma redução significativa do tempo necessário para executar um grande conjunto de testes, diminuindo, por sua vez, o tempo despendido na identificação e correção de falhas no *software*, o que representa um grande impacto no custo total de desenvolvimento. Além disso, o *framework* possui vantagem ao aumentar a confiabilidade nos resultados dos testes de unidade por meio da utilização de ambientes heterogêneos e sabidamente não contaminados para a execução dos testes, facilitando a exposição de falhas que de outra maneira só seriam encontradas durante a fase de execução do sistema.

Conforme apresenta a Figura 13, fundamentalmente o *framework* distribui o projeto que contém o conjunto de testes para a nuvem, após isso efetua reflexão nas classes locais desse conjunto de testes enviando requisições de execução para um método de modo independente, paralelizando ao máximo a execução das unidades. O balanceamento de carga de recursos da rede é baseada na implementação do Algoritmo *Round-Robin* [Ramabhadran e Pasquale 2003]. Assim, todas as requisições são distribuídas de modo uniforme entre as máquinas da nuvem participantes do balanceamento.

Figura 13 – Fluxo de execução do Teste de unidade de utilizando o *framework CloudTesting*.



Existe também a possibilidade que o *CloudTesting* seja adaptado para ser utilizado para permitir uma varredura exploratória de diferentes ambientes e configurações, visto que, a metodologia de Computação nas Nuvens se baseia na virtualização para prover ambientes distintos. A proposta é a de gerar a árvore de configurações possíveis, tal como diferentes versões de bibliotecas suspostamente compatíveis, para posterior execução. Contudo, o escopo desse trabalho se atém a utilização do *CloudTesting* como forma de agilizar o processo de testes.

4.2 Especificação

O processo de especificação procura estabelecer quais são as funcionalidades e as restrições do projeto. Com o intuito de desenvolver uma solução que abranja as tendências de mercado, desenvolveu-se um *survey* (discutido na seção 1.2). Após a análise dos resultados, comprovou-se que o *framework* JUnit é amplamente utilizado. Por esse motivo, o *CloudTesting* focou inicialmente nessa tecnologia. A seguir são descritos os requisitos funcionais e não funcionais do *framework*.

4.2.1 Requisitos

Os requisitos se baseiam na necessidade da resolução da proposta do projeto em comparação com as soluções relacionadas ao domínio do problema descrito no Capítulo 3. Na Tabela 8, são abordados os requisitos não funcionais desejados. Entretanto, a Tabela 9 exibe os requisitos não funcionais que devem ser atendidos pelo *framework*.

Tabela 8 – Requisitos funcionais do *CloudTesting*.

Identificador	Descrição
RF01	Mapear hosts de máquina na nuvem
RF02	Identificar todos os métodos que devem ser distribuídos para nuvem
RF03	Distribuir projetos JUnit para nuvem.
RF04	Executar teste de unidade de modo paralelo.

RF05	Escalonar distribuição de projetos JUnit

Tabela 9 – Requisitos Não-funcionais do *CloudTesting*.

Identificador	Descrição
RNF01	Adotar padrões de desenvolvimento de *software* e *framework*;
RNF02	Disponibilizar uma versão do *framework* para avaliação e testes experimentais.
RNF03	Obter um *speedup* aceitável, considerando a flutuação da rede.
RNF04	Disponibilizar documentação sobre o projeto.

4.2.3 Identificação das Entidades

A primeira etapa para a construção de um sistema baseado em Programação Orientada a Objetos é a identificação das entidades. Um modo de identificá-los é buscar visualizar o problema de um modo geral, considerando abstrair os substantivos que compõem um processo de teste de unidade de *software* de modo distribuído.

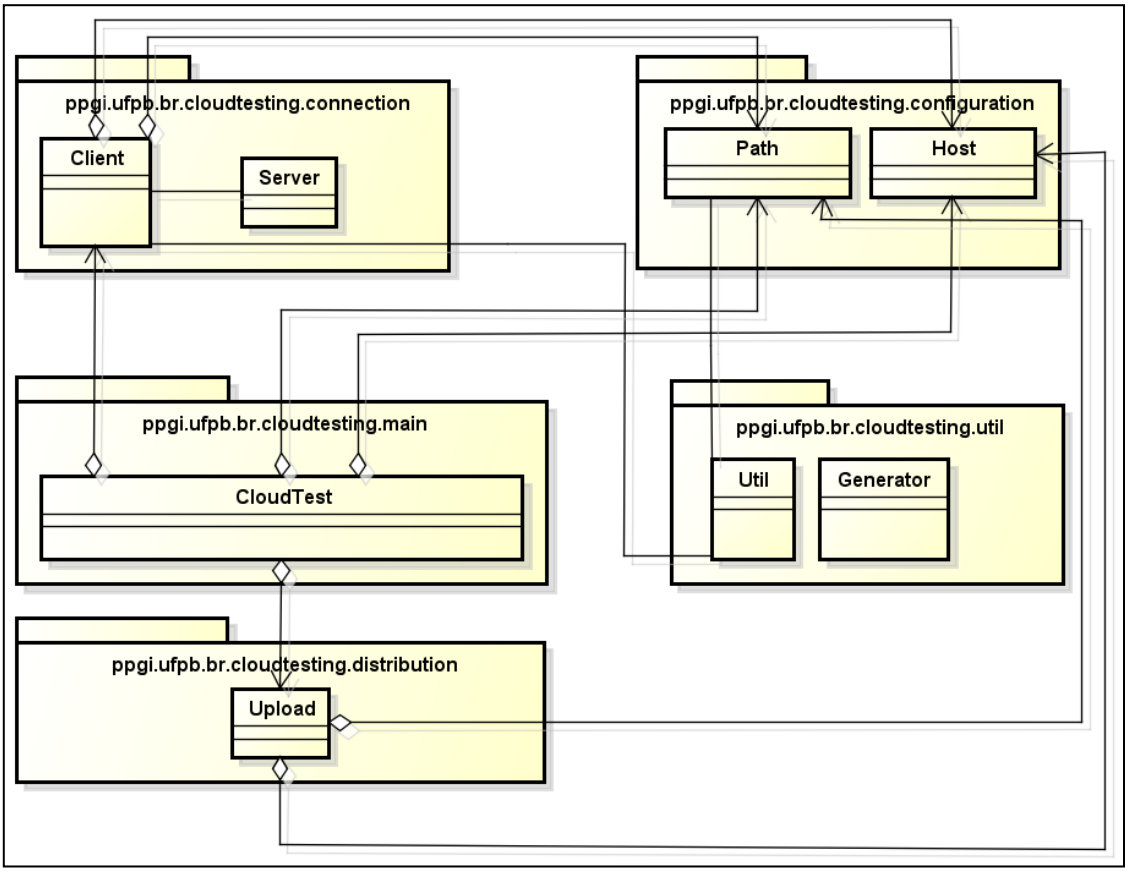
Durante a análise, foram identificadas as entidades apresentadas no Diagrama de Pacotes ilustrado na Figura 14. A classe Path concentra as informações relacionadas ao caminho de um dado recurso, tal como arquivo de permissão de acesso e diretórios de logs e bibliotecas. A classe Host, possui um conjunto de endereços de instâncias que serão administradas pelo algoritmo de balanceamento Round-Robin, tais endereços podem ser compostos de máquinas com arquiteturas heterogêneas (tais como: Windows 32 bits, Linux 64 e etc). A classe Util, fornece funcionalidades comuns entre as classes. A classe Generator, disponibiliza a criação de uma classe de teste com 1800 testes de unidade. A classe Upload, distribui projetos de teste de unidade para as instâncias na nuvem utilizando o protocolo SCP. A classe Client, fornece uma interface de comunicação entre o cliente e o serviço da nuvem. A classe Server, ativa o serviço de execução de teste de unidade para uma dada

instância na nuvem. A classe CloudTest efetua reflexão nas classes de testes do cliente e envia requisições de execução de um determinado método para as instâncias da nuvem.

Na linguagem de modelagem UML, um pacote é uma estrutura que visa constituir entidades em grupos [Booch *et al.* 1999]. Objetivando uma maior estruturação para sub-visualização lógica do framework as classes desenvolvidas para o *CloudTesting* foram inseridas em pacotes e agrupadas de acordo com a semelhança das funcionalidades, conforme pode ser visto na Figura 14. Assim, foram criados os seguintes pacotes:

- **Connection**: Contém classes de comunicação entre cliente e servidor;
- **Configuration**: Contém classes para configuração de *paths e hosts*;
- **Main:** Contém classe principal que dispara requisições para nuvem;
- **Distribution**: Contém classe que efetua *upload* de todo o projeto do usuário para as instâncias da nuvem;
- **Util**: Contém classes utilitárias.

Figura 14 – Diagrama de Pacotes do *framework CloudTesting*.



4.3 Uso do *Framework*

Por se tratar de um *framework* o *CloudTesting* pode ser utilizado em vários ambientes de desenvolvimento integrado (IDE), além de diversificadas plataformas de execução. No lado cliente, para utilizá-lo basta adicionar a biblioteca ao *build path* do IDE e em seguida importar o pacote *CloudTestingdi.main.cloudtest* e chamar o método *CloudTest.runnerDistributingMethods* informando o nome do pacote, um *array* contento o nome das classes de testes de unidade, o arquivo de permissão de acesso da instância da nuvem e o *path* completo do projeto no cliente. A Figura 15 ilustra o uso do *framework* em uma instanciação para o IDE Eclipse.

Figura 15 – Utilização do *framework CloudTesting*.

```
package main;

import cloudtestingdi.main.CloudTest;

public class GoTest {
    public static void main(String[] args) {
        String[] suite = {"tests.UnitStress"};

        CloudTest.runnerDistributingMethods("usingCloudTesting", suite,
                "/home/gustavo/coding/perm/cloudtesting.pem",
                "/home/gustavo/coding/workspace/usingCloudTesting");
    }
}
```

Contudo, na nuvem as instâncias devem executar o modo servidor do *CloudTesting* por meio do comando *java -classpath lib/CloudTesting.jar:. CloudTesting.connection.Server &* e terem previamente criado o diretório *lib* adicionado as bibliotecas *CloudTesting.jar* e do *junit.jar* e criado o diretório diretório log.

Como pode ser observado na Figura 15 não houve a necessidade de efetuar nenhuma modificação no código dos testes. Para distribui-los para a nuvem foi necessário apenas a execução do método *CloudTest.runnerDistributingMethods* em uma classe a parte especificando o nome do projeto, um *array* de *String* contendo o nome do pacote e a classe a ser testada, o arquivo de permissão de acesso ao servidor e o diretório local do teste.

