Desenvolvimento de mecanismos para
automatização de planejamento e execução de
experimentos em sistemas orientados a serviço

Luiz Henrique Nunes

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 14/08/2014

Assinatura: _____

Desenvolvimento de mecanismos para automatização de planejamento e execução de experimentos em sistemas orientados a serviço

Luiz Henrique Nunes

***Orientador:* Prof. Dr. Júlio Cezar Estrella**

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Agosto de 2014

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

N972d Nunes, Luiz Henrique
Desenvolvimento de mecanismos para automatização
de planejamento e execução de experimentos em
sistemas orientados a serviço / Luiz Henrique Nunes;
orientador Júlio Cezar Estrella. -- São Carlos,
2014.
58 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática
Computacional) -- Instituto de Ciências Matemáticas
e de Computação, Universidade de São Paulo, 2014.

1. Planejamento de Capacidade. 2. Testes
Funcionais em Sistemas Distribuídos. 3. Testes
Funcionais em Sistemas Orientados a Serviço. 4.
Testes em Arquiteturas Orientadas a Serviço. I.
Estrella, Júlio Cezar, orient. II. Título.

*“O único lugar onde o sucesso vem antes
do trabalho é no dicionário.”*

Albert Einstein

Dedico este trabalho aos meus pais, irmãos, namorada, e amigos.

Agradecimentos

Agradeço primeiramente aos meus queridos pais, Francisco e Zoraide, que sempre incentivaram meus estudos e forneceram a base para tudo que tenho até hoje. As minhas irmãs Vivian, Patrícia e Priscila e ao meu irmão Renato que também sempre me apoiaram, cobraram e torceram por mim nas decisões mais importantes que já tive. À minha namorada Geovana que apoio e estimulou meu ingresso no mestrado. Ao professor Júlio pela orientação, paciência e conhecimento transmitido durante estes dois anos e meio de projeto mas também pelos conselhos que levarei por toda a vida. A professora Regina, que contribuiu com críticas importantíssimas para o desenvolvimento do projeto final. Aos demais professores e aos colegas de laboratório, que proporcionaram várias discussões e momentos de risadas. Muito Obrigado.

Resumo

O planejamento de experimentos em sistemas computacionais não é uma tarefa trivial, pois envolve diversas etapas tais como, o planejamento propriamente dito, a execução dos experimentos e a análise dos resultados. A definição e a utilização de metodologias adequadas para cada uma destas etapas facilita a obtenção dos resultados de um experimento em um sistema computacional. Neste trabalho são apresentados mecanismos para auxiliar o planejamento e execução de experimentos em sistemas orientados a serviços. O planejamento de experimento é realizado a partir de um modelo baseado nos conjuntos de entradas comuns a arquiteturas orientadas a serviço. A execução deste planejamento é feita em um ambiente colaborativo real, a qual auxilia a identificação de gargalos que não estão presentes em simulações ou modelos analíticos. Um estudo de caso aplicado na arquitetura WSARCH, possibilitou avaliar seu desempenho e identificar problemas de configuração.

Abstract

The design of experiments in computational systems is not a trivial task as it involves several steps such as planning and execution of the experiments and the analyse of the results. The use of appropriate methodologies for each of these steps makes it easier obtain the experiment results of a computer system. In this dissertation, mechanisms to assist the planning and execution of experiments in service-oriented systems are presented. The planning of the experiment is made according to a model based on a set of common entries for service-oriented architectures. The experiment execution is performed in a real collaborative environment, which helps to identify bottlenecks that are not found in simulations or analytical models. A study case applied in WSARCH architecture, enables to evaluate the performance and identify configuration problems.

Sumário

Resumo	i
Abstract	iii
Lista de Figuras	viii
Lista de Tabelas	ix
Lista de Siglas	xi
1 Introdução	1
1.1 Considerações Iniciais	1
1.2 Contextualização	2
1.3 Motivação e Objetivos	3
1.4 Estrutura	4
2 Fundamentação Teórica	5
2.1 Arquiteturas Orientadas a Serviço (SOA)	5
2.1.1 Arquitetura WSARCH	6
2.2 <i>Web Services</i>	9
2.2.1 Protocolo SOAP	10
2.2.2 Conceitos Arquiteturais REST	11
2.3 Considerações Finais	12
3 Trabalhos Relacionados	13
4 PEESOS	19
4.1 Visão geral	19
4.2 Metodologias	21
4.3 Desenvolvimento da Ferramenta	22
4.3.1 Arquitetura	22
4.3.1.1 Monitor	23
4.3.1.2 <i>Transport</i>	23
4.3.1.3 Workload	24
4.3.1.4 Core	25
4.3.2 Funcionamento	29

4.3.2.1	Interface	29
4.3.2.2	Ferramenta	33
4.4	Considerações Finais	35
5	Resultados	37
5.1	Configurações do Ambiente	37
5.2	Configurações dos Experimentos	38
5.3	Resultados	41
5.4	Análise dos Resultados	46
5.5	Problemas Identificados	47
5.6	Considerações Finais	48
6	Conclusão	49
6.1	Considerações Iniciais	49
6.2	Conclusões e Contribuições	49
6.3	Produções Científicas	50
6.3.1	Artigos aceitos para publicação	50
6.4	Trabalhos Futuros	51

Lista de Figuras

2.1	Elementos que compõem uma Arquitetura Orientada a Serviços. Adaptado de (Erl, 2008)	6
2.2	Funcionamento típico de uma Arquitetura Orientada a Serviços (Papazoglou, 2003).	7
2.3	A arquitetura WSARCH e seus componentes (Estrella et al., 2011)	7
2.4	Componentes de um <i>Web Service</i> . Adaptado de Rosen et al. (2008)	10
2.5	Estrutura da mensagem SOAP (adaptado de PwC (2013))	11
2.6	Modelo de comunicação da mensagem SOAP (adaptado de PwC (2013))	11
2.7	Estrutura da mensagem REST (adaptado de PwC (2013))	12
2.8	Modelo de comunicação da mensagem REST (adaptado de PwC (2013))	12
4.1	Fluxo de Funcionamento da Ferramenta	20
4.2	Visão geral da proposta dos componentes do planejador de experimentos	21
4.3	Diagrama de pacotes do atual estágio do projeto	22
4.4	Diagrama de estado do Módulo <i>Remote Connection</i> - Servidor	23
4.5	Diagrama de estado Módulo <i>Transport</i> - Cliente	24
4.6	Diagrama de estado do módulo <i>Workload</i>	25
4.7	Diagrama de classes simplificado	26
4.8	Diagrama de estado do módulo <i>Core</i>	26
4.9	Fluxograma das etapas de configuração do experimento.	28
4.10	Página inicial da ferramenta	29
4.11	Página de <i>download</i> dos arquivos gerados	29
4.12	Página de configuração do ambiente de testes do experimento com 1 nível 4.12(a) e 2 níveis 4.12(b)	30
4.13	Página de configuração dos parâmetros do serviço com 1 nível 4.13(a) e 2 níveis 4.13(b)	31
4.14	Página para seleção dos clientes disponíveis	32
4.15	Página para seleção dos Provedores disponíveis	32
4.16	Página de revisão do experimento	33
4.17	Página para início da execução do experimento	33
4.18	Página com experimento em execução	33
4.19	Diagrama de Sequência da ferramenta	34
5.1	Ambiente de teste	38
5.2	Gráfico com Intervalo de confiança e média do tempo total das requisições	41

5.3	Boxplot do tempo total das requisições	42
5.4	Gráficos de dispersão dos experimentos com distribuição exponencial para os fatoriais de 1.000, 8.000 e 16.000	43
5.5	Gráficos de dispersão dos experimentos com distribuição uniforme para os fatoriais de 1.000, 8.000 e 16.000	43
5.6	Gráfico com intervalo de confiança e média do tempo de busca do provedor de serviços	44
5.7	Gráfico com intervalo de confiança e média do tempo de processamento do Broker	44
5.8	Gráfico com intervalo de confiança e média do tempo do cliente sem o tempo de busca e seleção do serviço	45
5.9	Gráficos em linha com os tempos médios de resposta das requisições para as distribuições Exponencial (5.9(a)) e Uniforme (5.9(b))	46
5.10	Gráficos em linha com os tempos médios de atendimento das requisições para as distribuições Exponencial (5.10(a)) e Uniforme (5.10(b)) no <i>Broker</i>	47

Lista de Tabelas

3.1	Objetivos, modelagem e forma de avaliação. Adaptado de Smit e Stroulia (2012)	17
5.1	Configurações dos Clientes e da Ferramenta	38
5.2	Configurações dos componentes da arquitetura WSARCH	39
5.3	Fatores e níveis do experimento	39
5.4	Modelo dos Experimentos	40

Lista de Siglas

API *Application Programming Interface*

BPEL *Web Services Business Process Execution Language*

CORBA *Common Object Request Broker Architectures*

CPU *Central Processing Unit*

DCOM *Distributed Component Object Model*

DDSOS *Distributed Service-Oriented Simulation*

DOE *Design of Experiments*

FAPESP *Fundação de Amparo à Pesquisa do Estado de São Paulo*

FDA *Função de Distribuição Acumulada*

HTTP *Hypertext Transfer Protocol*

IP *Internet Protocol*

JSCH *Java Security Channel*

JSON *JavaScript Object Notation*

JVM *Java Virtual Machine*

PEESOS *Planning and Execution of Experiments in Services Oriented Systems*

PSML-S *Process Specification and Modeling Language for Services*

OASIS *Organization for the Advancement of Structured Information Standards*

QoS *Quality of service*

REST *Representational State Transfer*

RMI *Remote Method Invocation*

SASF *Services-Aware Simulation Framework*

SLA *Service-level agreement*

SOA *Service-oriented architecture*

SOC *Service-oriented computing*

SOAP *Simple Object Access Protocol*

SOPM *Service-Oriented Performance Modeling*

SSH *Secure Shell*

TESSI *Testing of Service Implementations*

UDDI *Universal Description, Discovery and Integration*

URI *Uniform Resource Identifier*

URL *Uniform Resource Locator*

W3C *World Wide Web Consortium*

WADL *Web Application Description Language*

WSARCH *Web Services Architecture*

WS-BPEL *Web Services Business Process Execution Language*

WSDL *Web Services Description Language*

XSD *XML Schema Definition*

XML *eXtensible Markup Language*

Introdução

1.1 Considerações Iniciais

O planejamento de experimentos em sistemas computacionais não é uma tarefa trivial, pois envolve diversas etapas tais como, o planejamento propriamente dito, a execução dos experimentos e a análise dos resultados (Jain, 1991). A definição e a utilização de metodologias adequadas para cada uma destas etapas facilita a obtenção dos resultados de um experimento em um sistema computacional. Considerando os sistemas atuais, os quais grande parte encontram-se disponíveis na Internet, torna-se dispendiosa a realização de experimentos sem a utilização de mecanismos que considerem diferentes arquiteturas alvo, diferentes linguagens de programação, cargas de trabalhos distintas, heterogeneidade do ambiente computacional e falhas.

Atualmente, a Internet é constituída de diversos *hosts* interconectados por elementos de roteamento de pacotes, bem como de servidores Web e provedores de serviços, que fornecem conteúdo tanto para os usuários quanto para outros dispositivos.

De natureza distribuída, a Web permite a comunicação entre sistemas por meio dos chamados *Web Services*, os quais permitem a interoperabilidade entre aplicações sem se preocupar com diferenças de linguagens de programação, sistemas operacionais e *hardwares* utilizados. Por isso, o desenvolvimento e a implantação de aplicações orientadas a serviços aumentou drasticamente nos últimos anos. Isso somente foi possível devido à disponibilidade e desenvolvimento dos padrões e tecnologias para os *Web Services*.

Nesse sentido, um efeito transformador ocorreu sobre as comunidades científicas, pois ferramentas que costumavam ser destinadas a especialistas, foram disponibilizadas para todos e

permitiu a automatização de tarefas de processamento de dados que anteriormente eram manuais (Li et al., 2011).

Apesar de a interoperabilidade ser o ponto central de arquiteturas orientadas a serviços (do inglês, *Service-oriented architecture* (SOA)) desenvolvidas com o uso de *Web Services*, poucas são as preocupações com as soluções para verificar e garantir qualidade de serviço na interação entre aplicações executadas em sistemas distintos. Para amenizar os problemas relativos à SOA, questões de desempenho e qualidade de serviço para os clientes dos serviços devem ser consideradas no estágio de desenvolvimento de um sistema.

1.2 Contextualização

A utilização de arquiteturas SOA nos mais diversos domínios (bancos, bolsas de valores, integração de sistemas em geral) cresceu consideravelmente nos últimos anos, necessitando assim do funcionamento de um sistema confiável. Corporações que estão migrando seus sistemas para SOA devem dedicar esforços ao teste do funcionamento destes sistemas. O sucesso de qualquer aplicação desenvolvida depende do processo de garantia de qualidade aplicado (Kalamegam e Godandapani, 2012).

No entanto, é impossível garantir que um sistema computacional seja 100% confiável, pois é preciso considerar falhas de *hardware*, *software* e erro humano, ainda mais considerando a natureza distribuída que arquiteturas SOA podem apresentar.

Há dois importantes fatores a considerar ao se utilizar arquitetura SOA e *Web Services*: de um lado os provedores de serviços preocupam-se em alocar o mínimo de recursos possíveis enquanto de outro, esses recursos devem ser suficientes para garantir o acordo de nível de serviço (do inglês, *Service-level agreement* (SLA)) firmado com um cliente (Huang et al., 2011). É comum utilizar um conjunto de serviços, para a realização de tarefas mais complexas. Esse grupo de serviços é composto de diferentes serviços que executam concorrentemente, contribuindo para a variação dinâmica da carga de trabalho requerida pelo *Web Service* (Cui et al., 2011).

