PEESOS-Cloud: uma arquitetura para o
planejamento e execução de experimentos em
sistemas orientados a serviços considerando a
carga de trabalho

Carlos Henrique Gomes Ferreira

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Carlos Henrique Gomes Ferreira

PEESOS-Cloud: uma arquitetura para o planejamento e execução de experimentos em sistemas orientados a serviços considerando a carga de trabalho

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*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Julio Cezar Estrella

USP – São Carlos
Maio de 2016

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

F634p Ferreira, Carlos Henrique Gomes
 PEESOS-Cloud: uma arquitetura para o planejamento
 e execução de experimentos em sistemas orientados
 a serviços considerando a carga de trabalho /
 Carlos Henrique Gomes Ferreira; orientador Julio
 Cezar Estrella. - São Carlos - SP, 2016.
 70 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, 2016.

 1. Avaliação de Desempenho. 2. Caracterização
 de Cargas de Trabalho. 3. Computação em Nuvem.
 4. Sistemas Orientados a Serviços. I. Estrella, Julio
 Cezar, orient. II. Título.

Carlos Henrique Gomes Ferreira

**PEESOS-Cloud: an architecture for the planning and
execution of experiments in service-oriented systems
considering the workload**

Master dissertation submitted to the Instituto de
Ciências Matemáticas e de Computação – ICMC-
USP, in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Julio Cezar Estrella

USP – São Carlos
May 2016

Este trabalho é dedicado a todos que um dia acreditaram em mim.

AGRADECIMENTOS

Novamente uma etapa está se concluindo, cujo durante todo o processo pude contar com apoios, auxílios, incentivos, críticas, sugestões e vários outros tipos de contribuições diretas e indiretas. Por isso, não poderia aqui deixar de agradecer essas pessoas que fazem parte dessa conquista e ajudaram a realizar esse sonho.

Início os meus agradecimentos a Deus, pois sem Ele nada disso seria possível. Em seguida ao meu Orientador Prof. Dr. Julio Cezar Estrella pela orientação, reconhecimento, compreensão e paciência durante todo esse período. Aos Professores do LaSDPC pelo conhecimento compartilhado, a toda Universidade de São Paulo e ao CNPq pela oportunidade, suporte e apoio financeiro, permitindo que essa história fosse construída.

Agradeço a toda minha família, pais, irmãos, avós, tios e primos, que contribuíram de forma direta e indireta. Em especial ao meu pai Francisco que vem sendo um exemplo de paciência, esperança e fé, meus irmãos Pedro e Francisco Filho por me terem como inspiração, meus tios Heveraldo, Grazielle, Manoel e Fabíola, meus primos José Maria, Jéssica, João, Manoel Filho e Alceu, pelo apoio e atenção, compreendendo as inúmeras vezes que tive que me ausentar durante todo esse tempo.

Não poderia deixar de agradecer também aos colegas do LaSDPC, em especial os que estiveram batalhando dia após dia. Esses foram irmãos que eu tive a oportunidade de escolher e assim compartilhar vitórias, derrotas, alegrias e tristezas durante o tempo de convivência. Assim como os amigos externos à Universidade e companheiros de república que pude contar durante todo esse tempo. Gostaria de agradecer também aos amigos de infância e da graduação, que mesmo a distância se fizeram presentes das diferentes formas.

Enfim, a todas as pessoas que estiveram ao meu lado durante esse tempo e certamente continuarão, meu muito obrigado!