4.4 Considerações Finais

Neste capítulo foi apresentado o processo de modelagem e concepção do *framework CloudTesting*. Após a definição, foram discutidos os requisitos funcionais e não funcionais do

framework. Em seguida foi discutido o processo de identificação de entidades, a estrutura de classes e pacotes. Por último, foi mostrado como utilizar do *framework*.

O próximo capítulo (Capítulo 5) apresenta experimentos realizados com o *CloudTesting* e a Amazon EC2, sendo avaliados os resultados obtidos em três tipos de instâncias *Micro*, *Small* e *Medium*.

Capítulo 5

Experimentos e Resultados

Este capítulo apresenta os testes efetuados para avaliação e validação do *framework* de teste de unidade na nuvem *CloudTesting*. Todo o processo de teste na nuvem foi realizado na Amazon Elastic Computer (EC2). Inicialmente, foram executados conjuntos de testes de unidade em uma máquina local (Seção 5.2) com o intuito de comparar os resultados obtidos com os resultados alcançados pelos testes de unidade efetuados na nuvem. Em seguida, foram configuradas instâncias EC2 *Micro*, *Small* e *Medium*, (seções 5.2.1, 5.2.3 e 5.2.4 consecutivamente) tendo como principal objetivo verificar o *speedup* e a eficiência do algoritmo paralelo do *framework* em cada tipo de instância testada. Por último, realizou-se uma comparação entre valores e tempos de execução entre todos os tipos de testes efetuados (seção 5.3).

5.1 Metodologia

Para realizar os experimentos, desenvolveu-se um conjunto de testes que possuem tempo de processamento médio previamente conhecido quando executado em uma máquina local, objetivando efetuar uma comparação com os resultados obtidos através do uso do *framework CloudTesting.* O conjunto de teste adotado para os exames possuem 1800 testes de unidade*,* onde cada teste de unidade efetua o cálculo da proporção numérica entre a relação das grandezas do perímetro e o diâmetro de uma circunferência com 5000000 amostras com pontos dentro do círculo definidos de modo aleatório para o eixo X e eixo Y. Entretanto, a escolha desse conjunto de testes é apenas um exemplo de tarefa que consome tempo de processamento, não existe motivo específico para tal implementação. Conjunto de testes mais simples poderiam ter sido implementados, como por exemplo, a soma de números inteiros.

Basicamente os experimentos se dividem em 2 cenários: (01) *stand alone* onde o conjunto de teste é executado 45 vezes em uma máquina local; (02) onde o conjunto de teste é executado 45 vezes na nuvem utilizando 18 máquinas.

A análise dos resultados engloba uma série de sub-tarefas. Inicialmente utiliza-se o Critério de Chauvenet [Pop, Ciascai e Pitica 2010] buscando erradicar anomalias nos resultados. Em seguida, calcula-se a média do tempo de execução, o desvio padrão, o coeficiente de variação e identifica-se o

melhor e pior tempo de execução. Essas informações permitem o cálculo do *speedup* e da eficiência do algoritmo paralelo (Equação 1) [Cortez e Saavedramendez 1999]. Após isso, apresenta-se o tempo de execução baseado em intervalos de confiança de 95% e 99%) (Equação 2 e Equação 3 respectivamente) [Dillard 1997].

Equação 1 – Fórmula para cálculo da eficiência do algoritmo paralelo.

$$Ef = \frac{Sp}{Np}$$

Onde:

- *Sp* é o *speedup* alcançado e;
- *Np* é o número de processadores utilizados para executar o programa em paralelo.

Equação 2 – Fórmula para cálculo de Intervalo de confiança α= 95%.

$$X = \mp 1{,}96 \times \frac{S}{\sqrt{n}}$$

Onde:

- 1,96 é o valor utilizado para calcular o intervalo de confiança de 95%;
- *S* é o desvio padrão e;
- *n* é a média populacional

Equação 3 – Fórmula para cálculo de Intervalo de confiança α= 99%.

$$X = \mp 2{,}58 \times \frac{S}{\sqrt{n}}$$

Onde:

- 2,58 é o valor utilizado para calcular o intervalo de confiança de 99%;
- *S* é o desvio padrão e;
- *n* é a média populacional

5.2 Cenário 01 - Testes Locais

Os testes executados localmente seguiram uma política rígida em relação ao uso do equipamento durante o período de testes. Visando evitar resultados anômalos e obter resultados reais, utilizou-se uma máquina dedicada para executar os testes. Após o término de cada teste efetuava-se um *reboot* na máquina com o intuito de higienizar dados armazenados na memória RAM e cache do processador. A configuração da máquina possuía processador Core 2 Duo 2.20 GHz, 4 Gigabytes de RAM e Sistema Operacional Linux 32 bits.

Para capturar o tempo de execução real do conjunto de teste, utilizou-se o ambiente de desenvolvimento integrado Eclipse juntamente com o *plugin* JUnit PDE que possui um perfilador e gerencia por padrão todos os testes de unidade de *software* executados com o JUnit na plataforma. A Figura 16 apresenta um trecho do conjunto de testes utilizado para realizar a experiência.

Figura 16 – Trecho do conjunto de testes utilizado para realizar o teste.

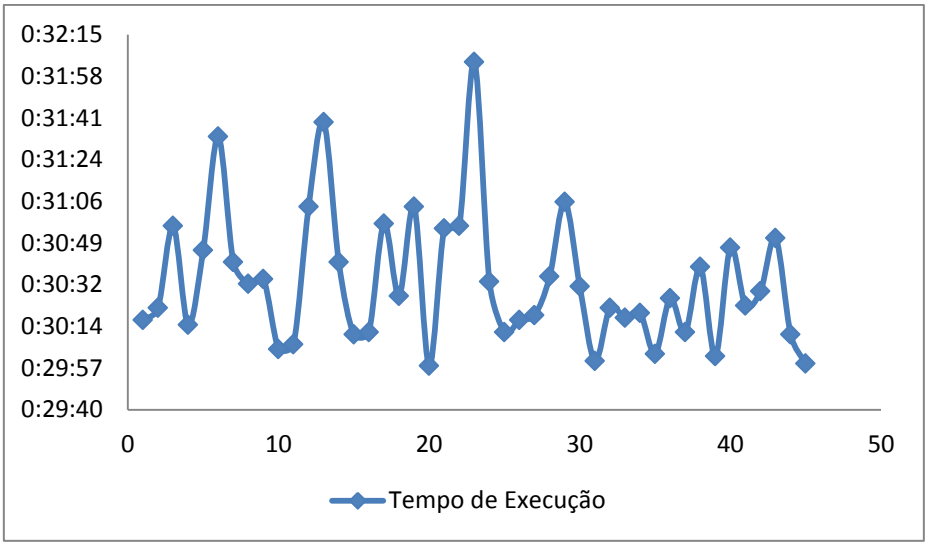
```
@Test
public void testAdd0()  {
        long amostras = 5000000;
        long pontosDentroDoCirculo = 0;

        for (int i = 0; i < amostras; i++) {
                double px = 2 * Math.random() - 1;
                double py = 2 * Math.random() - 1;

                if (Math.pow(px, 2) + Math.pow(py, 2) <= 1) {
                        pontosDentroDoCirculo++;
                }
        }
        assertEquals(314, Math.round(((4 * (double) pontosDentroDoCirculo / (double)amostras) * 100)/ 1d));
}
```

Cada teste de unidade obteve um tempo de execução médio próximo de 1 segundo. O conjunto foi composto por 1800 testes, assim um único conjunto de testes gastou em média 30 minutos para completar seu processamento. A execução total do conjunto em 45 vezes resultou em 22:54:51. O tempo médio de execução foi de 00:30:33 por conjunto, desvio padrão de 00:00:27, coeficiente de variação de 1,47%, melhor e pior tempo de execução 00:29:58 e 00:31:09 consecutivamente, além de intervalo de confiança (00:30:25 ± 00:30:41), α = 95% e (00:30:22 ± 00:30:44), α = 99%. Esses dados servirão de base para análise de *speedup* da execução na nuvem. A Figura 17 apresenta os resultados alcançados nesse primeiro experimento. O detalhamento dos resultados dos testes pode ser visto no Apêndice A.

Figura 17 – Resultados dos experimentos locais.



5.2.1 Cenário 02 - Testes na nuvem

Para dar início a essa etapa, tornou-se necessário definir o provedor de serviço de nuvem a ser empregado. A Amazon periodicamente abre chamadas de trabalhos e pesquisas científicas, objetivando ceder recursos e serviços de nuvem para os trabalhos que são julgados pertinentes. O projeto *CloudTesting* recebeu da Amazon um auxílio no valor de US$ 7.500,00 para ser utilizado em seus serviços de computação na nuvem, por esse motivo utilizou-se a EC2 como provedora de nuvem.

O teste na nuvem confrontou-se com características encontradas no contexto de sistemas distribuídos: latência e flutuação de rede. Por esse motivo, foi estabelecida uma política de escala de horário para a execução dos testes, com o intuito de obter, em tese, as mesmas condições de largura de banda para todos os experimentos. Assim, todos os testes foram realizados em uma faixa de tempo que varia entre 00:00 e 04:00 horas, comumente a fatia de tempo selecionada reflete o horário de menor utilização da rede no laboratório, conforme pode ser constatado no Anexo A. Devido à quantidade de testes, os experimentos não foram executados em um único dia, ao invés disso, foram executados em dias alternados seguindo a política supramencionada.