Com o intuito de garantir o funcionamento e a integração de todos os componentes pertencentes a uma SOA, testes devem ser realizados constantemente, pois a possibilidade de falhas em uma SOA é proporcional a quantidade de serviços nela oferecidos (Bozkurt et al., 2013).

A concorrência de um grande número de serviços muitas vezes é um problema ao utilizar SOA, pois ao buscar individualmente a solução ótima, sem considerar o desempenho de todo o sistema, a qualidade do serviço (do inglês, *Quality of service* (QoS)) oferecido diminui, pois os recursos disponíveis podem não ser suficientes para executar os serviços requisitados (Huqqani et al., 2010).

Uma das características principais para provisão de QoS em SOA e garantir o SLA é a resiliência contra falhas do servidor. Uma boa decisão de planejamento de capacidade, é capaz de sobrecarregar a arquitetura alvo e explorar mecanismos de tolerância a falhas. Além disso,

verifica o funcionamento de componentes essenciais para o atendimento de uma requisição tais como: aplicações, serviços e banco de dados.

1.3 Motivação e Objetivos

A imprevisibilidade de evolução de serviços, variações na carga de trabalhos e flexibilidade da infraestrutura tornam implementações SOA altamente dinâmicas. Isto aumenta os custos e os riscos da replicação de todas as possíveis configurações de um serviço, sob diferentes cargas de trabalho durante o teste. O teste de um sistema altamente distribuído é uma tarefa significativa, e o planejamento de testes de forma adequada é essencial para executá-los de maneira eficiente e eficaz (Kalamegam e Godandapani, 2012).

Para os propósitos do desenvolvimento deste projeto de mestrado é importante mencionar a utilização de uma arquitetura prototipada denominada *Web Services Architecture* (WSARCH). O desenvolvimento desta arquitetura foi apoiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) em primeira fase durante o doutorado do orientador do candidato finalizado em 2010, sob o processo 06/55207-5. O protótipo da arquitetura recebeu novamente o apoio da FAPESP para um projeto de pesquisa regular sob o processo 2011/09524-7, com período de execução entre 2012 a 2014. As características e detalhes da arquitetura serão apresentados no Capítulo 2 desta dissertação.

A concepção do protótipo da WSARCH (Estrella et al., 2011) é apenas o início de uma linha de investigação que deve se estender nos próximos anos em torno da proposta da arquitetura. Um dos aspectos ainda não considerados com a utilização do protótipo é a preocupação com levantamento de parâmetros de desempenho associados à arquitetura e aos serviços nela disponíveis, de forma que seja possível delinear modelos de planejamento de experimentos, execução e análise dos resultados para as aplicações hospedadas na WSARCH e posteriormente em qualquer sistema computacional alvo.

Considerando inicialmente a arquitetura WSARCH como sistema alvo e que há atualmente um ciclo de desenvolvimento de sistemas que não levam em conta quais as situações/condições de sobrecarga do sistema desenvolvido, é fundamental que haja uma dinamicidade no planejamento, na execução e na análise dos resultados, de modo a auxiliar no diagnóstico de soluções adequadas antes que a arquitetura alvo seja disponibilizada para uso. Tal diagnóstico deve auxiliar desenvolvedores e arquitetos de sistemas a considerar aspectos de QoS e desempenho no desenvolvimento de soluções, que serão disponibilizadas em um ambiente distribuído como a Internet.

Este projeto de mestrado apresenta mecanismos para auxiliar o planejamento e execução de experimentos em sistemas computacionais, com destaque para a utilização de um sistema/arquitetura alvo denominado WSARCH. Para isso foi desenvolvida uma ferramenta denominada

de Planejamento de Experimentos e Execução em Sistemas Orientados a Serviços (do inglês, *Planning and Execution of Experiments in Services Oriented Systems* (PEESOS)).

Esta ferramenta é composta por uma interface Web para guiar o usuário do sistema na condução, instrumentação e monitoramento de experimentos para a avaliação de desempenho de uma SOA. Ainda, é responsável por orquestrar todas as etapas necessárias para a realização dos experimentos, desde a implantação dos serviços nos provedores até a distribuição da carga de trabalho gerada. Diferentemente de trabalhos encontrados na literatura, que baseiam-se em modelos analíticos ou simulações, este trabalho utiliza ambientes reais para a execução dos experimentos e consequentemente obtenção de resultados mais precisos.

1.4 Estrutura

O restante desta dissertação está organizado da seguinte maneira:

- O Capítulo 2 apresenta os conceitos principais de SOA, a arquitetura WSARCH e os fundamentos sobre *Web Services*.
- O Capítulo 3 apresenta trabalhos relacionados ao tema desta pesquisa;
- O Capítulo 4 apresenta o projeto de pesquisa desenvolvido juntamente com a metodologia e ferramentas utilizadas;
- O Capítulo 5 descreve os experimentos realizados para validar o projeto de pesquisa proposto;
- O Capítulo 6 apresenta uma análise do desenvolvimento do trabalho e suas principais contribuições. Neste capítulo também são apresentados os trabalhos futuros e as produções científicas realizadas até o momento.

Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos de SOA e *Web Services*. Na seção 2.1 são discutidos os princípios fundamentais de SOA e seu funcionamento. A Seção 2.1.1 aborda os conceitos principais da arquitetura WSARCH e seu funcionamento. Na Seção 2.2, são discutidos os conceitos principais sobre *Web Services*. A Seção 2.2.1 descreve as principais características dos *Web Services* desenvolvidos com o protocolo simples de acesso a objetos (do inglês, *Simple Object Access Protocol* (SOAP)). Na Seção 2.2.2, são descritos as características dos *Web Services* desenvolvidos a partir dos conceitos de transferência de estado representativo (do inglês, *Representational State Transfer* (REST)) . Por fim, a Seção 2.3 apresenta as considerações finais sobre o SOA e *Web Services*.

2.1 Arquiteturas Orientadas a Serviço (SOA)

Computação orientada a serviços (do inglês, *Service-oriented computing* (SOC)) é um paradigma de computação que utiliza serviços como elementos fundamentais para o desenvolvimento de aplicações. Serviços são auto-descritores, independentes de plataformas que auxiliam o desenvolvimento rápido e de baixo custo de aplicações distribuídas (Papazoglou, 2003). Uma abordagem ‘orientada a serviços’ baseia-se na ideia de composição de aplicações por descoberta e invocação *Web Services* disponíveis na rede, para realização tarefas específicas (Papazoglou et al., 2007).

SOA é um estilo arquitetural para o desenvolvimento de soluções corporativas, baseada em serviços. Mais especificamente, as maiores preocupações ao aplicar os conceitos de SOA

consistem na construção de serviços independentes, que podem ser combinados, para atender tarefas de um processo de negócio (Rosen et al., 2008).

Figura 2.1: Elementos que compõem uma Arquitetura Orientada a Serviços. Adaptado de (Erl, 2008)

Uma implementação SOA consiste da combinação de tecnologias, produtos, APIs, extensões de infraestrutura, e alguns outros fatores tal como é mostrado na Figura 2.1. A arquitetura SOA utilizada em cada corporação é única, porém configurada para aceitar novas tecnologias e plataformas, que suportam a criação, execução e evolução de soluções orientadas a serviço. Como resultado, a implantação destes modelos viabiliza a aplicação de soluções, em conformidade com padrões de desenvolvimento orientados a serviço (Erl, 2008).

Em uma SOA, identifica-se a relação entre três tipos de participantes: o provedor de serviços, o repositório de serviços e o requisitor de serviços (cliente). As interações entre estes participantes envolvem as operações de busca, ligação e publicação dos serviços. A execução destas operações são baseados nos arquivos de descrição de serviço e sua implementação (Papazoglou, 2003).

Na Figura 2.2 é representado um cenário típico do funcionamento de uma SOA. O provedor de serviços, contém o *Web Service* com determinada funcionalidade, e é o responsável por sua publicação. O cliente do serviço busca a operação desejada nos *Web Services* cadastrados no repositório de serviços. A partir das informações obtidas no repositório, estabelece-se a ligação entre provedor e cliente e a requisição é realizada (Papazoglou, 2003).

2.1.1 Arquitetura WSARCH

A arquitetura WSARCH encontra-se implementada e os resultados já alcançados com a modelagem e a prototipação sinalizam para melhorias em relação à provisão de QoS entre as aplicações (Estrella et al., 2011; Cardellini et al., 2011). O propósito da arquitetura é servir de base para que problemas relacionados à interoperabilidade sejam diagnosticados detalhadamente, uma vez que envolvem principalmente aspectos relacionados à caracterização de carga

Figura 2.2: Funcionamento típico de uma Arquitetura Orientada a Serviços (Papazoglou, 2003).

de trabalho (Tavares et al., 2008), composição de serviços (Kuehne et al., 2010), rede de comunicação, segurança no contexto de mensagens trocadas entre os componentes de uma arquitetura orientada a serviços e o gerenciamento de falhas.

Figura 2.3: A arquitetura WSARCH e seus componentes (Estrella et al., 2011)

Na Figura 2.3 é mostrado funcionamento geral da arquitetura WSARCH, na qual destacam-se os seguintes componentes:

- **Provedor de serviços** - Disponibiliza um *Web Service* para um cliente. A localização deste serviço deve ser registrada num repositório denominado *Universal Description, Discovery and Integration* (UDDI). O provedor de serviços disponibiliza ao *Broker* informações sobre a qualidade do serviço oferecido aos clientes *Web Services*, tais como uso da unidade central de processamento (do inglês, *Central Processing Unit* (CPU)), *throughput* do sistema, etc;
- **Broker** - Responsável por realizar a busca de um serviço de acordo com as necessidades do cliente. Deve gerenciar informações de QoS vindas do provedor de serviços. Outro

aspecto importante é a negociação, visto que tentará obter o melhor serviço de acordo com as informações disponíveis sobre o provedor. Caso contrário, uma decisão deve ser tomada para que ou o serviço seja atendido com critérios de QoS relaxados ou não seja mais atendido. Uma base de dados é utilizada para o armazenamento de mensagens sobre a qualidade de serviço;

- **Cliente** - Faz o papel de requisitor de um determinado serviço. Primeiramente o cliente realiza uma requisição de um serviço por meio do *Broker*, trocando informações de QoS com ele. Após encontrar o serviço desejado pelo cliente, o *Broker* repassa informações para o cliente (tais como: localização dos *Web Services*, qual sua descrição e como invocá-lo). Deste ponto em diante, o cliente invoca o serviço diretamente do provedor de serviços. A utilização do *Broker* como um agente de roteamento/escalonamento de mensagens permite oferecer ao cliente condições para que tenha suas requisições atendidas de acordo com os contratos estabelecidos com o provedor de serviços. Isso significa que o *Broker* somente vai escolher provedores capazes de cumprir com o acordo firmado com o cliente

Ainda em relação à Figura 1 da arquitetura WSARCH, seus componentes e as possíveis interações entre eles são apresentados de modo simplificado a seguir. As seguintes etapas são necessárias para o atendimento de uma requisição:

1. Cliente faz requisição ao *Broker*, o qual possui informações atualizadas do provedor do serviço (carga, tipo de serviço, classe de cliente, entre outros);
2. Com base nas informações de QoS solicitadas pelo cliente do serviço, o *Broker* realiza uma busca num repositório de serviços com o objetivo de encontrar o serviço mais adequado;
3. *Broker* obtém a especificação do serviço apropriado e as respectivas informações de QoS do provedor;
4. Com a localização do serviço apropriado no repositório de serviços, e já de posse das informações dos provedores de serviços, o *Broker* escolhe o melhor provedor candidato (Seleção de Serviços);
5. Após a execução da função de seleção de serviços, o *Broker* realiza a requisição (invocação do Web Service) no provedor de serviços;
6. Depois de executada a requisição, a resposta é retornada diretamente ao *Broker* da WSARCH;
7. Finalmente a resposta é encaminhada ao cliente que inicialmente solicitou o *Web Service*.

Outras atividades são realizadas em conjunto com a requisição de um cliente, além das etapas referentes a uma requisição de serviço. As informações de QoS dos provedores de serviços são atualizadas periodicamente no registro de serviço para cada um dos provedores cadastrados. Essa propagação de informações ocorre em função da utilização de uma aplicação de monitoração presente nos provedores de serviços e no UDDI, denominada Ganglia (Massie et al., 2003).

Os provedores de serviços possuem monitores escravos que enviam as informações para um monitor mestre no registro UDDI, de modo que o *Broker* da WSARCH possa utilizar a informação de QoS para a seleção do melhor provedor de serviços em determinada ocasião. Como a arquitetura proposta é utilizada também para avaliar desempenho de *Web Services*, um novo componente denominado LogServer foi adicionado à WSARCH. Esse componente é opcional e utilizado para armazenar dados sobre o desempenho dos vários componentes da WSARCH (Estrella et al., 2011, 2010).