*“Não existe vento favorável a quem não sabe onde deseja ir.”
(Sêneca)*

RESUMO

FERREIRA, C.H.G.. **PEESOS-Cloud: uma arquitetura para o planejamento e execução de experimentos em sistemas orientados a serviços considerando a carga de trabalho**. 2016. 70 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Este trabalho de mestrado tem como objetivo o desenvolvimento de uma arquitetura denominada PEESOS-Cloud para realização de experimentos em sistemas orientados a serviços capaz de verificar a carga de trabalho. Possíveis problemas durante a geração de carga em ambientes de experimentos foram identificados e sumarizados. Com base nesses problemas e limitando-se às características dos sistemas distribuídos, dois módulos e um modelo para geração de carga foram desenvolvidos e associado à PEESOS-Cloud. Uma avaliação experimental foi realizada demonstrando a efetividade da proposta em atuar na avaliação de sistemas orientados a serviços. Para isso, cenários de experimentos foram definidos variando o ambiente de execução da arquitetura. Assim, o desempenho de uma aplicação sintética oferecida como serviço foi estudada e avaliada. Os resultados mostraram como a arquitetura proposta permite identificar a característica da carga de trabalho e verificar seu impacto sobre avaliação de um sistema alvo. Além disso, como uma carga de trabalho descaracterizada infere em uma avaliação de desempenho inconsistente. Por outro lado, como essa mesma carga de trabalho, sujeita a condições específicas, pode subsidiar uma avaliação de desempenho sólida.

Palavras-chave: Avaliação de Desempenho, Caracterização de Cargas de Trabalho, Computação em Nuvem, Sistemas Orientados a Serviços.

ABSTRACT

FERREIRA, C.H.G.. **PEESOS-Cloud: uma arquitetura para o planejamento e execução de experimentos em sistemas orientados a serviços considerando a carga de trabalho.** 2016. 70 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

This master thesis aims at developing an architecture called PEESOS-Cloud for conducting experiments in the service oriented systems capable of respecting the workload. Possible problems during the generation of load experiments environments were identified and summarized. Based on these problems and limited to the characteristics of distributed systems, two modules and a model for load generation were developed and associated with PEESOS-Cloud. An experimental evaluation was performed demonstrating the effectiveness of the proposal of acting in the evaluation of service-oriented systems. For this, experiments scenarios were defined by varying the architecture of execution environment. Thus, the performance of a synthetic application offered as a service has been studied and evaluated. The results showed how the proposed architecture allows us to identify the characteristics of the workload. In addition, uncharacterized workload can provide an inconsistent performance evaluation. On the other hand, as the same workload, subject to specific conditions, can support a solid performance evaluation.

Key-words: Performance Evaluation, Workload Characterisation, Cloud Computing, Systems-Oriented Services.

LISTA DE ILUSTRAÇÕES

Figura 1 – Elementos de uma SOA básica. Adaptado de (BRANCA; ATZORI, 2012) .	6
Figura 2 – Modelo visual da definição de computação em nuvem proposta pelo NIST - Adaptado de (ALLIANCE, 2011)	8
Figura 3 – Classificação simplificada sobre a geração de carga de trabalho.	22
Figura 4 – Exemplo de um ambiente de experimentos com geração de carga de trabalho	23
Figura 5 – Exemplo de comparação entre as cargas de trabalhos	24
Figura 6 – Abstração da arquitetura PEESOS-Cloud	26
Figura 7 – Diagrama de sequência da PEESOS-Cloud simplificado	28
Figura 8 – Fluxo de uma requisição e como as métricas são aferidas.	31
Figura 9 – Modelo entidade relacionamente base de dados PEESOS-Cloud	32
Figura 10 – Funcionamento do protótipo implementado	33
Figura 11 – Ambiente de experimentos do cenário I	40
Figura 12 – Cargas de trabalho utilizada no experimento.	41
Figura 13 – Carga de trabalho gerada pela PEESOS-Cloud - Cenário I	43
Figura 14 – Carga de trabalho processada pelo provedor - Cenário I	43
Figura 15 – CDF das cargas de trabalho processadas pelo provedor - Cenário I	45
Figura 16 – Tempo de execução do serviço fatorial - Cenário I	46
Figura 17 – Taxa de utilização média da CPU - Cenário I	47
Figura 18 – Ambiente de experimentos do cenário II	49
Figura 19 – Carga de trabalho gerada pela PEESOS-Cloud - Cenário II	50
Figura 20 – Carga de trabalho processada pelo provedor - Cenário II	51
Figura 21 – CDF das cargas de trabalho processadas pelo provedor - Cenário II	52
Figura 22 – Tempo de execução do serviço fatorial - Cenário II	53
Figura 23 – Taxa de utilização média da CPU - Cenário II	54
Figura 24 – Ambiente de experimentos do cenário III	55
Figura 25 – Carga de trabalho gerada pela PEESOS-Cloud - Cenário III	56
Figura 26 – Carga de trabalho processada pelo provedor - Cenário III	57
Figura 27 – CDF das cargas de trabalho processadas pelo provedor - Cenário III	58
Figura 28 – Tempo de execução do serviço fatorial - Cenário III	58
Figura 29 – Taxa de utilização média da CPU - Cenário III	59