Todos os 45 exames foram realizados na Amazon nos tipos de instâncias: *Micro*, *Small* e *Medium instance*. Para capturar o tempo de execução real do conjunto de teste, utilizou-se o ambiente de desenvolvimento integrado Eclipse juntamente com o *plugin* TPTP que possui um perfilador. Adotou-se uma largura de banda nominal de 15mbps para *download* e 1mbps para *upload*. Algumas configurações foram feitas nas máquinas da nuvem de modo prévio para a execução do

framework CloudTesting:

(1) Os diretórios log e lib foram criados para armazenamento de logs e bibliotecas respectivamente;

(2) As bibliotecas dos *Frameworks JUnit* e *CloudTesting* foram distribuídas para o diretório lib;

(3) O serviço do *CloudTesting* foi ativado através do comando: *java -XX:PermSize=128m -XX:MaxPermSize=256m -classpath lib/CloudTesting.jar:. CloudTesting.connection.Server.*

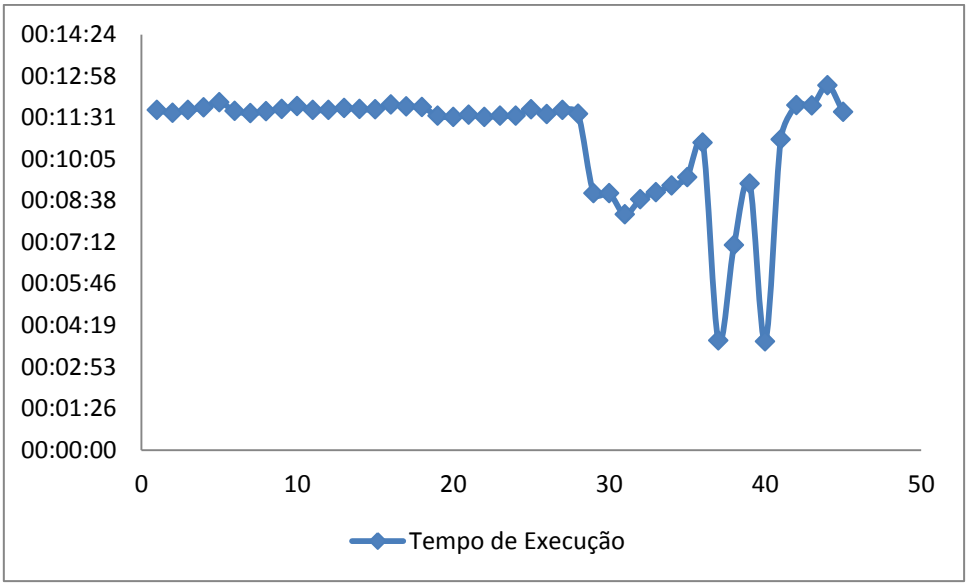
5.2.2 *Micro Instance*

Micro instances possuem recursos limitados de CPU, embora permitam o aumento da capacidade computacional quando ciclos adicionais estão disponíveis. São compostas por Sistema Operacional Linux 32 bits com até 2 unidades computacionais EC2 (para curtos períodos de pico), 613MB de memória RAM, apenas armazenamento EBS e baixo desempenho para entrada e saída de dados (Amazon, 2012).

Como resultado, obtivemos o tempo total de execução do conjunto de testes 08:04:53, diminuindo em 14:49:58 o tempo total de execução do cenário 01. O tempo médio de execução foi de 00:10:46 com desvio padrão 00:01:59, coeficiente de variação de 18,23%, melhor e pior tempo de execução 00:03:56 e 00:12:03 consecutivamente. Analisando o valor do desvio padrão observa-se uma variação considerável (conforme ilustra a Figura 18) resultando diretamente no intervalo de confiança (00:10:13 ± 00:11:21), $\alpha = 95\%$ e (00:10:02 ± 00:11:32), $\alpha = 99\%$.

Em relação a grande variação entre o tempo de execução de alguns experimentos, utilizou-se o Critério de Chauvenet para verificar se os mesmos poderiam ser considerados como anomalias. Contudo, nenhum resultado se enquadrou nessa situação. O que leva a inferir que essa diferença se constituiu na condição de que as *Micro instances* não são capazes de manter uma alta taxa de desempenho, embora, podem ultrapassar por um curto espaço de tempo até 2 ECUs, ou seja, o dobro do número de ECUs fornecidas a uma instância do tipo *Small*. Outra hipótese se relaciona com recursos de rede e subsistema de discos compartilhados, possivelmente quando um recurso é subutilizado todas as instâncias virtuais ativas recebem a oportunidade de consumir uma fatia extra desse recurso enquanto o mesmo permanecer disponível [Amazon 2012].

Figura 18 – Resultado dos experimentos na Amazon EC2 – *Micro Instance*.



5.2.3 *Small Instance*

Instâncias do tipo *Small* fazem parte do grupo de instancias padrão, são recomendadas para a execução da maior parte de aplicativos. Para os testes, as instâncias possuíam Sistema Operacional Linux 32 bits, 1 unidade computacional EC2; 1,7GB de memória RAM e 160GB disponível para armazenamento de dados.

O tempo de execução total do conjunto de testes foi de 04:07:31, diminuindo em 18:47:20 o tempo total de execução do cenário 01 e em 03:57:22 em comparação com a execução nas *Micro instances*. O tempo de execução médio foi de 00:05:29, desvio padrão 00:00:08, coeficiente de variação de 2,44% melhor e pior tempo de execução 00:05:19 e 00:05:52 respectivamente. O tempo de execução individual de cada experimento esteve consistente, como pode ser observado com o desvio padrão estreito com intervalo de confiança (00:05:27 ± 00:05:31), $\alpha = 95\%$ e (00:05:26 ± 00:05:32), $\alpha = 99\%$.

5.2.4 *Medium Instance*

Instâncias do tipo Medium também fazem parte do grupo padrão, possuem a mesma finalidade das instâncias do tipo Small, ou seja, executar a maior parte de aplicativos. Durante os experimentos, utilizou-se a seguinte configuração: CPU de alto desempenho, 3.75 GB de memória, 2

Unidades de processamento EC1 (1 núcleo virtual com 2 Unidades de processamento EC2 cada), 410 GB de armazenamento de dados e plataforma de 32 bits.

O tempo de execução total do conjunto de testes foi de 02:50:48, uma redução de 20:04:03 se comparado ao cenário 01, 05:14:05 se comparado a execução nas Micro instances e 01:16:43 se comparada a execução nas *Small instances*.

Também obtivemos como resultado, o tempo médio de 00:03:47, desvio padrão de 00:00:09, coeficiente de variação de 3,96%, melhor e pior tempo de execução 00:03:12 e 00:04:01 respectivamente, intervalo de confiança (00:03:44 ± 00:03:50), α= 95% e $(00:03:43 \pm 00:03:51)$ α= 99%.

5.3 Análise de *Speedup*

Alguns pontos devem ser considerados antes de avaliarmos um determinado *speedup* de modo positivo ou negativo. Ao idealizarmos o uso de 18 máquinas para efetuar processamento paralelo na nuvem, desejamos um *speedup* linear bastante aproximado ao número de máquinas empregadas.

Contudo, deve ser ponderado que para efetuarmos processamento paralelo na nuvem devemos realizar *upload* do código a ser testado para a mesma. Dependendo do tamanho do projeto, distribuir pacotes pela rede pode ser custoso. Outro fator se relaciona diretamente com o poder de processamento das máquinas e a capacidade de entrada e saída de dados na nuvem. Por último, ainda existe a relação com as instâncias de servidor virtualizadas, ou seja, o desempenho muitas vezes irá depender da quantidade de recursos disponíveis no servidor da nuvem.

Nos experimentos realizados observou-se que *Micro instances* apesar de terem alcançado um *speedup* considerável no seu melhor tempo de execução, não são adequadas para uma grande carga de requisições em um curto espaço de tempo. Para esse contexto se adequam melhor instâncias do tipo *Medium*.

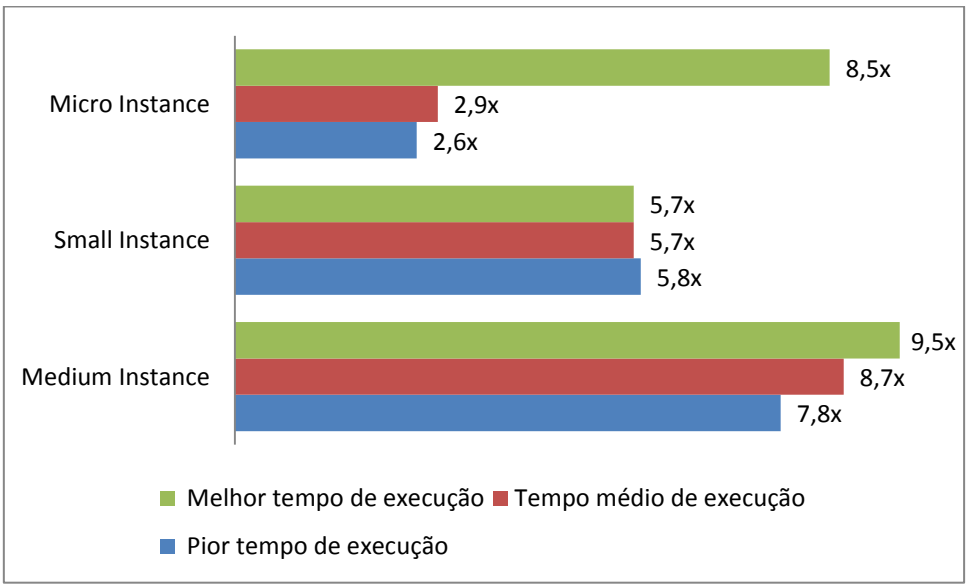
Os experimentos realizados nas *Micro instances* alcançaram um *speedup* de 8.55 e 0.48 de eficiência do algoritmo paralelo para o melhor tempo de execução, *speedup* de 2.89 e 0.16 de eficiência do algoritmo paralelo para o tempo médio e *speedup* 2.61 e 0.14 de eficiência do algoritmo paralelo para o pior tempo de execução em relação aos experimentos locais (cenário 01). Os testes executados nas *Small instances* alcançaram um *speedup* de 5.70 e 0.32 de eficiência do algoritmo paralelo para o melhor tempo de execução, *speedup* de 5.71 e 0.32 de eficiência do algoritmo paralelo para o tempo médio e *speedup* de 5.70 e 0.32 de eficiência do algoritmo paralelo para o pior

tempo de execução em relação aos experimentos locais. Os experimentos realizados em *Medium instances* alcançaram um *speedup* de 9.48 e 0.53 de eficiência do algoritmo paralelo para o melhor tempo de execução, *speedup* de 8.72 e 0.48 de eficiência do algoritmo paralelo para o tempo médio e *speedup* de 7.83 e 0.43 de eficiência do algoritmo paralelo para o pior tempo de execução em relação aos experimentos locais. A Figura 19 apresenta uma comparação de *speedup* entre os tipos de instâncias.

O melhor tempo de execução dos experimentos realizados em *Micro instances* resultou em um *speedup* superior ao resultado do melhor tempo de execução dos testes executados nas *Small instances*. Esse resultado é surpreendente a princípio, embora, como mencionado anteriormente máquinas do tipo *Micro* podem momentaneamente utilizar até 2 ECUs, obtendo o dobro do poder computacional de um máquina *Small*. Apesar disso, o tempo médio e o pior tempo de execução dos experimentos executados em *Micro instances* são bastante inferiores aos resultados obtidos nas *Small instances*. Os melhores tempos foram obtidos com *Medium instances*, esse resultado já era esperado devido às configurações de *hardware* e pela taxa de entrada e saída de dados superior as outras instâncias testadas.