2.2 Web Services

Web Services são um sistema de software projetado para suportar interoperabilidade entre máquinas sobre a rede (W3C, 2013). A plataforma de *Web Services* utiliza diversos padrões industriais cuja organização é responsabilidade da *World Wide Web Consortium* (W3C) e *Organization for the Advancement of Structured Information Standards* (OASIS). Esta plataforma pode ser separada em duas gerações (Erl, 2008):

- **1ª Geração:** Originalmente a plataforma de *Web Services* foi utilizada com as seguintes especificações e tecnologias: arquivos de linguagem para descrição dos serviços (do inglês, *Web Services Description Language* (WSDL)) por meio de linguagens de marcação (do inglês, *eXtensible Markup Language* (XML)), arquivos para descrever a estrutura de documentos XML (do inglês, *XML Schema Definition* (XSD)), SOAP e UDDI. Entretanto, os *Web Services* desta geração apresentaram problemas relacionados a atributos de QoS, tais como segurança e confiabilidade.
- **2ª Geração:** Para solucionar os problemas relacionados a QoS da primeira geração de *Web Services*, várias especificações foram criadas para complementar suas funcionalidades e atender os requisitos não-funcionais desejados. Estas especificações são geralmente nomeadas baseado no padrão 'WS-*', tais como *WS-Security* e *WS-Policy*.

Um *Web Service* também é definido como uma unidade discreta de uma funcionalidade de negócio que é disponibilizada por meio de um SLA. O SLA é responsável por especificar todas as interações entre o provedor de serviços e o cliente. Além disso, também gerenciam atributos de QoS e desempenho (Rosen et al., 2008).

Figura 2.4: Componentes de um *Web Service*. Adaptado de Rosen et al. (2008)

Na Figura 2.4 são representados os principais componentes de um *Web Service*, na qual destacam-se a interface e a implementação. A interface de um *Web Service* especifica as operações, os parâmetros e os protocolos para sua comunicação. Já a implementação de um *Web Service*, contém o código das operações providas para a interface. Estas operações podem ser baseadas em operações já existentes, na orquestração de outros *Web Services* ou em uma combinação entre eles (Rosen et al., 2008).

Desta maneira, *Web Services* diferem dos tradicionais *frameworks* de integração de software tais como *Common Object Request Broker Architectures* (CORBA), *Distributed Component Object Model* (DCOM) e *Java Remote Method Invocation* (RMI) por possuírem protocolos bem definidos para comunicação, delineados pelo protocolo de transferência de hipertexto (do inglês, *Hypertext Transfer Protocol* (HTTP)) e XML. Geralmente *Web Services* são divididos em duas categorias: *Web Services* baseados no protocolo SOAP e *Web Services* baseados nos conceitos arquiteturais REST.

2.2.1 Protocolo SOAP

SOAP é um protocolo simples baseado em XML, que permite a troca de informações estruturadas em um ambiente distribuído. Na Figura 2.5 é apresentada a estrutura de uma mensagem SOAP, a qual contém um elemento chamado de envelope que é constituído por duas partes: o *header* e o *body*. O *header* contém informações específicas da aplicação, as quais podem ser utilizadas para rotear a mensagem ou buscar níveis de qualidade de serviço, enquanto que o *body*

contém o conteúdo da mensagem a ser enviada (Pautasso et al., 2008; Mohamed e Wijesekera, 2012).

Figura 2.5: Estrutura da mensagem SOAP (adaptado de PwC (2013))

Figura 2.6: Modelo de comunicação da mensagem SOAP (adaptado de PwC (2013))

Um arquivo WSDL é responsável por definir um esquema XML que descreva os serviços desenvolvidos com este protocolo e como acessá-los. Esse arquivo abstrai detalhes do protocolo de comunicação e serialização assim como a plataforma de implementação do serviço (Sistema Operacional e linguagem de programação). O contrato WSDL fornece um descritor processável pela máquina com a sintaxe e a estrutura das mensagens de requisição e resposta, o qual possibilitam uma evolução flexível do serviço (Pautasso et al., 2008). Na Figura 2.6 é representada a comunicação entre provedores e consumidores de serviços com o auxílio de mensagens SOAP.

2.2.2 Conceitos Arquiteturais REST

REST é um estilo arquitetural que utiliza protocolos livres de estado para comunicação, tipicamente HTTP, em que clientes e servidores trocam representações dos recursos por meio de uma interface padrão conforme ilustrado na Figura 2.8. Estes recursos possuem um identificador uniforme de recursos (do inglês, *Uniform Resource Identifier* (URI)) e são manipulados por meio de quatro métodos HTTP: *GET*, *PUT*, *POST* e *DELETE*, exemplificados na Figura 2.7. Pode-se utilizar arquivos desenvolvidos com WSDL 2.0 ou linguagem de descrição de aplicações Web (do inglês, *Web Application Description Language* (WADL)) para descrever os serviços REST, embora não haja necessidade, já que geralmente estes serviços são descritos por meio de suas próprias URI. As principais vantagens dos serviços RESTful incluem (Kanagasundaram et al., 2012; Mohamed e Wijesekera, 2012):

- **Endereços únicos** - os recursos são acessados por meio de sua URI, dispensando o uso de um recurso separado para descobrir e localizar serviços (Kanagasundaram et al., 2012);
- **Livres de estado** - cada requisição cliente possui todos os dados necessários para o provedor realizar a operação e são independentes, ou seja, não estão relacionadas com as requisições anteriores (Kanagasundaram et al., 2012);

Figura 2.7: Estrutura da mensagem REST (adaptado de PwC (2013))

Figura 2.8: Modelo de comunicação da mensagem REST (adaptado de PwC (2013))

- **Vários suportes para o acesso aos dados** - a informação pode ser acessada utilizando vários formatos, tais como texto plano, notação de objetos *javascript* (do inglês, *JavaScript Object Notation* (JSON)) ou XML (Kanagasundaram et al., 2012).

2.3 Considerações Finais

Neste Capítulo foram discutidos os principais conceitos de SOA e *Web Services*.

ArquiteturasSOA é um modelo arquitetural utiliza *Web Services* como elementos fundamentais para desenvolver suas aplicações.

REST e SOAP são dois tipos de *Web Services* que permitem a troca de informações de forma estruturada em formato de máquina Kanagasundaram et al. (2012). Entretanto, cada uma possui suas próprias características e limitações, o que as tornam mais ou menos adequadas para cada tipo de aplicação (AlShahwan et al., 2011). Segundo Pautasso et al. (2008), a escolha pelo modelo arquitetural REST remove a necessidade de se fazer uma série de decisões arquiteturais relacionadas as camadas estabelecidas pelo protocolo SOAP, diminuindo sua complexidade. Porém para funcionalidades avançadas, tal como segurança, não é tão simples estender o protocolo REST para suportá-las, sendo preferível a utilização do protocolo SOAP.

No próximo Capítulo será realizada uma revisão sobre trabalhos relacionados a este projeto de mestrado, no qual serão destacadas as principais contribuições e falhas de cada trabalho analisado.

Trabalhos Relacionados

Arquiteturas SOA surgiram como um eficiente paradigma para o rápido crescimento das aplicações corporativas. Sistemas baseados em SOA utilizam *Web Services* como estrutura básica, pois permitem unificar plataformas diferentes e distribuídas por meio de tecnologias, tais como JSON e XML (Teixeira et al., 2010). Porém, enquanto arquiteturas SOA prometem suportar as necessidades de negócio exigidos cada vez mais por programas complexos, o planejamento, desenvolvimento, implantação e teste desses sistemas não é uma tarefa trivial (Tilley et al., 2008).

Meirong e Shaoming (2012) mostram os principais problemas ao realizar experimentos com arquiteturas SOA. O primeiro problema é a integração de todos os recursos disponíveis em uma SOA, principalmente se estiverem em posições geográficas diferentes. Outro problema, é a falta de um ambiente intuitivo para a realização desses experimentos. Por fim, o monitoramento do desempenho das aplicações desenvolvidas para estas arquiteturas, de acordo com as atualizações feitas com o tempo, ou de acordo com a quantidade de requisições desta aplicação.

Liu et al. (2007) e Tekli et al. (2012) apresentam o desempenho de arquiteturas SOA como o fator crucial para o sucesso de uma aplicação em larga escala. Problemas de desempenho podem acarretar vários tipos de consequências, como perda de lucros, diminuição da produção e também má reputação de uma companhia. Canfora e Penta (2009) apresentam uma revisão bibliográfica sobre testes em SOA, e caracterizam estes testes em quatro categorias:

- **Testes Funcionais:** validam as propriedades funcionais de um serviço;
- **Testes de Integração:** garantem que todos os componentes de um sistema estejam funcionando como o esperado;

- **Testes de Regressão:** autenticam as funcionalidades de um sistema baseado em testes já realizados. Este teste ocorre quando alguma modificação é feita em um sistema em funcionamento;
- **Testes Não Funcionais:** garantem que as propriedades não-funcionais, tais como, QoS, tempo de resposta, segurança entre outros apresentem valores adequados durante a execução de um serviço.

Testes funcionais podem ser divididos em outras três categorias: testes unitários, testes de composição e testes fim-a-fim. Testes unitários validam o funcionamento de um serviço. Testes de composição validam os fluxos que compõem um serviço composto e os serviços que fazem parte destes serviços compostos. Testes fim-a-fim testam todos os fluxos de negócio que constituem um serviço simples ou composto (Kalamegam e Godandapani, 2012).

Segundo Kalamegam e Godandapani (2012), para realizar testes fim-a-fim quatro passos devem ser realizados:

1. Identificar os principais cenários de teste;
2. Identificar todas as composições dinâmicas de um serviço;
3. Estabelecer um ambiente de teste real;
4. Reportar e analisar os testes obtidos;

Xing e Yao (2010) propõem o desenvolvimento de um sistema capaz de realizar experimentos colaborativos em uma arquitetura SOA. Por meio desta ferramenta, usuários que estão conectados a Internet em diferentes parte do mundo são capazes de se comunicar e colaborar com requisições de serviços nessas arquiteturas, permitindo assim a realização de um experimento em um ambiente real. Porém este trabalho não apresenta formas de identificar os principais cenários de teste.

Petrova-Antonova et al. (2013) apresentam uma ferramenta para testes de implementação de serviço (do inglês, *Testing of Service Implementations* (TESSI)) como parte de um *framework* denominado TASSA para obter um ambiente para testes de serviços. TESSI possibilita a geração de casos de teste e sua execução, por meio de modelos baseados nos protocolos HTTP, SOAP e *Web Services Business Process Execution Language* (BPEL). Entretanto, de acordo com Kalamegam e Godandapani (2012), a geração destes testes deve ter valores especificados pelo usuário, uma vez que grande parte dos resultados dos testes gerados de forma automática não são aproveitados. Outros trabalhos como WSDLTest (Sneed e Huang, 2006), monadWS (Zhang et al., 2011) e BPELUnit (Mayer e Lübke, 2006) também são capazes de automatizar os testes baseados em arquivos de descrição de serviços.

Gu e Ge (2009) apresentam um método para gerar casos de testes automáticos baseados em dois passos consecutivos. O primeiro passo é a análise do fluxo de execução, no qual

fatores de QoS são identificados baseado em experimentos já realizados. No segundo passo um algoritmo genético é usado para estabelecer o experimento a ser executado, a partir dos valores de mínimo/máximo dos fatores de QoS. Porém, neste trabalho os autores trabalham com a premissa de que já existem experimentos realizados com determinados serviços.

Algumas ferramentas como SoapUI (SmartBear, 2014) e CloudPort (Networks, 2014) podem criar ambientes virtuais e executar testes automáticos em SOA baseados em arquivos de descrição de serviços como WSDL. Tais ferramentas podem realizar testes funcionais, de regressão, de carga entre outros. Porém, tais ferramentas são de uso comercial, com código fechado para as versões completas e inviáveis financeiramente para pequenos grupos.

Tsai et al. (2006) apresentam o *framework* de simulação orientado a serviços distribuídos (do inglês, *Distributed Service-Oriented Simulation* (DDSOS)). Seu principal objetivo é dar suporte ao desenvolvimento dos serviços e testar suas composições durante a simulação. Este *framework* apresenta sérias limitações, já que trabalha apenas com a simulação da arquitetura alvo e dos serviços, não considerando fatores encontrados em ambientes reais. Além disso, esta simulação é configurada por meio de uma linguagem específica, denominada especificação de processos e linguagem de modelagem para serviços (do inglês, *Process Specification and Modeling Language for Services* (PSML-S)) (Tsai et al., 2007).

Grundy et al. (2006) utilizam uma plataforma de testes denominada MaramaMTE para gerar testes de carga para serviços compostos estáticos. O objetivo principal deste trabalho é permitir a análise das interações entre serviços compostos, as quais podem impactar atributos não funcionais tais como QoS durante sua composição. As cargas são feitas por meio de clientes emulados, no qual os intervalos entre as requisições dos serviços são definidos pelo desenvolvedor. Os resultados dos testes de composição apresentados neste trabalho são precisos, porém não são capazes de prever ou emular o desempenho de serviços em um ambiente real.

Brebner (2009) apresenta o *framework* Modelagem de Serviços Orientados a Desempenho (do inglês, *Service-Oriented Performance Modeling* (SOPM)), que contém uma ferramenta visual para simular SOA, serviços e suas composições. Testes funcionais realizados com este *framework* apresentaram uma margem de erro de 15%, causado por elementos encontrados nos ambientes reais que não estão presentes na simulação.

Desta maneira é importante estabelecer modelos capazes de prever o comportamento de um sistema sob condições específicas. O planejamento de capacidade é um tipo de teste funcional, utilizado para realizar previsões das tendências da utilização de recursos computacionais durante um período de tempo. A previsão da utilização de recursos é muito importante para arquiteturas SOA, principalmente em situações em que o SLA não deve ser violado e os recursos gerenciados com eficiência (Vasudevan e Parthasarathy, 2007). A capacidade para estimar o desempenho de SOA em cenários com diferentes serviços e carga de trabalho é importante para prever comportamentos inesperados (Teixeira et al., 2010).