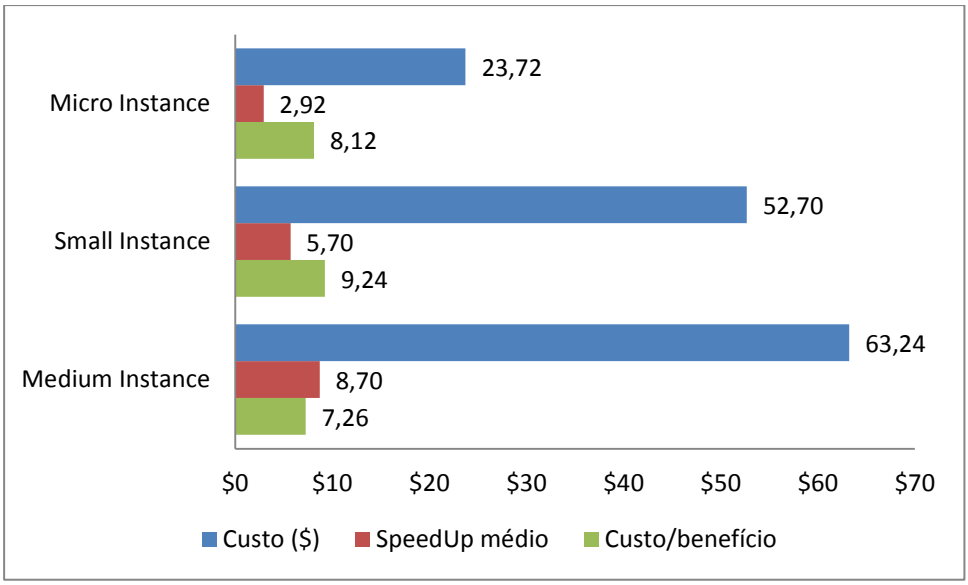
Para realizarmos esses experimentos tivemos o custo de apenas U$23.72 para executar as 18 máquinas *Micro*, U$52.70 para as 18 máquinas do tipo *Small* e U$63.24 para as 18 máquinas do tipo *Medium*. Efetuando o cálculo da razão entre o valor do *speedup* médio e o custo gasto para a realização dos experimentos, obtemos o custo/benefício de cada tipo de instância. Nesse caso, quanto menor for o valor resultante melhor, visto que, isso implica em um gasto menor para obter o mesmo benefício.

Figura 19 – Comparação do *speedup* entre as instâncias *Micro*, *Small* e *Medium*.



Conforme apresenta a Figura 20, as *Micro* instâncias alcançaram um custo/benefício de 8.12, instâncias *Small* obtiveram um custo/benefício de 9.24 e máquinas *Medium* resultaram em um custo/benefício de 7.26.

Figura 20 – Custo benefício entre as instâncias.



Como citado anteriormente, instâncias *Micro* não são capazes de manter uma alta taxa de desempenho, embora, ultrapassem por um curto espaço de tempo até 2. Esse fato refletiu diretamente no valor do custo/benefício em relação à instância *Small*. Apesar dos experimentos realizados em instâncias Micro resultarem em um *speedup* médio inferior aos mesmos experimentos realizados em instâncias do tipo *Small*, conclui-se que é mais vantajoso utilizar instâncias Micro para essa situação. Contudo, apesar das instâncias *Medium* terem alcançado o maior custo, as mesmas obtiveram o melhor valor relacionado a custo/benefício, além do melhor *speedup* médio.

Capítulo 6

Conclusão

Este capítulo apresenta as conclusões da dissertação. Inicialmente, apresenta-se uma discussão geral sobre a pesquisa. Em seguida, comentam-se sobre as dificuldades encontradas, trabalhos futuros e contribuições.

6.1 Considerações Finais

Através dos resultados obtidos após a execução dos experimentos, pode-se inferir que adotar a nuvem para efetuar testes automáticos de *software* é uma tendência e uma alternativa viável. Todavia, realizar esse procedimento requer o desenvolvimento de aplicações que sejam capazes de distribuir e gerenciar os testes de *software*. Elaborar essa estrutura requer esforços para entender e definir o escopo. Essa problemática, foi uma das dificuldades encontradas na elaboração do *CloudTesting.* Além disso, fatores relacionados à compactação dos dados para transferência e gerenciamento de diversas requisições paralelas também se enquadram como problemas identificados. Objetivando não só executar testes automáticos de *softwares* na nuvem, o *software* proposto nesse trabalho também foi modelado para servir de um *framework*, permitindo o reuso e a adaptação do código em outras infraestruturas de execução distribuídas.

Como contribuição o estudo forneceu um modo de automatizar e paralelizar execução de testes na nuvem, embasando a concepção do desenvolvimento de *software* baseado em testes na nuvem de um modo mais rápido, como também permitiu analisar o *speedup* e calcular o custo envolvido ao executar teste de unidade na nuvem, servindo de base para ilustrar parâmetros reais para empresas e desenvolvedores que desejam avaliar o processamento paralelo na nuvem, visto que, com a mensuração do *speedup*, se constitui os valores relacionados a ganhos de desempenho. Ao aumentar o *speedup* dos testes, o *CloudTesting* torna possível a execução de testes que antes não eram executados devido ao seu tempo de execução. Assim, reduzindo o tempo de desenvolvimento.

Se comparado com os trabalhos correlatos apresentados no Quadro 3.4 o *CloudTesting* além de simplificar o método de efetivação de testes automáticos em ambientes distribuídos, indicou ganhos significativos no tempo de execução dos testes sem aumentar proporcionalmente os custos

referentes a montagem da infraestrutura de execução, facilitando assim o processo de utilização de plataformas de computação em nuvem como ambientes para a execução de testes automáticos de *software.*

Em relação a trabalhos futuros, a pesquisa pode ser expandida em várias subáreas. Das quais se destacam:

1. Realização de um estudo estatístico relacionado aos resultados obtidos, com a possível realização da execução de uma quantidade maior de novos experimentos.
2. Implementar otimização do algoritmo utilizado para efetuar o balanceamento das instâncias;
3. Implementar otimização no processo de compactação de dados para efetuar a transferência pela rede;
4. Implementar otimização no algoritmo que efetua as requisições para as instâncias da nuvem;
5. Validar o framework em diversas infraestruturas paralelas de execução;
6. Implementar *plug-ins* para o framework que se unam aos ambientes de desenvolvimento integrados, fornecendo um ambiente mais amigável para o usuário.
7. Implementar uma varredura exploratória de diferentes ambientes e configurações

Assim, considera-se que a proposta de *framework* da dissertação obteve avanços no processo de teste automático de *software* na nuvem. Conforme pode ser visto na Tabela 6, o *framework CloudTesting* agrega mais funcionalidade que as outros sistemas analisadas, pois ela é a única a suportar a execução de suítes de testes JUnit juntamente ao processamento paralelo, além de possibilitar a execução dos testes em ambientes variados, fornecer integração com os ambientes de desenvolvimento via a distribuição de bibliotecas, possibilitar o uso dos recursos de virtualização de hardware, redimensionamento automático de recursos de hardware virtualizado vantagens e modelo de negócio flexível oferecido pela nuvem.

O *CloudTesting* é um *framework open source* que pode ser adquirida pelo link: http://code.google.com/p/cloudtestingdi/ e disponibilizada pela licença GNU General Public License 3 (GPLv3). Assim, o *framework* pode ser distribuída ou modificada de acordo com os termos da licença.

Referências

[Banzai *et al.* 2010] BANZAI, Takayuki; KOIZUMI, Hitoshi; KANBAYASHI, Ryo; IMADA, Takayuki; HANAWA, Toshihiro; SATO, Mitsuhisa. *D-Cloud*: Design of a *Software* Testing Environment for Reliable Distributed Systems Using Cloud Computing Technology. *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* , vol., no., pp.631-636, 17-20 May 2010.

[Barbosa 2009] BARBOSA, Guilherme Mauro Germoglio. Um Livro-texto para o Ensino de Projeto de Arquitetura de *Software*. Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande, 2009.

[Beizer 1995] BEIZER, B. Black box testing: techniques for functional testing of *Software* and systems. John Willey, 1995.

[Ben-Ari 1999] BEN-ARI, M. The Bug that Destroyed a Rocket. Journal of Computer Science Education, v. 13, n. 2, p. 15—16, 1999

[Binder 2000] BINDER, R. Testing Object-oriented Systems: Models, Patterns, and Tools. Addison-Wesley Professional, 2000.

[Buyya *et al.* 2009] BUYYA, R; YEO, C.S; VENUGOPAL, S; BROBERG, J; BRANDIC, I. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems, 25 (6), pp. 599-616, 2009.

[Catelani *et al.* 2008] CATELANI, M.; CIANI, L.; SCARANO, V.L.; BACIOCCOLA, A. A Novel Approach To Automated Testing To Increase *Software* Reliability. *Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE* , vol., no., pp.1499-1502, 12-15 May 2008.

[Crespo *et al*. 2004] CRESPO, A. N.; SILVA, O. J.; BORGES, C. A.; SALVIANO, C. F.; ARGOLLO, M.; JINO, M. Uma metodologia para teste de *Software* no Contexto da Melhoria de Processo, In: III Simpósio Brasileiro de Qualidade de *Software* (SBQS 2004), Brasília 2004.

[Coulouris *et al*. 2007] COULOURIS, George; KINDBERG, TIM; Dollimore, JEANCOU. Sistemas Distribuídos: Conceitos e Projeto. 4ª Ed. Editora: Bookman, 2007.

[Duarte *et al*. 2006] DUARTE, A. *et al*.. Multi-environment *Software* Testing on the Grid. In: PADTAD '06: Proceedings of the 2006 workshop on Parallel and distributed systems: testing and debugging. New York, NY, USA: ACM, 2006. p. 61–68. ISBN 1-59593-414-6.

[Duarte *et al*. 2010] DUARTE, A. Uma Abordagem Baseada em Testes Automáticos de *Software* para Diagnóstico de Faltas em Grades Computacionais. Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, Coordenação de Pós-Graduação em Ciência da Computação. Maio, 2010.

[Fewster e Graham 1999] FEWSTER, M; GRAHAM, D. *Software* Test Automation Technology and Example. BeiJing Electronic Industry Press, 1999:3332342.

[Fuggeta 2000] FUGGETA, A. *Software* Process: a Roadmap. In: 22nd International Conference on the Future of *Software* engineering. Limerick, Ireland: ACM, 2000, p.25-34.

[Gamma e Back 1999] GAMMA, E.; BECK, K. Junit: A cook's tour. Java Report, v. 5, n. 4, p. 27–38, 1999

[Gimenes *et al*. 2005] GIMENES, I. M. DE S.; HUZITA, E. H. M. Desenvolvimento Baseado em Componentes: conceitos básicos e técnicas. Rio de Janeiro: Editora Ciência Moderna. 2005.

[Gong *et al*. 2010] GONG, Chunye; LIU, Jie; ZHANG, Qiang; CHEN, Haitao; GONG, Zhenghu. The Characteristics of Cloud Computing. *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on* , vol., no., pp.275-279, 13-16 Sept. 2010.

[Gupta e Jalote 2007] GUPTA, A.; JALOTE, P. Test Inspected Unit or Inspect Unit Tested Code?. *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on* , vol., no., pp.51-60, 20-21 Sept. 2007.

[Rothermel *et al*. 2001] ROTHERMEL, R. H. Untch and C. Chu. Prioritizing test cases for regression testing, IEEE Transactions on *Software* Engineering, 27(10):929-948, 2001.

[Harrold 2000] HARROLD, M. J., Testing: A Roadmap, In 22 International Conference on *Software* Engineering – Future of SE Track, 61-72, June 2000.

[Hughes *et al.* 2004] HUGHES, D.; GREENWOOD, P.; COULSON, G. A Framework for Testing Distributed Systems. in: Proceedings of the 4th IEEE International Conference on Peer-to-Peer computing (P2P'04), 2004.

[Jeffries *et al*. 2000] JEFFRIES, R. E.; ANDERSON, A.; HENDRICK-SON, C. Extreme Programming Installed. Addison-Wesley, 2000.

[Juristo *et al*. 2004] JURISTO, N., MORENO, A. M., Vegas, S. (2004) "Reviewing 25 years of testing technique experiments". Empirical *Software* Engineering: An International Journal, 9, p.7-44, Mar.

[Kapfhammer 2001] KAPFHAMMER, G. M. Automatically and Transparently Distributing the Execution of Regression Test Suites, in: Proceedings of the 18th International Conference on Testing Computer *Software*, 2001.

[Mazer e Loring 2008] MAZER, A.S.; LORING, S.M. Automated Testing of Science Instrument Flight *Software*. *Aerospace Conference, 2008 IEEE* , vol., no., pp.1-12, 1-8 March 2008.

[Mell e Grace 2009] MELL, P.; GRANCE, T. Draft NIST Working Definition of Cloud Computing v14, Nat. Inst. Standards Technol., 2009. Disponível em: <http://csrc.nist.gov/groups/SNS/cloud-computing/index.html.>

[Mellor 1994] MELLOR, P. CAD: Computer-Aided Disaster. Highly Integrated Systems, v. 1, n. 2, p. 101–156, 1994.

[McGregor e Sykes 2001] MCGREGOR, J.D., SYKES, D.A., A Practical Guide to Testing Object-Oriented *Software*, Addison-Wesley, 2001.

[Myers 2004] MYERS, G. J. *et al*.. The Art of *Software* Testing.Wiley, 2004.

[Nurmi *et al*. 2008] NURMI, D.; WOLSKI, R.; GRZEGORCZYK, C.; OBERTELLI, S. G.;SOMAN, YOUSEFF, L; ZAGORODNOV, D. "Eucalyptus opensource cloud-computing system" In CCA08: Cloud Computing and Its Applications, 2008.

[Pressman 2006] PRESSMAN, R. S. Engenharia de *Software*. 6ª. ed. São Paulo: MacGrow-Hill, 2006.

[Taipale *et al.* 2006] TAIPALE, O.; SMOLANDER, K.; KALVIAINEN, H. A survey on *Software* testing. Proc. of the 6th International SPICE Conference on *Software* Process Improvement and Capability dEtermination (SPICE '06), Luxembourg, May 2006.

[Offutt e Irvine 1995] OFFUTT, A. J., IRVINE, A., Testing Object-Oriented *Software* Using The Category Partition Method, In 17th International Conference on Technology of Object-Oriented Languages and Systems, Santa Barbara, CA, Prentice-Hall, pp 293-304, August 1995.

[Oriol e Ullah 2010] ORIOL, M.; ULLAH, F. YETI on the Cloud. *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on* , vol., no., pp.434-437, 6-10 April 2010.

[Pádua 2003] PÁDUA, F. W. Engenharia de *Software*: Fundamentos, Métodos e Padrões. 2ª. ed. Rio de Janeiro: LTC-Livros Técnicos e Científicos Editora S.A., 2003.

[Patton 2002] PATTON, R. "*Software* Testing", BeiJing Machine Press, 2002:1250.

[Shahamiri *et al*. 2009] SHAHAMIRI, S.R.; KADIR, W.M.N.W.; MOHD-HASHIM, S.Z. A Comparative Study on Automated *Software* Test Oracle Methods. *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on* , vol., no., pp.140-145, 20-25 Sept. 2009.

[The Open Group 2003] THE OPEN GROUP. TETware, http://tetworks.opengroup.org, 2005.

[Vaquero *et al.* 2009] VAQUERO, L.M.; MERINO, L.R.; CACERES, J.; Lindner, M. A break in the clouds: towards a cloud definition. ACM SIGCOMM Computer Communication Review, v.39 n.1, 2009.

[Whittaker 2000] WHITTAKER, J.A. "What Is *Software* Testing? And Why Is It So Hard?", IEEE *Software*, January/February, 2000.

[Wu 1999]. WU, J. (1999). Distributed System Design, CRC Press.

[Wu e Sun 2010]. WU, Xiaojun; SUN, Jinhua. The Study on an Intelligent General-Purpose Automated *Software* Testing Suite. *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on* , vol.3, no., pp.993-996, 11-12 May 2010.

[Xiaohui *et al*. 2010] XIAOHUI, Liu; YUQING, Lan; LIKE, Ma. Design and Implementation of Automated Testing Framework for Linux *Software* GUI Testing. Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on , vol., no., pp.1-4, 23-25 Sept. 2010.

[Zhang *et al.* 2010] ZHANG, Shufen *et al*.. Analysis and Research of Cloud Computing System Instance. Future Networks, 2010. ICFN '10. Second International Conference on , vol., no., pp.88-92, 22-24 Jan. 2010.

[Grundy *et al*. 2012] GRUNDY, J. and KAEFER, G. and KEONG, J. and LIU, A. (2012), "Guest Editors' Introduction: *Software* Engineering for the Cloud," *Software*, IEEE , vol.29, no.2, pp.26-29, March-April.

[Riungu-Kalliosaari *et al.* 2012] RIUNGU-KALLIOSAARI, L.; TAIPALE, O.; SMOLANDER, K.; , "Testing in the Cloud: Exploring the Practice," *Software*, IEEE , vol.29, no.2, pp.46-51, March-April 2012.

[Pop, Ciascai e Pitica 2010] POP, S.; CIASCAI, I.; PITICA, D.; , "Statistical analysis of experimental data obtained from the optical pendulum," Design and Technology in Electronic Packaging (SIITME), 2010 IEEE 16th International Symposium for , vol., no., pp.207-210, 23-26 Sept. 2010 doi: 10.1109/SIITME.2010.5653515

[Dillard 1997] DILLARD, G.M.; , "Confidence intervals for power estimates," Signals, Systems & Computers, 1997. Conference Record of the Thirty-First Asilomar Conference on , vol.1, no., pp.925-929 vol.1, 2-5 Nov. 1997doi: 10.1109/ACSSC.1997.680578

[Kalagiakos e Karampelas 2011] KALAGIAKOS, P.; KARAMPELAS, P.; , "Cloud Computing learning," Application of Information and Communication Technologies (AICT), 2011 5th International Conference on , vol., no., pp.1-4, 12-14 Oct. 20.

[Jadeja e Modi 2012] JADEJA, Y.; MODI, K.; , "Cloud computing - concepts, architecture and challenges," Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on , vol., no., pp.877-880, 21-22 March 2012 doi: 10.1109/ICCEET.2012.6203873

[Silva 2006] SILVA, R. P.; PRICE, R. T. O uso de técnicas de modelagem no projeto de frameworks orientados a objetos. International Conference of the Argentine Computer Science and Operational Research Society (26th JAIIO) / First Argentine Symposium on Object Orientation (ASOO'97). Buenos Aires, Argentina, 1997.

[Prota 2012] PROTA, T. M. MoonDo-Eclipse: Um Ambiente para Desenvolvimento de Aplicações Declarativas para o SBTVD. Dissertação de Mestrado. Universidade Federal de Pernambuco, Centro de Informática, Pós-Graduação em Ciência da Computação, 2012.

[Mian e Natali 2001] MIAN, P.; NATALI, A. Ambientes de Desenvolvimento de *Software* e Projeto ADS. n. 1, 2001.

[Mattsson e Bosch 1997] MATTSSON, M.; BOSCH, J.; , "Framework composition: problems, causes and solutions," Technology of Object-Oriented Languages and Systems, 1997. TOOLS 23. Proceedings , vol., no., pp.203-214, 28 Jul-1 Aug 1997 doi: 10.1109/TOOLS.1997.654724

[Feng Zhu e Wei-Tek Tsai 1998] FENG ZHU; WEI-TEK TSAI. "Framework-oriented analysis," Computer *Software* and Applications Conference, 1998. COMPSAC '98. Proceedings. The Twenty-Second Annual International , vol., no., pp.324-329, 19-21 Aug 1998 doi: 10.1109/CMPSAC.1998.716675

[Krajnc e Hericko 2003] KRAJNC, A.; HERICKO, M.; , "Classification of object-oriented frameworks," EUROCON 2003. Computer as a Tool. The IEEE Region 8 , vol.2, no., pp. 57- 61 vol.2, 22-24 Sept. 2003.

[Campbell *et al.* 1991] CAMPBELL, R.H.; ISLAM, N.; JOHNSON, R.; KOUGIOURIS, P.; MADANY, P.; , "Choices, frameworks and refinement," Object Orientation in Operating Systems, 1991. Proceedings., 1991 International Workshop on , vol., no., pp.9-15, 17-18 Oct 1991

[Ramabhadran e Pasquale 2003] RAMABHADRAN, S. and PASQUALE, J. (2003). "Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay," Proc. of SIGCOMM '03, Jan 2003.

[Amazon 2012] Amazon EC2 (2012). "Amazon Elastic Compute Cloud (Amazon EC2)", http://aws.amazon.com/pt/ec2/instance-types/, Junho.

[Robinson e Ragusa 2011] Robinson, P.; Ragusa, C.; , "Taxonomy and Requirements Rationalization for Infrastructure in Cloud-based Software Testing," *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on* , vol., no., pp.454-461, Nov. 29 2011-Dec. 1 2011.

[Riungu-Kalliosaari *et al.* 2012] RIUNGU, L.M.; TAIPALE, O.; SMOLANDER, K.; , "Software Testing as an Online Service: Observations from Practice," *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on* , vol., no., pp.418-423, 6-10 April 2010.