Schiesser (2004) e Bozkurt et al. (2013) apresentam alguns dos problemas relacionados com a utilização do planejamento de capacidade e experimentos em SOA:

- **Falta de tempo:** Analistas de sistema e de programas estão tão envolvidos em outras atividades tais como instalação, manutenção e desenvolvimento que não tem tempo para realizar o planejamento.
- **Falta de interesse:** O usuário final não tem interesse pelo crescimento da carga de trabalho no futuro, as preocupações concentram-se apenas nas características funcionais do produto.
- **Previsões imprecisas:** Devido a falta de ferramentas, habilidade, experiência e treino, as previsões podem ser imprecisas e gerar prejuízos no futuro.
- **Uso de métodos convencionais:** Analistas podem ser relutantes em tentar usar novos métodos e podem preferir utilizar técnicas tradicionais e conseqüentemente não ter conhecimento de outras ferramentas mais eficazes e eficientes.
- **Custo:** Devido ao alto custo e a mão de obra limitada, as organizações tendem a dar pouca importância ao planejamento de capacidade.

Trabalhos como o apresentado por Smit et al. (2008), propõem um modelo de planejamento de capacidade passo a passo. Neste modelo, o planejamento é feito a partir de detalhes abstraídos do serviço que será testado. Para isso são utilizadas boas práticas e modelos matemáticos para determinar a configuração inicial do experimento e as variações que podem ser geradas neste ambiente. Posteriormente a execução deste serviço é simulada e as configurações são testadas e avaliadas.

Smit e Stroulia (2012) apresentam um *framework* para geração de simulações executáveis (do inglês, *Services-Aware Simulation Framework (SASF)*), baseadas em dados existentes sobre a execução do serviço. Este *framework* permite a interação dos usuários com a simulação, por meio de uma interface gráfica ou por meio de uma interface de programação de aplicativos (do inglês, *Application Programming Interface (API)*). Além disso, um componente é responsável por coletar e exibir as métricas desejadas na avaliação do usuário da simulação.

Smit e Stroulia (2012) também apresentam uma revisão sistemática de *frameworks* para simulação de aplicações em SOA. A funcionalidade mais comum encontrada entre eles é o teste de serviços compostos antes da implantação. A Tabela 3.1 apresenta um resumo das principais características dos *frameworks*. Nota-se que grande parte das ferramentas analisadas, utilizam arquivos para descrição dos serviços para desenvolver os casos de teste, exceto para as ferramentas SOPM e PEESOS. Além disso, a maior parte dos trabalhos validam suas abordagens baseados em provas de conceitos.

Diante dos problemas expostos nesta Seção, o escopo deste trabalho baseia-se no desenvolvimento de um sistema capaz de solucionar os problemas expostos por Meirong e Shaoming (2012), ao instrumentar e executar planejamento de capacidade integrando todos os componentes de uma arquitetura SOA alvo.

Tabela 3.1: Objetivos, modelagem e forma de avaliação. Adaptado de Smit e Stroulia (2012)

	Objetivo	Modelagem	Avaliação
DDSOS	Teste de composição de serviços antes da implantação	PSML-S	Prova de conceito
MaramaMTE	Análise de requisitos não funcionais durante a execução dos serviços	BPMN	Prova de conceito com serviços simples
SOPM	Análise de desempenho e escalabilidade antes da implantação	Visual, Jmeter	Prova de conceito (teste funcional)
Ferramentas Comerciais	Testes de composição e de carga automáticos a partir do arquivo WSDL	WSDL	-
SASF	Planejamento de Capacidade por meio de um modelo virtual para os componentes de uma aplicação.	WSDL, Perfis	Prova de conceito
TESSI	Fornecer geração de casos de teste, execução e gerenciamento	WSDL, BPEL	Prova de conceito
PEESOS	Fornecer geração de casos de teste fim-a-fim e execução baseados em modelos de capacidade em ambientes reais	Modelo de Teste	Prova de conceito

Diferentemente dos trabalhos propostos por Smit et al. (2008); Smit e Stroulia (2012), este trabalho aborda SOA em forma de protótipo e não simulada. Uma interface para configuração passo a passo é proposta neste trabalho, mas diferentemente dos trabalhos apresentados, um modelo com um conjunto de entradas pré-estabelecidos é utilizado para configurar o ambiente de testes e a aplicação. Problemas relacionados com geração de carga de trabalho e o desempenho de arquiteturas SOA, mostrados nos trabalhos de Liu et al. (2007); Vasudevan e Parthasarathy (2007); Teixeira et al. (2010); Tekli et al. (2012) serão abordados de forma similar ao do trabalho de Xing e Yao (2010). Modelos matemáticos para a geração de carga de trabalho sintética serão aplicados pelo sistema. A utilização de clientes reais para realizar as requisições, de acordo com o modelo desejado, possibilita uma avaliação mais precisa da arquitetura alvo.

PEESOS

Neste capítulo é apresentado a ferramenta desenvolvida durante projeto de pesquisa proposto. Na Seção 4.1, é apresentada uma visão geral sobre o projeto. Na Seção 4.2 são descritas a metodologia e as ferramentas utilizadas durante o desenvolvimento. Na Seção 4.3, são abordados detalhadamente os módulos existentes na ferramenta e suas funcionalidades. Por fim, na Seção 4.4, são apresentadas as considerações finais sobre o desenvolvimento do projeto.

4.1 Visão geral

A ferramenta proposta neste trabalho é um protótipo denominado PEESOS que suporta a modelagem de experimentos (do inglês, *Design of Experiments* (DOE)) em SOA. Esta ferramenta também permite a execução e a modelagem do ambiente de execução do sistema alvo. O principal objetivo deste trabalho é a realização de testes fim-a-fim com foco em capacidade de SOA. Para reduzir a complexidade da realização destes testes, etapas como implantação do serviço e geração da carga sintética são abstraídas ao usuário.

Os resultados das execuções são utilizados para prever o desempenho de um sistema, sujeito a diferentes configurações e cargas de trabalho. Além disso, é possível realizar a análise de métricas não funcionais, tais como QoS oferecida aos usuários.

A principal vantagem de um protótipo é a utilização de um ambiente real, que proporciona resultados mais precisos que modelos matemáticos e simulações. Consequentemente, é necessário lidar com os gargalos que não estariam presentes na simulação ou em modelos analíticos. O DOE desta ferramenta é baseado em um conjunto de entradas que descrevem fatores comuns para a realização de testes fim-a-fim em SOA, tais como, quantidade de clientes, quantidade de

Figura 4.1: Fluxo de Funcionamento da Ferramenta

provedores, modelos de distribuições de carga, etc.. Os casos de testes gerados a partir deste conjunto de entradas, são baseados no modelo fatorial completo proposto por Jain (1991), em que a todas as combinações possíveis entre os níveis de todos os fatores são considerados.

A Figura 4.1, apresenta o funcionamento da PEESOS:

1. O usuário configura os parâmetros do experimento: número de provedores, número de clientes, serviço, aplicação cliente, tipo de carga de trabalho, etc.;
2. O serviço é implantado nos provedores;
3. A aplicação cliente é transferida para os clientes;
4. Clientes fazem requisições ao *broker*;
5. *Broker* escolhe provedor apto a atender a requisição;
6. *Broker* redireciona a requisição cliente ao provedor escolhido;
7. *Broker* redireciona a resposta do provedor para o cliente;
8. Os serviços são excluídos dos provedores.

A estrutura geral da ferramenta proposta é apresentada na Figura 4.2. A ferramenta é composta por quatro módulos, que serão descritos com mais detalhes nas próximas seções:

Figura 4.2: Visão geral da proposta dos componentes do planejador de experimentos

- **Monitor:** monitora o estado dos provedores e dos clientes disponíveis para a realização do experimento;
- **Transport:** gerencia a transferência de arquivos e a execução de instruções nos clientes e provedores;
- **Workload:** determina o intervalo entre as requisições dos clientes;
- **PEESOS Core:** gerencia a comunicação entre os outros três módulos e contém as principais classes da ferramenta.

4.2 Metodologias

A metodologia utilizada neste trabalho foi um estudo descritivo, exploratório e experimental, com análise dos dados por meio de uma abordagem quantitativa.

O estudo exploratório contemplou a busca por problemas relacionados ao planejamento de experimentos e de capacidade em arquiteturas SOA. O estudo descritivo buscou soluções existentes na literatura, para os problemas descritos no estudo exploratório e suas possíveis falhas. Por fim, um estudo experimental foi realizado, no qual desenvolveu-se uma ferramenta para instrumentar, orquestrar e executar o planejamento de experimentos de forma distribuída em arquiteturas SOA.

Para análise dos dados optou-se por uma abordagem quantitativa, na qual uma avaliação de desempenho foi realizada, baseada em diferentes modelos de geração de carga sintética em SOA.

Utilizou-se a ferramenta Ganglia para realizar o monitoramento da arquitetura alvo, pois é específica para o monitoramento de sistemas distribuídos. Além disso, utiliza estruturas de dados e algoritmos cuidadosamente projetados para diminuir a sobrecarga no sistema e a alta concorrência. Sua implementação é robusta e está disponível para uma vasta quantidade de sistemas operacionais e processadores (Ganglia, 2013).

Distribuições probabilísticas foram utilizadas para auxiliar a geração de carga de trabalho. A biblioteca *Commons Math* da fundação Apache foi utilizada por ser leve e apresentar componentes voltados para resolução de problemas de maneira simples na linguagem Java (Apache, 2013b).

A ferramenta foi desenvolvida com a tecnologia Java pelo fato de poder ser executada na maioria dos hardwares e plataformas, sendo necessário apenas a configuração de uma máquina virtual Java (do inglês, *Java Virtual Machine* (JVM)). Além disso, possui uma documentação bem definida e um conjunto de bibliotecas que facilitam a integração com outros recursos (Java, 2013). Uma destas bibliotecas é a *Java Security Channel* (JSCH), que possibilita a conexão remota por meio do protocolo *Secure Shell* (SSH), o redirecionamento de portas, a transferência de arquivos, etc. e permite facilmente a integração do módulo inteligente com a arquitetura alvo (JSch, 2013).

4.3 Desenvolvimento da Ferramenta

4.3.1 Arquitetura

No diagrama de pacotes ilustrado na Figura 4.3 é representada a organização arquitetural da ferramenta proposta. Nele, observa-se a organização da ferramenta em quatro pacotes e suas respectivas organizações internas. A seguir, as características de cada pacote são apresentadas e discutidas.

Figura 4.3: Diagrama de pacotes do atual estágio do projeto

4.3.1.1 Monitor

O pacote ***br.usp.monitor*** representado na Figura 4.3, possui um conjunto de métodos cujo objetivo é monitorar a disponibilidade e o estado dos provedores de serviço da arquitetura alvo e dos dispositivos clientes responsáveis por fazer as requisições.

O pacote ***br.usp.monitor.model***, contém as classes com o modelos das informações obtidas dos dispositivos monitorados. Por outro lado informações como, processamento da CPU, quantidade de memória e utilização da CPU são obtidas para os provedores de serviço . Para os clientes monitorados, informações como localização e disponibilidade de execução são recuperadas.

O pacote ***br.usp.monitor.controller***, é responsável por implementar os métodos para manipular as classes disponíveis no pacote ***br.usp.monitor.model***. Para o monitoramento dos provedores de serviço da arquitetura alvo, os métodos de monitoramento foram desenvolvidos em conjunto com a ferramenta Ganglia, uma vez que é o monitor padrão da arquitetura alvo e é capaz de obter uma gama grande de informações. Para o monitoramento dos dispositivos clientes, os métodos desenvolvidos basearam-se na troca de mensagens por *sockets*.

4.3.1.2 Transport

O pacote ***br.usp.transport*** representado na Figura 4.3, realiza instruções remotas nos provedores de serviço e clientes utilizados em um experimento.

O pacote ***br.usp.transport.model*** possui os modelos das classes que realizam as atividades remotas. O pacote ***br.usp.transport.controller*** apresenta os métodos que manipulam os modelos propostos e realizam as atividades remotamente.

Para os provedores de serviço, os modelos utilizados para a comunicação com a ferramenta utilizaram o padrão proposto pela biblioteca JSch (JSch, 2013). Assim, a transferência de arquivos e execução dos mesmos nos provedores de serviço é feita por meio do protocolo SSH.

Figura 4.4: Diagrama de estado do Módulo *Remote Connection* - Servidor

Por outro lado, os modelos utilizados para a comunicação entre clientes e ferramenta basearam-se em *sockets*, já que a comunicação via SSH pode não estar disponível em ambientes heterogêneos. Desta maneira, a comunicação com a ferramenta ocorre por meio de uma aplicação cliente, que é manipulada por troca de mensagens via *socket* e realiza a transferência de arquivos e a execução de instruções.

O diagrama de estados apresentado na Figura 4.4, representa os estados possíveis deste módulo no provedor de serviços. O estado **Cópia** indica que a transferência de arquivos para os servidores será feita. O estado de **Execução** transmite instruções remotas para os dispositivos alvos. Caso alguma atividade inesperada seja detectada o módulo irá para um estado de **Erro**. Caso contrário o processamento estará concluído.

Figura 4.5: Diagrama de estado Módulo *Transport* - Cliente

Na Figura 4.5 é representado o diagrama de estados para aplicação cliente. O estado de **Conexão** é o estado inicial da aplicação, e busca estabelecer uma conexão com a ferramenta. O estado de **Operação** é um estado de espera em que os dados recebidos pela aplicação são analisados. Caso o dado que chegue seja irrelevante, a aplicação continua no estado de **Operação**. Caso contrário, a aplicação poderá ir para dois estados: **Cópia** ou **Execução**.