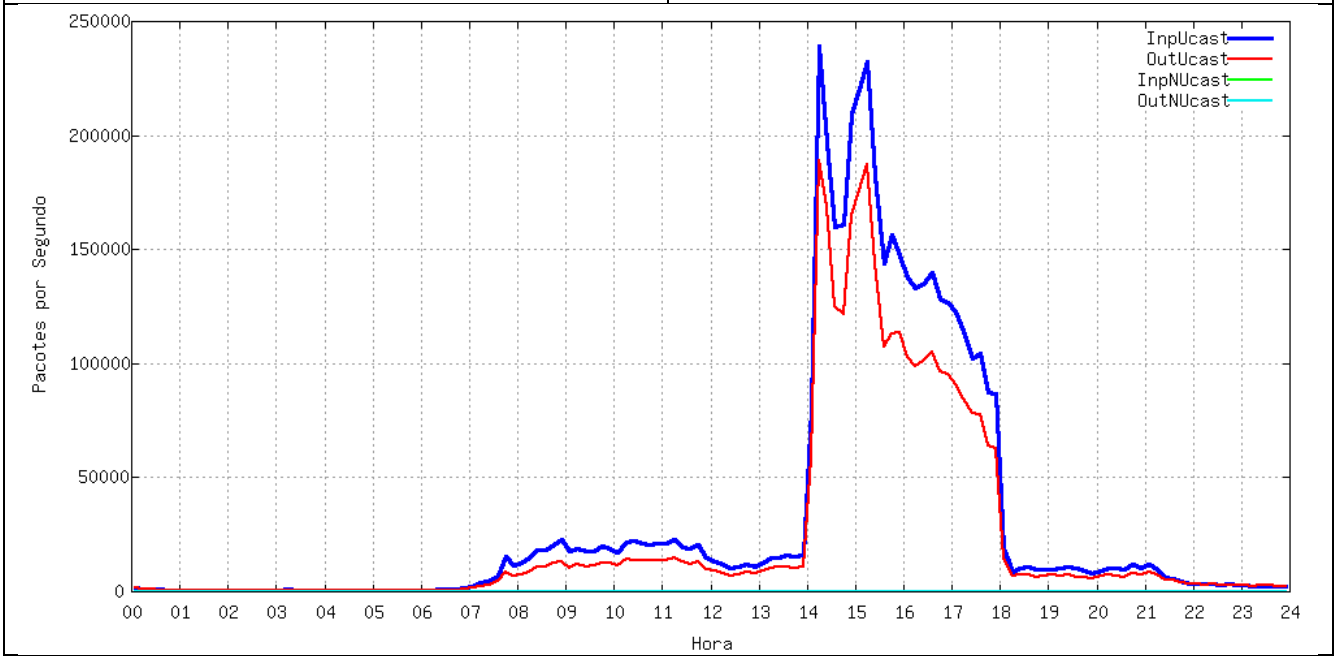
[Soasta 2012] The CloudTest Plataform. http://www.soasta.com/cloudtest/, 2012.
[Skytap 2012] SkyTap Cloud. http://www.skytap.com/skytap-cloud/, 2012.

Anexo A – Estatística de Interface WAN STM-3/2/0

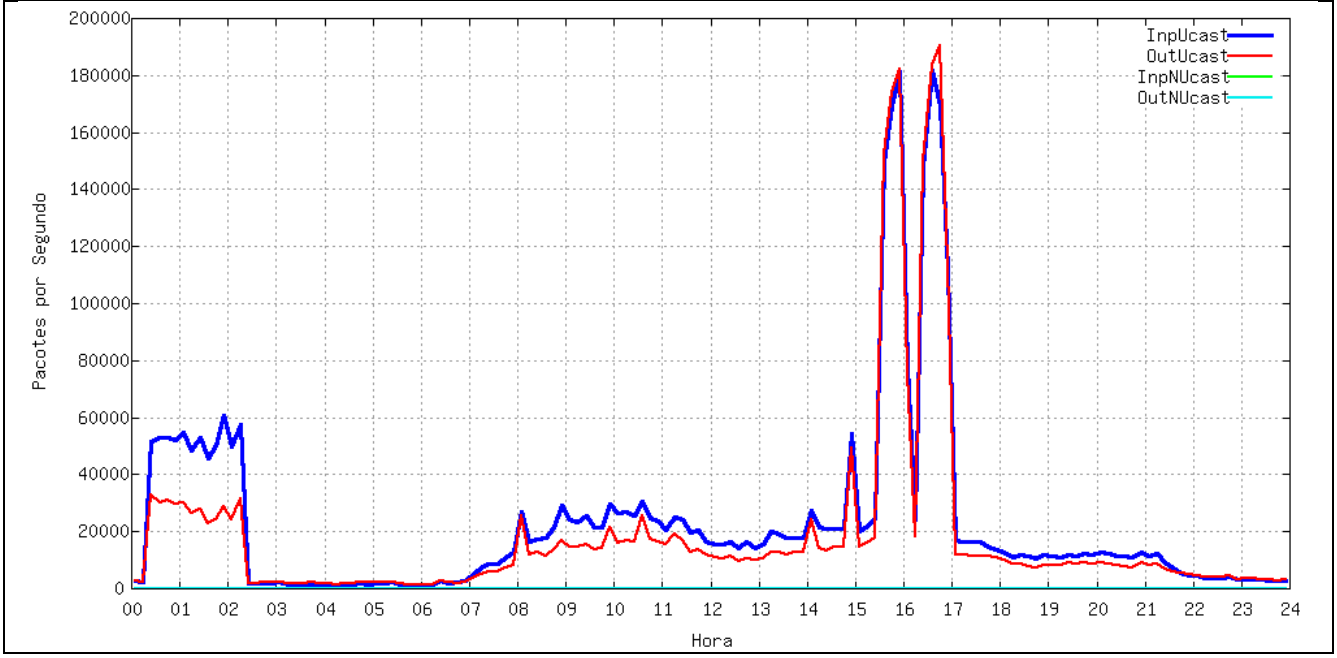
Pacotes de Entrada e Saída

Este anexo apresenta gráficos entre o período de 24/09/12 até 30/09/12 relacionados à estatística de tráfego de pacotes da Interface WAN STM-3/2/0 obtidos pelo WebManager (http://webmngr.pop-pb.rnp.br).

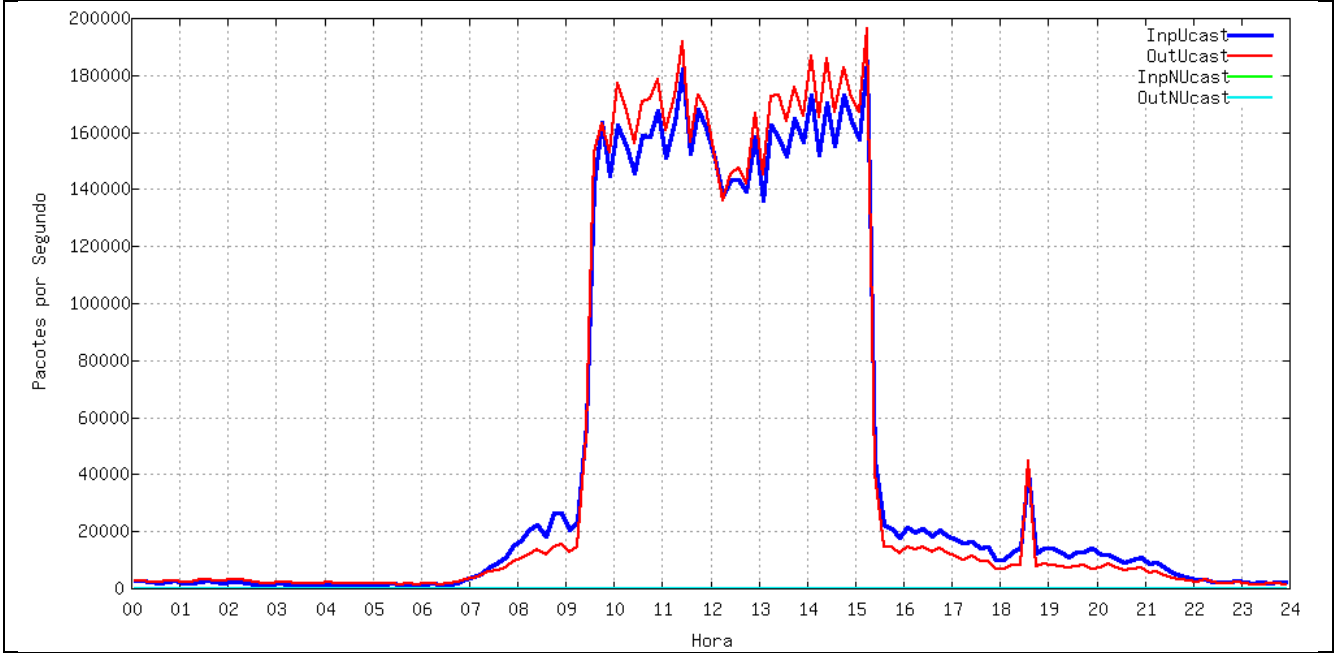
Data: **24/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



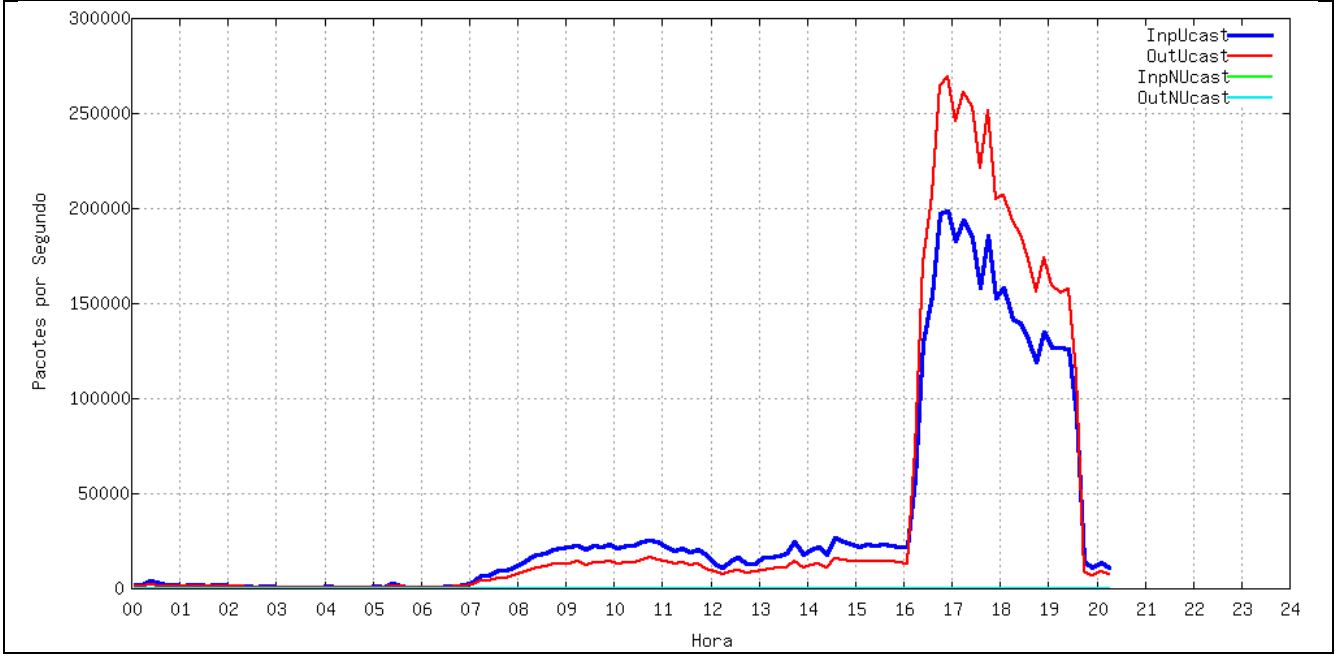
Data: **25/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