Um dos estados é o estado **Cópia**, no qual a aplicação fica pronta para receber um arquivo da ferramenta. O outro estado é o de **Execução**, em que uma instrução é recebida e executada pelo dispositivo cliente. Se ambos os estados forem concluídos com sucesso, a aplicação estará apta a receber outro comando. Senão, a aplicação retornará para o estado de **Conexão** e o fluxo de estados será reiniciado.

4.3.1.3 Workload

O pacote *br.usp.workload*, representado na Figura 4.3, estabelece modelos de carga sintética para as requisições. Os intervalos entre as requisições são determinados a partir de modelos de distribuição probabilísticas.

O pacote *br.usp.workload.model* contém os modelos desenvolvidos para a geração da carga sintética com o auxílio dos métodos de distribuição probabilísticas contidos na biblioteca *Commons Math* da Fundação Apache.

O pacote *br.usp.workload.controller* contém os métodos para o cálculo das funções de distribuição. Estas funções são calculadas a partir de métricas específicas a cada uma, tais como:

média para a distribuição exponencial ou valor mínimo e máximo para distribuição uniforme. Para determinar o intervalo entre as requisições utiliza-se os valores de números aleatórios na Função de Distribuição Acumulada (FDA). Para garantir que os intervalos entre os clientes sejam diferentes, seus respectivos identificadores são utilizados como semente da função de geração de números aleatórios.

Figura 4.6: Diagrama de estado do módulo *Workload*

O diagrama de estados apresentado na Figura 4.6 representa os estados que o gerador de carga pode obter. No primeiro estado, **Gerador de Carga**, é determinada qual distribuição probabilística será utilizada. No estado **Função de Distribuição Probabilidade**, a FDA é calculada. No estado **Gerador de Números Aleatórios**, números aleatórios com valores entre 0 e 1 são gerados para cada cliente e aplicados na FDA, que retornará o intervalo entre a requisição. Esta etapa é repetida enquanto o número de intervalos gerados for menor que a quantidade de requisições totais. Caso contrário, a aplicação passa para o estado **Intervalos**, no qual os intervalos de tempo entre as requisições são retornados para a aplicação.

4.3.1.4 Core

O pacote *br.usp.core*, representado na Figura 4.3, é o principal pacote da ferramenta. Dentre suas principais tarefas destacam-se:

- Realizar a comunicação entre o usuário e a ferramenta.
- Realizar a comunicação entre módulos existentes na ferramenta.
- Instrumentar e gerenciar o planejamento de experimento.

A Figura 4.7, apresenta o diagrama de classes simplificado da ferramenta com destaque para as classes do módulo *br.usp.core*. O pacote *br.usp.core.model* contém as classes utilizadas como modelo para realizar a comunicação entre os módulos de toda a aplicação. Dentro deste módulo destaca-se a classe *ExperimentPlanning*, a qual possui o modelo do planejamento de

Figura 4.7: Diagrama de classes simplificado

experimento básico. Esta classe pode ser herdada por outra, tal como a classe *ExperimentPlanningA* que estende este modelo para casos mais específicos.

Figura 4.8: Diagrama de estado do módulo *Core*

O pacote *br.usp.core.controller* contém os métodos que instrumentam e manipulam o experimento. Este módulo é responsável por interagir e orquestrar o funcionamento dos outros módulos da aplicação por meio da classe *ExperimentController*. Analogamente ao pacote *br.usp.core.model*, esta classe pode ser estendida para atender particularidades de um sistema, tal como a classe *ExperimentControllerA*.

Os estados possíveis deste módulo são apresentados na Figura 4.8. O primeiro estado, **Coleta de Dados**, é responsável por interagir com o usuário e obter os valores para os conjuntos

de entrada do planejamento do experimento. Durante esta fase, a ferramenta interage com o módulo 4.3.1.1, que fornece informações sobre os clientes e provedores disponíveis para o experimento.

Algoritmo 4.1: Preparar Experimento

```
Entrada Vetor_de_Fatores, Nivel, Lista_de_Retorno

Inteiro i = 0

Se Vetor_de_Fatores.tamanho > 0
  Enquanto (i < Vetor_de_Fatores[0].Niveis.Tamanho) faca
    Literal nvl = nivel + Vetor_de_Fatores.Niveis[i]
    Vetor_Temporario = Remove_Fator_Do_Vetor(Vetor_de_Fatores[0])
    Preparar_Experimentos(Vetor_Temporario, nvl, Lista_de_Retorno)
    i++
  Fim Enquanto
Se nao
  Enquanto (i < Vetor_de_Fatores[0].Niveis.Tamanho) faca
    String nvl = nivel + Vetor_de_Fatores.Niveis[i]
    Adicione nvl a Lista_de_Retorno
    i++
  Fim Enquanto
```

O estado **Preparação do Ambientes de Testes** define quais experimentos serão realizados baseado no conjunto de entradas configurado pelo usuário de acordo com o Algoritmo 4.1. O ambiente dos provedores de serviço e dos dispositivos clientes são preparados para a execução do experimento, interagindo diretamente com o módulo 4.3.1.2. Nesta etapa, há interação com o módulo 4.3.1.3, já que os intervalos entre requisições de cada cliente também são atribuídos.

O estado **Execução** interage com o módulo 4.3.1.2 para realizar requisições a partir dos clientes vinculados ao experimento. Por fim, o estado **Resultado** disponibiliza um arquivo para o usuário contendo os resultados dos experimentos realizados. Ainda, é válido destacar o pacote *br.usp.core.view*, no qual usuários e ferramenta interagem para estabelecer as configurações do experimento.

Na Figura 4.9 é apresentado o fluxograma do pacote *br.usp.wstester.view*. No estágio **Configuração do Experimento** o usuário atribui valores a um conjunto padrão de entradas. Este conjunto é formado por informações tais como quantidade de provedores, quantidade de clientes, número de replicações, número de requisições, intervalo entre experimentos e tipo da distribuição das requisições.

No estágio **Configurações do Serviço** escolhido, o usuário atribui valores a parâmetros cadastrados para um determinado serviço. Os estágios Clientes Disponíveis e Provedores Disponíveis permitem a escolha dos clientes e provedores aptos a participarem do experimento. A escolha dos provedores fornece as configurações pertinentes a cada provedor disponível, tais como: capacidade de processamento, memória, etc. Já a escolha dos clientes utiliza outros tipos de informações, como por exemplo, localização e tipo do cliente.

Figura 4.9: Fluxograma das etapas de configuração do experimento.

Ao alcançar o estágio **Revisão do Experimento**, o planejamento do experimento é mostrado ao usuário para confirmação de sua realização. Se todas as informações estiverem corretas, o experimento irá para o estágio de **Execução**, em que o planejamento de fato poderá ser iniciado. Ao final do experimento, o identificador do experimento é retornado ao usuário para que ele possa recuperar os resultados após a execução.

4.3.2 Funcionamento

4.3.2.1 Interface

Na Figura 4.10 é apresentada a página inicial da ferramenta, juntamente com as operações disponíveis: *Download*, *Upload* e *Start Experiment Planning*.

Figura 4.10: Página inicial da ferramenta

Ao clicar em *Download* o usuário será redirecionado para uma página com os arquivos de experimentos já executados, conforme ilustrado na Figura 4.11. Os arquivos disponíveis para download possuem um identificador único, disponibilizado durante a inicialização do planejamento de experimentos configurado.

PEESOS	
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems	
Reset	
1983243003.csv	Download
1267067337.csv	Download
1790933942.csv	Download
1901207111.csv	Download
1268140061.csv	Download
1384923886.csv	Download
1704587125.csv	Download
2039552427.csv	Download
1414576813.csv	Download

Figura 4.11: Página de *download* dos arquivos gerados

Ao clicar em *Start Experiment Planning*, a configuração do experimento será inicializada. Na Figura 4.12(a) é representada a página para configuração do ambiente de testes. Os fatores que devem ser preenchidos para a realização do experimento são respectivamente:

PEESOS
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Number of Clients
Available Clients: 9
5 +/-

Number of Providers
Available Hosts: 12
3 +/-

Workload Type (ms)
Exponential
Mean (ms): 9000 +/-

Service
FatorialService.aar +/-

Number of Requests
50

Number of Replications
10

Time Between Experiments (min)
5

Next

(a) 1 nível

PEESOS
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Number of Clients
Available Clients: 9
5 +/-
Available Clients: 9
9 +/-

Number of Providers
Available Hosts: 12
3 +/-
Available Hosts: 12
12 +/-

Workload Type (ms)
Exponential
Mean (ms): 9000.0 +/-

Service
FatorialService.aar +/-

Number of Requests
50

Number of Replications
10

Time Between Experiments (min)
5

Next

(b) 2 níveis

Figura 4.12: Página de configuração do ambiente de testes do experimento com 1 nível 4.12(a) e 2 níveis 4.12(b)

- **Número de clientes (*Number of Clients*):** quantidade de clientes que farão as requisições de serviço aos provedores;
- **Número de Provedores (*Number of Providers*):** quantidade de provedores nos quais o serviço será implantado;
- **Tipo da Distribuição da Carga de Trabalho (*Workload Type*):** forma da distribuição das requisições feitas pelos clientes escolhidos;
- **Serviço (*Service*):** serviço a ser implantado e testado;
- **Quantidade de Requisições (*Number of Requests*):** número de requisições feitas por cliente;
- **Quantidade de Replicações (*Number of Replications*):** número de replicações do experimento;
- **Intervalo entre Experimentos (*Time between Experiments*):** intervalo de espera entre os experimentos.

O botão “+/-” disponível em alguns fatores, permite adicionar níveis a eles, tal como mostrado na Figura 4.12(b) para Clientes e Provedores. O botão *Next* realiza as validações desta página e redireciona o usuário para a próxima etapa, a configuração dos parâmetros do serviço.

The screenshot shows the 'Application Parameters' section of the PEESOS interface. It features a table with four rows: 'Client Type', 'Service Application', 'Service Name', and 'Factorial Number'. Each row has input fields for 'Level 0' and 'Level 1', and a dropdown menu on the right. The 'Client Type' is set to 'gold', 'Service Application' to 'FatorialService', 'Service Name' to 'fatorial', and 'Factorial Number' to '1000'. All dropdown menus are set to '1'. A 'Reset' button is in the top right, and a 'Next' button is at the bottom center.

#	Level 0	Level 1	
Client Type	gold		1 ▾
Service Application	FatorialService		1 ▾
Service Name	fatorial		1 ▾
Factorial Number	1000		1 ▾

(a) 1 nível

This screenshot is identical to the one above, but the 'Factorial Number' dropdown menu is now set to '2', and the 'Level 1' input field for 'Factorial Number' contains the value '10000'.

#	Level 0	Level 1	
Client Type	gold		1 ▾
Service Application	FatorialService		1 ▾
Service Name	fatorial		1 ▾
Factorial Number	1000	10000	2 ▾

(b) 2 níveis

Figura 4.13: Página de configuração dos parâmetros do serviço com 1 nível 4.13(a) e 2 níveis 4.13(b)

Na Figura 4.13 é representada a página para seleção dos parâmetros do experimentos. Estes parâmetros são registrados durante a fase de *upload* do experimento e podem ter valores pré-determinados, tais como os parâmetros deste exemplo, tipo de cliente (*Type of Client*), aplicação do serviço (*Service Application*) e nome do serviço (*Service Name*). Parâmetros que não possuem valores pré-estabelecidos são configurados pelo usuário da aplicação, tal como número do fatorial (*Factorial Number*). Além disso, os parâmetros podem possuir múltiplos níveis ao modificar o valor do último campo de '1' para '2' tal como representado na Figura 4.13(b). O botão *Next* valida os parâmetros configurados nesta página e redireciona o usuário para a próxima etapa, a seleção dos clientes para o experimento.

Na Figura 4.14 é representada a página para seleção dos clientes disponíveis para o experimento. Nesta página são listados os clientes disponíveis por meio da aplicação cliente descrita na Seção 4.3.1.2. As informações obtidas destes clientes são focadas na localização e representam o *Internet Protocol* (IP) local do cliente, o IP da rede e a porta pela qual está conectado à ferramenta. O botão *Next* valida a quantidade de clientes selecionados com a quantidade configurada nos parâmetros do ambiente e redireciona o usuário para a página de seleção dos provedores de serviço.

Na Figura 4.15 é representada a página para seleção dos provedores de serviço da arquitetura alvo disponíveis para o experimento. Nesta página são listados os provedores disponíveis para o

	Local IP	Connected IP	Port
☒	192.168.0.20	143.107.232.253	48935
☒	192.168.0.80	143.107.232.253	45836
☒	192.168.0.30	143.107.232.253	38738
☒	192.168.0.90	143.107.232.253	39610
☒	192.168.0.40	143.107.232.253	52165
☐	192.168.0.70	143.107.232.253	51150
☐	192.168.0.100	143.107.232.253	59842
☐	192.168.0.50	143.107.232.253	34067
☐	192.168.0.110	143.107.232.253	53241

Figura 4.14: Página para seleção dos clientes disponíveis

experimento com suas respectivas informações de conexão e memória. Analogamente a página para seleção de clientes, o botão *Next* valida a quantidade de provedores selecionados com a quantidade configurada nos parâmetros do ambiente e redireciona o usuário para a página de revisão do experimento.