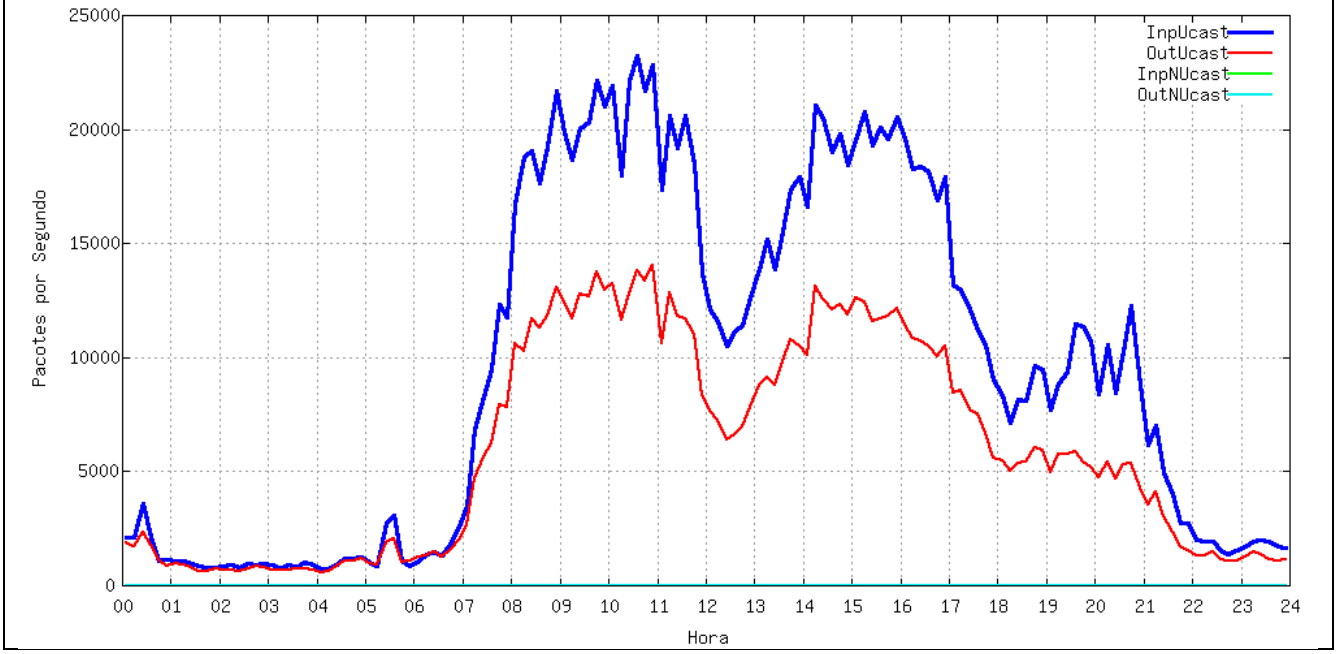
Data: **26/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



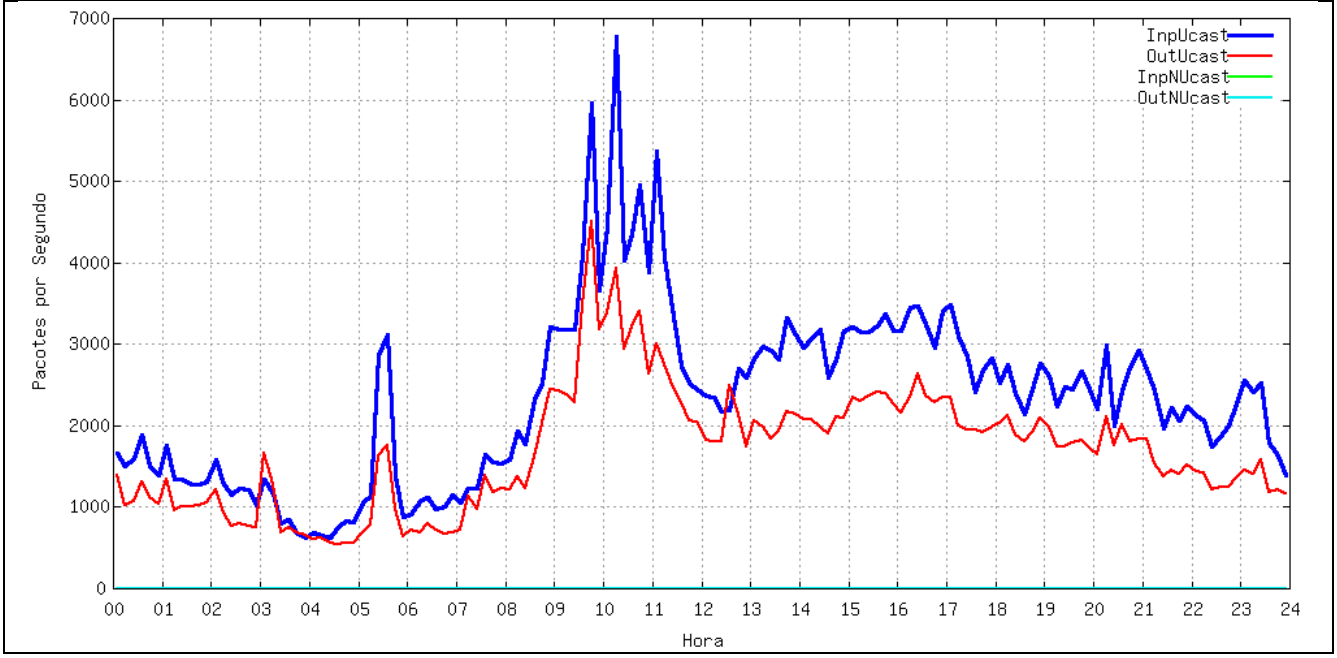
Data: **27/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



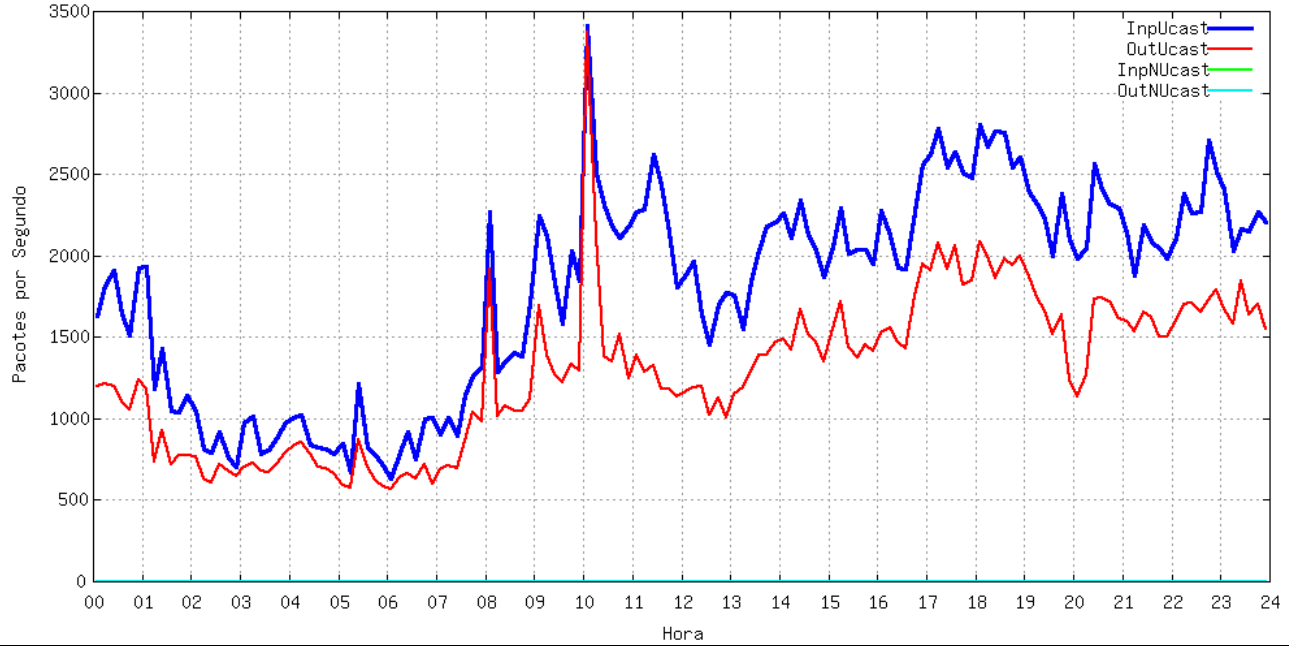
Data: **28/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



Data: **29/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



Data: **30/09/2012**	*Hardware*: **WAN STM-3/2/0 - Juniper mx480 - Joao Pessoa**



Apêndice A - Resultado dos experimentos realizados

Este apêndice apresenta o detalhamento dos resultados obtidos de todos os experimentos efetuados durante a pesquisa.