	IP	Port	Memory
☒	192.168.1.141	2041	2057108.0
☒	192.168.1.113	2013	2057108.0
☒	192.168.1.142	2042	2057108.0
☐	192.168.1.143	2043	2057108.0
☐	192.168.1.121	2021	2057108.0
☐	192.168.1.133	2033	2057108.0
☐	192.168.1.132	2032	2057108.0
☐	192.168.1.123	2023	2057108.0
☐	192.168.1.131	2031	2057108.0
☐	192.168.1.122	2022	2057108.0
☐	192.168.1.112	2012	2057108.0
☐	192.168.1.111	2011	2057776.0

Figura 4.15: Página para seleção dos Provedores disponíveis

Na Figura 4.16 é mostrada a página de revisão do experimento. Nesta página são listadas as configurações preenchidas nas etapas de preparação do ambiente, configuração dos parâmetros do serviço, seleção de clientes e seleção de serviços. O botão *Next* inicia a transferência das aplicações clientes para os clientes selecionados e redireciona o usuário para a página de execução.

Na Figura 4.17 é mostrada a página de execução do experimento. Ao clicar em *Execute Experiment*, o experimento será iniciado e a transferência do serviço para os provedores de serviços será realizada. Ao término da configuração do ambiente e antes do início das requisições o *id* do experimento é retornado ao usuário e ele é redirecionado a página inicial, como mostrado na Figura 4.18. Ainda, de acordo com a Figura 4.18, nota-se que é impossível configurar um experimento enquanto outro está em execução.

PEESOS
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Experiment Review

Id	804678533
Number of Clients:	5
Number of Providers:	3
Service Files	FatorialService.aar
Number of Replications	10
Number of Requests	50
Time Between Requests (ms):	5

Selected Clients		Selected Providers	
Connected IP	Port	IP	Port
143.107.232.253	48935	192.168.1.141	2041
143.107.232.253	45836	192.168.1.113	2013
143.107.232.253	38738	192.168.1.142	2042
143.107.232.253	39610		
143.107.232.253	52165		

Application Parameters

#	Level 0	Level 1
Client Type	gold	
Service Application	FatorialService	
Service Name	fatorial	
Factorial Number	1000	10000

Next

Figura 4.16: Página de revisão do experimento

PEESOS
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Execute Experiment

In this phase a full factorial experiment will be executed.
Each client will do N requests.

Execute

Figura 4.17: Página para início da execução do experimento

PEESOS
A Web Tool for Planning and Execution of Experiments in Service Oriented Systems

Locked for experiments. Executing Experiment ID 804678533

Download Files Upload Start Experiment Planning

Do you want collaborate? Download the Client File
Need help?

Figura 4.18: Página com experimento em execução

4.3.2.2 Ferramenta

A Figura 4.19, apresenta o diagrama de sequência da aplicação. Nesta figura é possível notar a integração do módulo *core* com todos os outros módulos. De acordo com os estados apresentados na Figura 4.8 é possível estabelecer a seguinte relação:

- **Coleta de dados:** o estado de **Coleta de Dados** é iniciado na etapa de número um e termina na etapa de número dez. Nesta etapa o módulo *core* interage com o usuário guiando-o durante o planejamento do experimento. Nota-se que entre as etapas de número três e oito, o módulo *core* interage com o módulo monitor. Esta interação oferece ao usuário informações para escolha dos provedores e clientes disponíveis para o experimento;
- **Preparação do Ambiente de Testes:** o estado **Preparação do Ambiente de Testes** é iniciado na etapa de número onze e termina na etapa de número dezessete. Neste estado

Figura 4.19: Diagrama de Sequência da ferramenta

o módulo *core* primeiramente estabelece quais experimentos serão realizados, aplicando o conceito de modelo de experimentos fatorial completo. Logo após, nota-se a interação com o módulo *Workload* para determinar o intervalo de espera entre as requisições a serem feitas. Determinados os intervalos, o módulo *core* interage com o módulo *transport*

para realizar a transferência dos arquivos necessários para a execução do experimento nos clientes.

- **Execução:** o estado de **Execução** corresponde as etapas de número dezoito à vinte e cinco. Durante este estado o módulo *core* interage uma primeira vez com o módulo *transport* para transferência dos serviços. A segunda interação com o módulo *transport* gerencia as requisições para cada cliente do experimento.

4.4 Considerações Finais

Este Capítulo descreveu as ferramentas, tecnologias e metodologias utilizadas no desenvolvimento do projeto de pesquisa. Baseado nos resultados levantados pelo estudo exploratório e descritivo, este Capítulo apresentou parte do estudo experimental proposto, descrevendo detalhadamente os módulos e o funcionamento da ferramenta proposta. No próximo Capítulo serão discutidos os experimentos propostos para um estudo de caso funcional e os resultados obtidos.

Resultados

Este Capítulo discute os resultados obtidos durante a avaliação de desempenho da arquitetura WSARCH. Na Seção 5.1 são apresentadas as configurações dos ambientes utilizados nos testes. A Seção 5.2 descreve as configurações dos experimentos realizados. Na Seção 5.3 são apresentados os resultados obtidos. Na Seção 5.4 é realizada a análise dos resultados. Na Seção 5.5 são relatados os problemas identificados durante a execução dos experimentos. Na Seção 5.6 são apresentadas as considerações finais sobre os resultados analisados.

5.1 Configurações do Ambiente

O ambiente de testes foi preparado com um conjunto de máquinas disponíveis em uma rede externa para realizar as requisições. O *Broker* da WSARCH foi disponibilizado para implantar as aplicações e manipular as requisições. A Figura 5.1 ilustra a organização do ambiente de testes. A ferramenta gerencia as requisições feitas em uma rede externa para o *Broker*, que é responsável por escaloná-las entre os provedores e encaminhar as respostas. A ferramenta também utiliza a arquitetura WSARCH para implantar os serviços nos provedores e opcionalmente registrá-los em um repositório *UDDI*.

O ambiente de testes é composto por uma máquina virtual para executar a ferramenta que está hospedada em uma máquina física. Quatorze clientes estão disponíveis para a realização dos testes. A Tabela 5.1 descreve as configurações das máquinas utilizadas.

A arquitetura WSARCH é composta por um *Broker*, um servidor de Log, doze provedores virtualizados (três provedores virtuais por máquina física) e doze repositórios *UDDI* virtuali-

Figura 5.1: Ambiente de teste

Tabela 5.1: Configurações dos Clientes e da Ferramenta

	Clientes	Máquina Física	Ferramenta
Processador	Processador Intel Core 2 Quad 6600		2 Processadores Virtuais
Memória	8 GB RAM DDR3		2GB RAM
Disco Rígido	HD 160GB Seagate Sata II 7200RPM		50GB
Sistema Operacional	Linux Ubuntu Server 12.04 64 Bits LTS		Linux Ubuntu Server 12.04 64 Bits LTS
Aplicações	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0	Qemu / KVM	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0

zados (três UDDIs virtuais por máquina física). A Tabela 5.2 descreve as configurações dos componentes da arquitetura WSARCH.

5.2 Configurações dos Experimentos

O serviço utilizado para o desenvolvimento dos experimentos foi o cálculo do fatorial de um número, que possui características *CPU Bound*. O desenvolvimento deste serviço foi feito de forma iterativa a partir da Equação $n! = \prod_{k=1}^n k, \forall k \in \mathbb{R}$, em que n representa o número fatorial a ser calculado iniciando a partir de k .

Na Tabela 5.3 são representados os fatores e níveis do conjunto de entradas utilizados para realização dos experimentos.

Tabela 5.2: Configurações dos componentes da arquitetura WSARCH

Ambiente WSARCH			
	Broker e Servidor de Log	Máquina Física	Máquina Virtual (Provedores e Repositório)
Processador	Processador AMD Vishera 4.2 Ghz		2 Processadores Virtuais
Memória	32 GB RAM DDR3 Corsair Vegeance		2GB RAM
Disco Rígido	HD 2TB Seagate Sata III 7200RPM		50GB
Sistema Operacional	Linux Ubuntu Server 12.04 64 Bits LTS		Linux Ubuntu Server 12.04 64 Bits LTS
Aplicações	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0 MySQL 5.4	Qemu/KVM	Java JDK 1.7 Apache Axis2 1.5 Apache Tomcat 7.0

Tabela 5.3: Fatores e níveis do experimento

Fatores	Níveis
nº de Clientes	7 e 14
nº de Provedores	1 e 6
Carga de Trabalho	Exponencial e Uniforme
Fatorial	1.000, 8.000 e 16.000

A Tabela 5.4 apresenta a configuração dos experimentos. Um experimento fatorial completo foi realizado a partir dos valores dos níveis de três fatores disponíveis no conjunto de entrada: número de clientes (7 e 14), número de provedores (1, 6) e cargas de trabalho (uniforme e exponencial). Utilizando diferentes números de clientes e tipo de carga de trabalho é possível observar o comportamento do sistema sobre determinadas circunstâncias. A variação dos provedores de serviço contribui para avaliar o sistema funcionando com diferentes quantidades de recursos disponíveis. Além disso, um quarto fator relacionado ao fatorial (1.000, 8.000 e 16.000) do serviço foi adicionado, possibilitando utilizar os recursos disponíveis de várias formas.

Cada experimento representado na Tabela 5.4 realizou um total de 700 requisições aos provedores de serviço e foram replicados dez vezes. A replicação dos experimentos foi realizada com o intuito de assegurar um intervalo de confiança de 95%.

A carga de trabalho exponencial foi gerada baseada na Equação 5.1 (Ross, 2009),

$$f(x, \lambda) = \begin{cases} \lambda e^{-\lambda x} & \text{para } x \geq 0, \\ 0 & \text{para } x < 0. \end{cases} \quad (5.1a)$$

$$(5.1b)$$

Tabela 5.4: Modelo dos Experimentos

nº de Provedores	nº Clientes	Carga de Trabalho	Fatorial
1	7	Exponencial	1.000
1	7	Exponencial	8.000
1	7	Exponencial	16.000
1	7	Uniforme	1.000
1	7	Uniforme	8.000
1	7	Uniforme	16.000
1	14	Exponencial	1.000
1	14	Exponencial	8.000
1	14	Exponencial	16.000
1	14	Uniforme	1.000
1	14	Uniforme	8.000
1	14	Uniforme	16.000
6	7	Exponencial	1.000
6	7	Exponencial	8.000
6	7	Exponencial	16.000
6	7	Uniforme	1.000
6	7	Uniforme	8.000
6	7	Uniforme	16.000
6	14	Exponencial	1.000
6	14	Exponencial	8.000
6	14	Exponencial	16.000
6	14	Uniforme	1.000
6	14	Uniforme	8.000
6	14	Uniforme	16.000

em que a média da distribuição exponencial é representada por $\lambda = 9000$, $\forall x \in [0, \infty)$. A carga de trabalho uniforme teve sua geração baseada na Equação 5.2 (Casella e Berger, 2002),

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{para } a \leq x \leq b, \\ 0 & \text{para } x < a \text{ ou } x > b. \end{cases} \quad (5.2a)$$

$$(5.2b)$$

em que o valor mínimo é representado por $a = 1.000$ e o valor máximo representado por $b = 21.000$. Os intervalos entre as requisições foram determinados a partir de valores aleatórios da função de distribuição acumulada das Equações 5.1 e 5.2. Os valores de λ na Equação 5.1 e a e b na Equação 5.2 foram definidos de forma a obter-se intervalos máximos de espera próximos entre as equações.

5.3 Resultados

No gráfico da Figura 5.2 são apresentados os intervalos de confiança juntamente com a média do tempo médio necessário para o atendimento das requisições. Nota-se neste gráfico que a utilização de apenas um provedor para a execução do serviço apresentou tempos menores na maioria dos casos, independentemente da quantidade de clientes, da distribuição e do fatorial a ser calculado. O único cenário no qual há tempos estatisticamente equivalentes para ambas as quantidades de provedores é quando o fatorial de 16.000 é calculado com uma distribuição exponencial realizada a partir de quatorze clientes.

Figura 5.2: Gráfico com Intervalo de confiança e média do tempo total das requisições

O tempo necessário para o atendimento das requisições é diretamente proporcional a quantidade de clientes utilizados e ao valor do fatorial calculado. Ainda, percebe-se que a distribuição exponencial apresentou tempos maiores ou estatisticamente equivalentes a distribuição uniforme.

No gráfico da Figura 5.3 é apresentado o comportamento médio para o atendimento das requisições por meio do gráfico Boxplot. Primeiramente, nota-se que as medianas (segundo quartil) utilizadas para a geração dos gráficos são maiores quando utiliza-se seis provedores para o tratamento das requisições. Além disso, é possível observar as requisições que apresentaram tempos atípicos ou *outliers* superiores e inferiores.

Os *outliers* inferiores ocorreram em menor quantidade e são observados apenas em alguns casos quando apenas um provedor é utilizado. Geralmente, isso ocorre no momento em que o

Figura 5.3: Boxplot do tempo total das requisições

servidor encontra-se ocioso e apto a responder a requisição. Por outro lado, os *outliers* superiores são encontrados em todos os casos estudados. Eles podem ser registrados durante o período de *warm up* (tempo para inicialização das máquinas virtuais Java e de *cache*) do sistema, possíveis rajadas nas distribuições ou sobrecarga dos provedores.

Nas Figuras 5.4 e 5.5, são apresentados os gráficos de dispersão em pontos contendo o tempo médio para o atendimento de cada uma das 700 requisições realizadas nos experimentos. Por meio destes gráficos observa-se o período de *warm up* no início de cada experimento, em que as requisições apresentam tempos maiores que a média da distribuição, o qual é mais acentuado para as distribuições exponenciais com a utilização de quatorze clientes. Além disso, estes gráficos possibilitam a visualização dos instantes em que ocorrem os *outliers* descritos na Figura 5.3.