Testes Locais			
Data	**Quantidade de Testes de Unidade**	**Experimento**	**Tempo de Execução**
12/5/2011	1800	1	0:30:17
12/5/2011	1800	2	0:30:22
12/6/2011	1800	3	0:30:56
12/6/2011	1800	4	0:30:15
12/6/2011	1800	5	0:30:46
12/6/2011	1800	6	0:31:33
12/6/2011	1800	7	0:30:41
12/6/2011	1800	8	0:30:32
12/6/2011	1800	9	0:30:34
12/6/2011	1800	10	0:30:05
12/6/2011	1800	11	0:30:07
12/6/2011	1800	12	0:31:04
12/6/2011	1800	13	0:31:39
12/6/2011	1800	14	0:30:41
12/6/2011	1800	15	0:30:11
12/7/2011	1800	16	0:30:12
12/7/2011	1800	17	0:30:57
12/7/2011	1800	18	0:30:27
12/7/2011	1800	19	0:31:04
12/7/2011	1800	20	0:29:58
12/7/2011	1800	21	0:30:55
12/7/2011	1800	22	0:30:56
12/7/2011	1800	23	0:32:04
12/7/2011	1800	24	0:30:33
12/7/2011	1800	25	0:30:12
12/8/2011	1800	26	0:30:17
12/8/2011	1800	27	0:30:19
12/8/2011	1800	28	0:30:35
12/8/2011	1800	29	0:31:06
12/8/2011	1800	30	0:30:31
12/9/2011	1800	31	0:30:00
12/11/2011	1800	32	0:30:22
12/11/2011	1800	33	0:30:18
12/11/2011	1800	34	0:30:20
12/11/2011	1800	35	0:30:03
12/11/2011	1800	36	0:30:26
12/11/2011	1800	37	0:30:12
12/11/2011	1800	38	0:30:39
12/12/2011	1800	39	0:30:02
12/12/2011	1800	40	0:30:47
12/12/2011	1800	41	0:30:23
12/12/2011	1800	42	0:30:29
12/12/2011	1800	43	0:30:51
12/12/2011	1800	44	0:30:11
12/12/2011	1800	45	0:29:59
Total de tempo gasto ao executar 45 testes com 1800 testes de unidade cada			22:54:51

Micro Instance				
Data	**Quantidade de Testes de Unidade**	**Quantidade de Máquinas**	**Experimento**	**Tempo de Execução**
1/9/2012	1800	18	1	00:11:47
1/9/2012	1800	18	2	00:11:41
1/9/2012	1800	18	3	00:11:47
1/9/2012	1800	18	4	00:11:52
1/9/2012	1800	18	5	00:12:03
1/10/2012	1800	18	6	00:11:45
1/10/2012	1800	18	7	00:11:40
1/10/2012	1800	18	8	00:11:44
1/10/2012	1800	18	9	00:11:49
1/10/2012	1800	18	10	00:11:55
1/10/2012	1800	18	11	00:11:47
1/10/2012	1800	18	12	00:11:47
1/10/2012	1800	18	13	00:11:51
1/10/2012	1800	18	14	00:11:49
1/10/2012	1800	18	15	00:11:48
1/10/2012	1800	18	16	00:11:58
1/10/2012	1800	18	17	00:11:54
1/11/2012	1800	18	18	00:11:53
1/11/2012	1800	18	19	00:11:35
1/11/2012	1800	18	20	00:11:32
1/11/2012	1800	18	21	00:11:37
1/11/2012	1800	18	22	00:11:32
1/11/2012	1800	18	23	00:11:35
1/11/2012	1800	18	24	00:11:35
1/11/2012	1800	18	25	00:11:48
1/11/2012	1800	18	26	00:11:38
1/11/2012	1800	18	27	00:11:47
1/11/2012	1800	18	28	00:11:39
11/01/2012	1800	18	29	00:08:54
12/01/2012	1800	18	30	00:08:54
12/01/2012	1800	18	31	00:08:10
12/01/2012	1800	18	32	00:08:41
12/01/2012	1800	18	33	00:08:56
12/01/2012	1800	18	34	00:09:10
12/01/2012	1800	18	35	00:09:27
12/01/2012	1800	18	36	00:10:39
16/01/2012	1800	18	37	00:03:48
18/01/2012	1800	18	38	00:07:06
21/01/2012	1800	18	39	00:09:14
24/01/2012	1800	18	40	00:03:46
20/02/2012	1800	18	41	00:10:46
20/02/2012	1800	18	42	00:11:57
20/02/2012	1800	18	43	00:11:56
20/02/2012	1800	18	44	00:12:38
20/02/2012	1800	18	45	00:11:43
Total de tempo gasto ao executar 45 testes com 1800 testes de unidade cada				08:04:53

Small Instance				
Data	**Quantidade de Testes de Unidade**	**Quantidade de Máquinas**	**Experimento**	**Tempo Gasto**
1/30/2012	1800	18	1	00:05:45
1/30/2012	1800	18	2	00:05:51
1/30/2012	1800	18	3	00:05:45
1/30/2012	1800	18	4	00:05:43
1/30/2012	1800	18	5	00:05:32
1/30/2012	1800	18	6	00:05:34
1/30/2012	1800	18	7	00:05:23
1/30/2012	1800	18	8	00:05:20
1/30/2012	1800	18	9	00:05:32
1/30/2012	1800	18	10	00:05:29
1/30/2012	1800	18	11	00:05:52
1/30/2012	1800	18	12	00:05:29
1/30/2012	1800	18	13	00:05:32
1/30/2012	1800	18	14	00:05:29
1/30/2012	1800	18	15	00:05:26
1/30/2012	1800	18	16	00:05:19
1/30/2012	1800	18	17	00:05:31
1/30/2012	1800	18	18	00:05:25
1/30/2012	1800	18	19	00:05:30
1/31/2012	1800	18	20	00:05:32
1/31/2012	1800	18	21	00:05:35
1/31/2012	1800	18	22	00:05:28
1/31/2012	1800	18	23	0:05:32
1/31/2012	1800	18	24	0:05:33
1/31/2012	1800	18	25	0:05:40
1/31/2012	1800	18	26	0:05:23
1/31/2012	1800	18	27	0:05:28
1/31/2012	1800	18	28	0:05:29
1/31/2012	1800	18	29	0:05:21
1/31/2012	1800	18	30	0:05:34
1/31/2012	1800	18	31	0:05:21
1/31/2012	1800	18	32	0:05:38
1/31/2012	1800	18	33	0:05:21
1/31/2012	1800	18	34	0:05:29
1/31/2012	1800	18	35	0:05:28
1/31/2012	1800	18	36	0:05:37
01/02/2012	1800	18	37	0:05:21
01/02/2012	1800	18	38	0:05:22
01/02/2012	1800	18	39	0:05:21
01/02/2012	1800	18	40	0:05:24
01/02/2012	1800	18	41	0:05:29
01/02/2012	1800	18	42	0:05:21
01/02/2012	1800	18	43	0:05:24
01/02/2012	1800	18	44	0:05:32
01/02/2012	1800	18	45	0:05:21

Total de tempo gasto ao executar 45 testes com 1800 testes de unidade cada	04:07:31

Medium Instance				
Data	**Quantidade de Testes de Unidade**	**Quantidade de Máquinas**	**Experimento**	**Tempo Gasto**
2/3/2012	1800	18	1	00:03:44
2/3/2012	1800	18	2	00:03:51
2/3/2012	1800	18	3	00:03:51
2/5/2012	1800	18	4	00:03:49
2/5/2012	1800	18	5	00:04:01
2/5/2012	1800	18	6	00:03:29
2/6/2012	1800	18	7	00:03:46
2/6/2012	1800	18	8	00:03:55
2/6/2012	1800	18	9	00:03:50
2/6/2012	1800	18	10	00:03:54
2/6/2012	1800	18	11	00:03:39
2/6/2012	1800	18	12	00:03:59
2/6/2012	1800	18	13	00:03:41
2/6/2012	1800	18	14	00:03:41
2/6/2012	1800	18	15	00:03:57
2/6/2012	1800	18	16	00:03:50
2/6/2012	1800	18	17	00:03:59
2/6/2012	1800	18	18	00:03:46
2/6/2012	1800	18	19	00:03:42
2/6/2012	1800	18	20	00:03:49
2/6/2012	1800	18	21	00:03:59
2/6/2012	1800	18	22	00:03:42
2/6/2012	1800	18	23	00:03:32
2/6/2012	1800	18	24	00:03:51
2/6/2012	1800	18	25	00:03:49
2/6/2012	1800	18	26	00:03:54
2/6/2012	1800	18	27	00:03:50
2/6/2012	1800	18	28	00:03:55
2/6/2012	1800	18	29	00:03:57
2/6/2012	1800	18	30	00:03:50
2/6/2012	1800	18	31	00:03:54
2/6/2012	1800	18	32	00:03:52
2/7/2012	1800	18	33	00:03:48
2/7/2012	1800	18	34	00:03:53
2/7/2012	1800	18	35	00:03:44
2/7/2012	1800	18	36	00:03:41
2/7/2012	1800	18	37	00:03:33
2/7/2012	1800	18	38	00:03:46
2/7/2012	1800	18	39	00:03:38
2/7/2012	1800	18	40	00:03:57
2/7/2012	1800	18	41	00:03:59
2/7/2012	1800	18	42	00:03:51
2/7/2012	1800	18	43	0:03:39

2/7/2012	1800	18	44	00:03:12
2/7/2012	1800	18	45	00:03:49
Total de tempo gasto ao executar 45 testes com 1800 testes de unidade cada				02:50:48

Apêndice B – Formulário disponibilizado na Internet para efetuar pesquisa com desenvolvedores de software

Este apêndice apresenta o formulário utilizado para obter indícios anedóticos dos desenvolvedores de *software* sobre quais são as tecnologias mais utilizadas pela indústria.

Survey sobre processos de desenvolvimento e testes de *software*

Olá, Este formulário é composto por 10 questões de múltipla escolha e faz parte da minha pesquisa de mestrado no Programa de Pós-Graduação em Informática da UFPB.

Obrigado por sua contribuição!

Gustavo Sávio

*Obrigatório

1. O processo de desenvolvimento de software utilizado no seu local de trabalho/estudo/pesquisa contempla a realização de testes? * Sim

2. Tais testes são realizados de forma *

- [ ] Automática
- [ ] Manual

3. Qual o tempo máximo que você já gastou para executar um conjunto de testes automáticos? *

	0	1	2	3	4	5	6	7	8	9	10	
hora	○	○	○	○	○	○	○	○	○	○	○	horas

4. Que framework(s) ou ferramenta(s) você utiliza ou já utilizou para desenvolver e executar testes automáticos de software? *

- [ ] JUnit
- [ ] TestComplete
- [ ] Selenium
- [ ] Ranorex
- [ ] Silk Test
- [ ] TestPartner
- [ ] Visual Test
- [ ] PHPUnit
- [ ] Não uso frameworks
- [ ] Outro:

5. Que Ambientes de Desenvolvimento Integrado (IDE) você utiliza ou já utilizou? *

- [ ] Eclipse
- [ ] Jude
- [ ] Xcode
- [ ] NetBeans
- [ ] VisualStudio
- [ ] BlueJ
- [ ] Delphi
- [ ] Aptana
- [ ] Geany
- [ ] Outro:

6. Quais as linguagens de programação suportadas por seu(s) ambiente(s) de desenvolvimento integrado? *

- [ ] Java
- [ ] C/C++
- [ ] Python
- [ ] C#
- [ ] .Net
- [ ] Ruby
- [ ] PHP
- [ ] Outro:

7. Algum dos ambientes de desenvolvimento mencionados na questão 5 suporta plugins desenvolvidos por terceiros? * Sim

8. Você já desenvolveu algum plugin para ambientes de desenvolvimento? * Sim

9. Você utiliza algum plugin no ambiente de desenvolvimento para efetuar teste de software? *

- ( ) Sim
- ( ) Não

10. Seria proveitoso para seu trabalho ser capaz de executar testes automáticos de forma mais rápida, explorando recursos de computação paralela ou distribuída? * Sim