Na Figura 5.6 é representado o gráfico do intervalo de confiança com o tempo médio necessário para escolher o provedor de serviço que irá responder a requisição. Neste gráfico é possível observar que independentemente da quantidade de clientes, da distribuição e do fatorial calculado, a utilização de seis provedores aumentou o tempo necessário para encontrar o provedor de serviços apto a responder a requisição. Este comportamento é observado pois há o aumento de registros no repositório de serviços, que neste caso é o UDDI. Ainda, o seletor padrão da arquitetura WSARCH atualiza constantemente os índices de QoS dos serviços presentes no UDDI. Estes índices são utilizados pelo *Broker* para realizar a seleção dos provedores de serviço. A atualização constante de seus valores sobrecarrega a comunicação com o repo-

(a) 7 Clientes e 1 Provedor

(b) 7 Clientes e 6 Provedores

(c) 14 Clientes e 1 Provedor

(d) 14 Clientes e 6 Provedores

Figura 5.4: Gráficos de dispersão dos experimentos com distribuição exponencial para os fatoriais de 1.000, 8.000 e 16.000

(a) 7 Clientes e 1 Provedor

(b) 7 Clientes e 6 Provedores

(c) 14 Clientes e 1 Provedor

(d) 14 Clientes e 6 Provedores

Figura 5.5: Gráficos de dispersão dos experimentos com distribuição uniforme para os fatoriais de 1.000, 8.000 e 16.000

Figura 5.6: Gráfico com intervalo de confiança e média do tempo de busca do provedor de serviços

sitório de serviços e consequentemente o tempo de busca pelo serviço torna-se proporcional a quantidade de serviços nele registrados.

Figura 5.7: Gráfico com intervalo de confiança e média do tempo de processamento do Broker

Na Figura 5.7 é representado o gráfico do intervalo de confiança contendo o tempo médio de processamento do *Broker*. Nota-se um comportamento similar ao da Figura 5.7, em que

para a maior parte dos casos analisados, há um melhor desempenho independentemente da distribuição, com a utilização de um único provedor. Os casos em que isso ocorre são os cálculos dos fatoriais de 1.000 e 8.000 para qualquer quantidade de clientes e para o cálculo do fatorial de 16.000 com a utilização de sete clientes.

Entretanto, é válido destacar o comportamento do fatorial de 16.000 ao utilizar quatorze clientes. Para a distribuição exponencial, o *Broker* passa a ser mais eficiente com 6 provedores e para a distribuição uniforme, estatisticamente equivalentes. Este comportamento é justificado pelo fato do cálculo do fatorial de 16.000 exigir mais processamento e tempo dos provedores de serviço, aumentando o tempo de espera para o atendimento das requisições em ambos os cenários.

Na Figura 5.8 é representado o gráfico do tempo médio de atendimento das requisições sem considerar o tempo de busca e seleção dos provedores de serviço. Nota-se que ao desprezar o tempo necessário para a busca e seleção de serviços no repositório, os tempos médios de resposta diminuem ao utilizar seis provedores de serviço, contrariando assim parte dos resultados discutidos na Figura 5.2. Este contraste ocorre, pois como mostrado na Figura 5.6, o tempo de busca pelo serviço é proporcional a quantidade de provedores utilizados.

Figura 5.8: Gráfico com intervalo de confiança e média do tempo do cliente sem o tempo de busca e seleção do serviço

Os resultados divergentes ocorrem na distribuição exponencial com quatorze clientes para o cálculo do fatorial de 8.000, o qual passou a ser estatisticamente equivalente, e também para o cálculo do fatorial de 16.000 que apresentou redução em seus valores médios. O cálculo do fatorial de 16.000 também teve seu tempo médio reduzido para a distribuição uniforme, sobrepondo assim os intervalos de confiança dos testes realizados.

5.4 Análise dos Resultados

Os resultados obtidos por meio dos experimentos realizados pela ferramenta PEESOS possibilitaram investigar o comportamento da arquitetura WSARCH sob diversas condições.

Ao contrário do esperado, a quantidade de provedores disponíveis não garante um melhor desempenho da arquitetura alvo. Por meio dos resultados discutidos na Seção 5.3, notou-se que o desempenho da arquitetura é relacionado principalmente a dois fatores: a quantidade de clientes que utilizam o sistema e a sobrecarga dos serviços utilizados. Verifica-se por meio da Figura 5.9, que a utilização de fatoriais que exigem pouco processamento dos provedores e que a utilização de poucos clientes não são capazes de sobrecarregar os provedores disponíveis na arquitetura alvo. Além disso, a distribuição destas requisições também influencia o desempenho. Por apresentar maior ocorrência de rajadas, os resultados com o uso da distribuição exponencial foram piores que aqueles obtidos com o uso da distribuição uniforme.

Figura 5.9: Gráficos em linha com os tempos médios de resposta das requisições para as distribuições Exponencial (5.9(a)) e Uniforme (5.9(b))

A utilização de mais provedores provou ser adequada em situações que exigem processamento e estressam o sistema. Por meio do gráfico da Figura 5.10, que representa o tempo médio de atendimento das requisições pelo *Broker*, observa-se que sob uma alta demanda de requisições (na utilização de 14 clientes) distribuídas exponencialmente e com exigência de alto processamento (fatorial 16.000), o uso de mais de um provedor apresentou resultados mais satisfatórios. Por outro lado, ao utilizar a distribuição uniforme, nota-se que os tempos são equivalentes, uma vez que há menos probabilidade de ocorrer rajadas.

Também, é válido destacar de acordo com a Seção 5.3, que a quantidade de provedores utilizados influencia diretamente no tempo de busca e seleção do serviço. No caso da arquitetura WSARCH, estes serviços são registrados num repositório UDDI, o qual além das informações

Figura 5.10: Gráficos em linha com os tempos médios de atendimento das requisições para as distribuições Exponencial (5.10(a)) e Uniforme (5.10(b)) no *Broker*

sobre a localização dos serviços possui informações de desempenho dos provedores. Estas informações, são atualizadas de tempos em tempos, sobrecarregando o repositório de acordo com a quantidade de serviços nele cadastrados e consequentemente aumentando o tempo final de resposta da requisição cliente.

5.5 Problemas Identificados

Alguns problemas puderam ser identificados e corrigidos na arquitetura WSARCH por meio da análise dos resultados (logs) dos testes funcionais, mais precisamente nos provedores de serviço e no módulo *Logserver*. É válido destacar que estes problemas foram detectados justamente pela aplicação do planejamento de capacidade, o qual procurou utilizar a arquitetura WSARCH de forma intensiva.

Dois problemas puderam ser identificados nos provedores de serviço. O primeiro aborda o *log* do servidor Apache Tomcat 7.0, o qual estava crescendo demasiadamente até ocupar todo o disco rígido do provedor e travá-lo. Para resolver este problema utilizou-se a biblioteca de log de dados Log4J 2 (Apache, 2013a), a qual possibilitou dividir o arquivo de *log* em dez arquivos menores com tamanho máximo de 1 Mbyte cada.

O outro problema identificado nos provedores de serviço explorou a limitação de arquivos abertos simultaneamente no sistema operacional. Por padrão, o sistema operacional instalado nos provedores pode abrir no máximo 1024 arquivos simultaneamente e, ao ultrapassar esse limite, as requisições não eram mais atendidas. Como solução, este limite foi alterado no arquivo do sistema operacional “/etc/security/limits.conf” para permitir a abertura de 65.555 arquivos simultâneos.

O módulo *Logserver* apresentou degradação em seu desempenho proporcional a quantidade de registros inseridos no banco de dados MySQL. Este problema ocorreu por utilizar as configurações padrões do banco de dados, o qual é limitado para servidores domésticos. A resolução deste problema baseou-se na otimização dos parâmetros do arquivo de configuração “/etc/mysql/my.cnf”.

5.6 Considerações Finais

Neste Capítulo foram apresentadas as configurações de ambiente, o planejamento de experimento, os resultados dos testes funcionais e os problemas identificados durante a execução dos experimentos.

A análise dos resultados fornecidos pelo *log* da arquitetura WSARCH permitiu investigar o comportamento dos componentes desta arquitetura sob determinadas condições. Os resultados obtidos mostraram que o tempo de resposta da arquitetura é influenciado por diversos fatores tais como quantidade de clientes, tipo da distribuição e sobrecarga gerada na arquitetura. Além disso, a utilização de mais provedores de serviço não garante a obtenção de tempos de resposta menores, já que outros fatores devem ser considerados, tais como o tempo de busca e seleção dos provedores de serviços utilizados.

No próximo Capítulo serão apresentadas as conclusões sobre o projeto de pesquisa desenvolvido, suas limitações e trabalhos futuros.

Conclusão

6.1 Considerações Iniciais

A proposta da criação de uma ferramenta para a realização de planejamento de capacidade em SOA surge da necessidade de avaliar seu desempenho sob condições pré-estabelecidas. Um planejamento de capacidade bem feito auxilia a previsão da utilização máxima dos recursos de um sistema e também de seus requisitos não funcionais. Nesse contexto, este capítulo descreve as principais conclusões e trabalhos futuros provenientes da modelagem e prototipação de mecanismos para a automatização do planejamento e execução de experimentos em sistemas orientados a serviços, nos quais baseou-se o desenvolvimento da ferramenta PEESOS - Planejamento e Execução de Experimentos em Sistemas Orientados a Serviços.

Na Seção 6.2 são discutidas as conclusões e as contribuições para o estado da arte obtidas com o desenvolvimento do projeto de mestrado. Na Seção 6.3 são apresentadas as produções científicas oriundas do trabalho proposto. Por fim, na Seção 6.4 são detalhados alguns trabalhos futuros relacionados ao tema deste projeto.

6.2 Conclusões e Contribuições

A principal contribuição deste trabalho constituiu-se no desenvolvimento de uma ferramenta denominada PEESOS, a qual é composta por um conjunto de mecanismos para automatizar o planejamento e execução de experimentos, com um estudo de caso voltado para o uso de *Web Services* em uma SOA denominada WSARCH.

De acordo com a análise dos trabalhos disponíveis na literatura da literatura apresentada no Capítulo 3 desta dissertação, muitos trabalhos utilizam arquivos de descrição de serviço para modelar os casos de teste, tais como WSDL e BPEL. Porém ao realizar esta modelagem automaticamente, muitos testes são desnecessários e tornam esta atividade dispendiosa. Diferentemente dos trabalhos existentes, este trabalho apresentou um mecanismo para modelagem de experimentos baseado em um conjunto de entradas, as quais são comuns a SOA. A partir do conjunto de valores atribuídos a estas entradas, um experimento fatorial completo é realizado, no qual todos os valores definidos no conjunto de entrada são combinados.

Além disso, todos os trabalhos analisados no Capítulo 3 implementam sua carga de trabalho por meio de simulação ou modelos matemáticos. Entretanto, esta forma de execução não considera fatores que são encontrados em ambientes reais, ocasionando erros nos resultados obtidos. Para maior precisão dos resultados obtidos, este trabalho apresentou um mecanismo para a execução da carga de trabalho por meio de um ambiente colaborativo real.

Nos testes funcionais executados realizou-se um planejamento de capacidade para averiguar o comportamento da arquitetura WSARCH condicionada a diferentes formas de uso. Os resultados mostraram que os problemas de desempenho encontrados na arquitetura WSARCH estão diretamente relacionados ao seu repositório de serviços, que é sobrecarregado constantemente com a taxa de atualização dos atributos de QoS de cada provedor.

Os resultados também mostraram que o tempo final da requisição pode ser influenciado por fatores externos tais como o tráfego da rede, roteamento, e configurações de *software* uma vez que o tempo total de atendimento da requisição é influenciado por eles.

Por fim, a ferramenta provou sua real contribuição durante a execução do planejamento de capacidade, o qual permitiu também identificar e solucionar problemas relacionados a configurações do ambiente da arquitetura nos provedores de serviço e no módulo *LogServer*. É válido destacar que estes problemas não seriam identificados apenas pela realização de testes unitários ou de integração, uma vez que costumam ocorrer após um longo período de utilização da arquitetura WSARCH.

6.3 Produções Científicas

Os resultados obtidos a partir do desenvolvimento deste trabalho caracterizaram a produção científica a seguir:

6.3.1 Artigos aceitos para publicação

- NUNES, L. H.; NAKAMURA, L. H. V.; KUEHNE, B. T.; OLIVEIRA, E. M.; LIBARDI, R. M. O.; ADAMI, L. J.; ESTRELLA, J. C; REIFF-MARGANIEC, S.. *PEESOS: A Web Tool for Planning and Execution of Experiments in Service Oriented Systems*. 21th IEEE

International Conference on Web Services (ICWS), June 27 - July 2, 2014, Alaska, USA. (*Conferência Qualis A1 - Full Paper*)

- NUNES, L. H.; NAKAMURA, L. H. V.; Vieira, H. F.; OLIVEIRA, E. M.; LIBARDI, R. M. O.; ADAMI, L. J.; ESTRELLA, J. C; REIFF-MARGANIEC, S.. *A Study Case of Restful Frameworks in Raspberry Pi: A Performance and Energy Overview*. 21th IEEE International Conference on Web Services (ICWS), June 27 - July 2, 2014, Alaska, USA. (*Conferência Qualis A1 - Extended Abstract*)
- NAKAMURA, L. H. V.; PRADO, P.; LIBARDI, R. M. O.; NUNES, L. H.; ESTRELLA, J. C; SANTANA, R. H. C.; SANTANA, M. J.; REIFF-MARGANIEC, S.. *Fast Selection of Web Services with QoS using a Distributed Parallel Semantic Approach*. 21th IEEE International Conference on Web Services (ICWS), June 27 - July 2, 2014, Alaska, USA. (*Conferência Qualis A1 - Short Paper*)

6.4 Trabalhos Futuros

Este trabalho é apenas o primeiro protótipo de uma ferramenta, no qual inúmeras melhorias podem ser realizadas para transformá-lo em uma ferramenta mais abrangente. Dentre elas destacam-se:

- **Mecanismos de escalabilidade:** Embora a ferramenta seja capaz de gerenciar o planejamento e execução de experimentos em um ambiente colaborativo, apenas um experimento pode ser executado por vez. Mecanismos que visam a escalabilidade da ferramenta devem ser desenvolvidos e aplicados ao protótipo. Além disso, o mecanismos para gerenciar milhares de conexões em tempo hábil devem ser considerados nesta etapa;
- **Mecanismos de tolerância a falhas:** Devido a natureza distribuída do trabalho proposto, problemas de *hardware*, *software* e infraestrutura podem ocorrer. Mecanismos de tolerância a falhas devem ser aplicados ao protótipo para garantir a integridade dos experimentos;
- **Usabilidade:** A aplicação de melhorias na interface do projeto, seguindo técnicas de interface humano-computador, devem ser realizadas juntamente com um experimento qualitativo para averiguar a usabilidade da ferramenta;
- **Testes de capacidade:** A utilização do protótipo proposto deve ser aplicado em outras SOA para a realização de testes de capacidade e, dessa maneira, averiguar a adaptabilidade da ferramenta em outros cenários.
- **Testes em outros ambientes:** Atualmente a ferramenta é aplicada apenas para SOA, porém outros módulos podem ser desenvolvidos e acoplados à ferramenta para a realização de testes de capacidade, como por exemplo, em ambiente de *cloud computing*.

- **Aplicação de técnicas de aprendizado de máquina:** Técnicas de aprendizado de máquina podem ser aplicadas a ferramenta para auxiliar o desenvolvedor a realizar o planejamento do experimento.

Referências Bibliográficas

ALSHAHWAN, F.; MOESSNER, K.; CARREZ, F. Distributing resource intensive mobile web services. In: *Innovations in Information Technology (IIT), 2011 International Conference on*, 2011, p. 41–46.

APACHE, S. F. Apache log4j 2 v.2.0-rc1 - user's guide. Disponível em <http://logging.apache.org/log4j/2.x/log4j-users-guide.pdf>, Último acesso em: 12/03/2014, 2013a.

APACHE, S. F. Commons math: The apache commons mathematics library. Disponível em <http://commons.apache.org/proper/commons-math/>, Último acesso em: 08/11/2013, 2013b.

BOZKURT, M.; HARMAN, M.; HASSOUN, Y. Testing and verification in service-oriented architecture: A survey. *Software Testing Verification and Reliability*, v. 23, n. 4, p. 261–313, 2013.

Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84877618825&partnerID=40&md5=97999ca1d34c4f0067b79861d238dd4a>

BREBNER, P. Service-oriented performance modeling the mule enterprise service bus (esb) loan broker application. *Conference Proceedings of the EUROMICRO*, p. 404–411, 2009.

Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-74549201609&partnerID=40&md5=9b8c85ff0649a0a9cf3ef4f3ef21490a>

CANFORA, G.; PENTA, M. Service-oriented architectures testing: A survey. In: LUCIA, A.; FERRUCCI, F., eds. *Software Engineering*, v. 5413 de *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, p. 78–105, 2009.

Disponível em: http://dx.doi.org/10.1007/978-3-540-95888-8_4

- CARDELLINI, V.; CASALICCHIO, E.; BRANCO, K. R. L. J. C.; ESTRELLA, J. C.; J., M. F. *Performance and dependability in service computing: Concepts, techniques and research directions*. IGI Global; 1 edition, 2011.
- CASELLA, G.; BERGER, R. *Statistical inference*. Duxbury advanced series in statistics and decision sciences. Thomson Learning, 2002.
Disponível em: http://books.google.com.br/books?id=0x_vAAAAMAAJ
- CUI, L.; SHANG, Z.; SHI, Y. A transaction management model based on compensation planning graph for web services composition. In: *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, p. 275–282.
- ERL, T. *Soa: principles of service design*, v. 1. Prentice Hall Upper Saddle River, 2008.
- ESTRELLA, J.; TOYOHARA, R.; KUEHNE, B.; TAVARES, T.; SANTANA, R.; SANTANA, M.; BRUSCHI, S. A performance evaluation for a qos-aware service oriented architecture. In: *Services (SERVICES-1), 2010 6th World Congress on*, 2010, p. 260–267.
- ESTRELLA, J. C.; SANTANA, R. H. C.; SANTANA, M. J. *Wsarch: An architecture for web services provisioning with qos support: Performance challenges*. VDM Verlag Dr. Müller, 2011.
- GANGLIA Ganglia monitor system. Disponível em <http://ganglia.info/>, Último acesso em: 15/01/2013, 2013.
- GRUNDY, J.; HOSKING, J.; LI, L.; LIU, N. Performance engineering of service compositions. In: *Proceedings of the 2006 International Workshop on Service-oriented Software Engineering*, SOSE '06, New York, NY, USA: ACM, 2006, p. 26–32 (SOSE '06,).
Disponível em: <http://doi.acm.org/10.1145/1138486.1138493>
- GU, Y.; GE, Y. Search-based performance testing of applications with composite services. In: *Web Information Systems and Mining, 2009. WISM 2009. International Conference on*, 2009, p. 320–324.
- HUANG, X.; WANG, W.; ZHANG, W.; WEI, J.; HUANG, T. An automatic performance modeling approach to capacity planning for multi-service web applications. In: *Quality Software (QSIC), 2011 11th International Conference on*, 2011, p. 67–75.
- HUQQANI, A.; LI, X.; BERAN, P.; SCHIKUTA, E. N2cloud: Cloud based neural network simulation application. In: *Neural Networks (IJCNN), The 2010 International Joint Conference on*, 2010, p. 1–5.
- JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. Wiley professional computing. Wiley,

1991.
Disponível em: <http://books.google.com.br/books?id=HetQAAAAMAAJ>
- JAVA Java. Disponível em <http://www.oracle.com/br/technologies/java/features/index.html>, Último acesso em: 08/02/2013, 2013.
- JSCH Java security channel. Disponível em <http://www.jcraft.com/jsch/>, Último acesso em: 15/01/2013, 2013.
- KALAMEGAM, P.; GODANDAPANI, Z. A survey on testing soa built using web services. *International Journal of Software Engineering and its Applications*, v. 6, n. 4, p. 91–104, 2012.
Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84874965699&partnerID=40&md5=169310728ffe11e9ed486f33bd8a5758>
- KANAGASUNDARAM, R.; MAJUMDAR, S.; ZAMAN, M.; SRIVASTAVA, P.; GOEL, N. Exposing resources as web services: A performance oriented approach. In: *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2012 International Symposium on*, 2012, p. 1–10.
- KUEHNE, B.; ESTRELLA, J.; PEIXOTO, M.; TAVARES, T.; SANTANA, R.; SANTANA, M. Dynamic web service composition middleware: A new approach for qos guarantees. In: *Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on*, 2010, p. 174–177.
- LI, H.; ZHU, Q.; OUYANG, Y. Non-cooperative game based qos-aware web services composition approach for concurrent tasks. In: *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, p. 444–451.
- LIU, Y.; GORTON, I.; ZHU, L. Performance prediction of service-oriented applications based on an enterprise service bus. In: *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, 2007, p. 327–334.
- MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The ganglia distributed monitoring system: Design, implementation and experience. 2003, p. 2004.
- MAYER, P.; LÜBKE, D. Towards a bpel unit testing framework. In: *Proceedings of the 2006 Workshop on Testing, Analysis, and Verification of Web Services and Applications, TAV-WEB '06*, New York, NY, USA: ACM, 2006, p. 33–42 (TAV-WEB '06,).
Disponível em: <http://doi.acm.org/10.1145/1145718.1145723>

- MEIRONG, D.; SHAOMING, C. Design and realization of soa based management system for open experiment lab. In: *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*, 2012, p. 3182 –3185.
- MOHAMED, K.; WIJESEKERA, D. A lightweight framework for web services implementations on mobile devices. In: *Mobile Services (MS), 2012 IEEE First International Conference on*, 2012, p. 64 –71.
- NETWORKS, C. Cloudport. Disponível em <http://www.crosschecknet.com/products/cloudport.php>, Último acesso em: 13/02/2014, 2014.
- PAPAZOGLU, M. Service-oriented computing: concepts, characteristics and directions. In: *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, 2003, p. 3–12.
- PAPAZOGLU, M.; TRAVERSO, P.; DUSTDAR, S.; LEYMANN, F. Service-oriented computing: State of the art and research challenges. *Computer*, v. 40, n. 11, p. 38–45, 2007.
- PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. Restful web services vs. "big" web services: making the right architectural decision. In: *Proceedings of the 17th international conference on World Wide Web, WWW '08*, New York, NY, USA: ACM, 2008, p. 805–814 (WWW '08,).
Disponível em: <http://doi.acm.org/10.1145/1367497.1367606>
- PETROVA-ANTONOVA, D.; ILIEVA, S.; STOYANOVA, V. Tessi: A web service testing tool: Demonstration paper. In: *Research Challenges in Information Science (RCIS), 2013 IEEE Seventh International Conference on*, 2013, p. 1–2.
- PWC, T. F. Consumerization of apis: Scaling integrations. Disponível em <http://www.pwc.com/us/en/technology-forecast/2012/issue2/features/feature-consumerization-apis.jhtml>, Último acesso em: 28/02/2013, 2013.
- ROSEN, M.; LUBLINSKY, B.; SMITH, K.; BALCER, M. *Applied soa: Service-oriented architecture and design strategies*. Wiley, 2008.
- ROSS, S. *Introduction to probability and statistics for engineers and scientists*. Elsevier Science, 2009.
Disponível em: http://books.google.com.br/books?id=mXP_UEiUo9wC
- SCHIESSER, R. Why capacity planning is seldom done well. Disponível em <http://www.informit.com/articles/article.aspx?p=169584>, Último acesso em: 16/02/2013, 2004.

- SMARTBEAR Soapui. Disponível em <http://www.soapui.org/>, Último acesso em: 13/02/2014, 2014.
- SMIT, M.; NISBET, A.; STROULIA, E.; EDGAR, A.; ISZLAI, G.; LITOIU, M. Capacity planning for service-oriented architectures. In: *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, CASCON '08, New York, NY, USA: ACM, 2008, p. 11:144–11:156 (CASCON '08,). Disponível em: <http://doi.acm.org/10.1145/1463788.1463803>
- SMIT, M.; STROULIA, E. Simulating service-oriented systems: A survey and the services-aware simulation framework. *IEEE Transactions on Services Computing*, v. 99, n. PrePrints, 2012.
- SNEED, H.; HUANG, S. Wsdlttest - a tool for testing web services. In: *Web Site Evolution, 2006. WSE '06. Eighth IEEE International Symposium on*, 2006, p. 14–21.
- TAVARES, T. C.; SANTANA, R. H. C.; ESTRELLA, J. C.; SANTANA, M. J. Caracterização dos dados trocados entre serviços web e provedores de serviços. In: *Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web*, WebMedia '08, New York, NY, USA: ACM, 2008, p. 65–68 (WebMedia '08,). Disponível em: <http://doi.acm.org/10.1145/1809980.1809998>
- TEIXEIRA, M.; LIMA, R.; OLIVEIRA, C.; MACIEL, P. A stochastic model for performance evaluation and bottleneck discovering on soa-based systems. In: *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, 2010, p. 358 –365.
- TEKLI, J.; DAMIANI, E.; CHBEIR, R.; GIANINI, G. Soap processing performance and enhancement. *Services Computing, IEEE Transactions on*, v. 5, n. 3, p. 387 –403, 2012.
- TILLEY, S.; WONG, K.; SMITH, S. Teaching and using service-oriented architecture (soa) in an academic environment. In: *Systems Conference, 2008 2nd Annual IEEE*, 2008, p. 1 –4.
- TSAI, W.; FAN, C.; CHEN, Y.; PAUL, R. Ddsos: a dynamic distributed service-oriented simulation framework. In: *Simulation Symposium, 2006. 39th Annual*, 2006, p. 8 pp.–.
- TSAI, W.-T.; WEI, X.; CAO, Z.; PAUL, R.; CHEN, Y.; XU, J. Process specification and modeling language for service-oriented software development. In: *Future Trends of Distributed Computing Systems, 2007. FTDCS '07. 11th IEEE International Workshop on*, 2007, p. 181–188.
- VASUDEVAN, N.; PARTHASARATHY, G. Comparative analysis of neural network techniques vs statistical methods in capacity planning. In: *Software Engineering Research, Management Applications, 2007. SERA 2007. 5th ACIS International Conference on*, 2007, p. 799 –806.

- W3C, T. W. W. W. C. Web services activity statement. Disponível em <http://www.w3.org/2002/ws/Activity>, Último acesso em: 15/01/2014, 2013.
- XING, Y.; YAO, E. Remote collaborative experiments based on service-oriented architecture(soa). In: *Signal Processing Systems (ICSPS), 2010 2nd International Conference on*, 2010, p. V2-605 –V2-608.
- ZHANG, Y.; FU, W.; NIE, C. monadws: A monad-based testing tool for web services. In: *Proceedings of the 6th International Workshop on Automation of Software Test, AST '11*, New York, NY, USA: ACM, 2011, p. 111–112 (AST '11,).
Disponível em: <http://doi.acm.org/10.1145/1982595.1982622>