LISTA DE TABELAS

Tabela 1 – Comparação entre os trabalhos relacionados	18
Tabela 2 – Carga de trabalho em forma de série temporal	27
Tabela 3 – Decomposição da carga de trabalho	27
Tabela 4 – Descrição e exemplo dos parâmetros de entrada para o experimento	34
Tabela 5 – Exemplo: <i>Log</i> das requisições do experimento provenientes da base de dados	35
Tabela 6 – Média e desvio padrão do tempo de execução e de resposta	36
Tabela 7 – Carga de trabalho exemplo hipotético	36
Tabela 8 – Consumo de recursos durante o tempo de experimento	37
Tabela 9 – Média e desvio padrão dos recursos mensurados	37
Tabela 10 – Fatores e níveis dos experimentos	40
Tabela 11 – Especificação do provedor de serviço no cenário I	42
Tabela 12 – Especificação do servidor físico da MV no cenário I	42
Tabela 13 – Especificação do ambiente da PEESOS-Cloud	42
Tabela 14 – Fatores e níveis dos experimentos	48
Tabela 15 – Especificação da Amazon EC2 t2.micro	49
Tabela 16 – Especificação do provedor de serviço no cenário III	56
Tabela 17 – Especificação do servidor físico da MV no cenário III	56

LISTA DE ABREVIATURAS E SIGLAS

QoS		<i>Quality of Service</i>
API		<i>Application Programming Interface</i>
BaaS		<i>Benchmark as a Service</i>
CDF		Função de distribuição acumulada
IaaS		<i>Infrastructure as a Service</i>
JSON		<i>JavaScript Object Notation</i>
JVM		<i>Java Virtual Machine</i>
MVs		Máquinas Virtuais
NTP		<i>Network Time Protocol</i>
REA		Recursos Educacionais Abertos
RIA		<i>Rich Internet Application</i>
SLA		<i>Service Level Agreement</i>
SOA		<i>Service Oriented Architecture</i>
SOAP		<i>Simple Object Access Protocol</i>
TaaS		<i>Test as a Service</i>
UDDI		<i>Universal Description, Discovery and Integration</i>
URI		<i>Uniform Resource Identifier</i>
W3C		<i>World Wide Web Consortium</i>
WAR		<i>Web application Archive</i>
WSDL	...	<i>Web Services Description Language</i>
XaaS		Anything as a Service
XML		<i>eXtensible Markup Language</i>

SUMÁRIO

1	INTRODUÇÃO	1
1.1	Contextualização	1
1.2	Motivação e Objetivos	3
1.3	Estrutura da Monografia	4
2	FUNDAMENTAÇÃO TEÓRICA	5
2.1	Considerações Iniciais	5
2.2	SOA e <i>Web Services</i>	5
2.3	Computação em Nuvem	7
2.3.1	<i>Virtualização</i>	10
2.4	Avaliação de Desempenho	12
2.5	Considerações Finais	14
3	TRABALHOS RELACIONADOS	15
3.1	Considerações Iniciais	15
3.2	Discussão	15
3.3	Considerações Finais	20
4	PEESOS-CLOUD	21
4.1	Considerações Iniciais	21
4.2	Caracterização Do Problema Abordado	21
4.3	Arquitetura	25
4.3.1	<i>Modelo para geração de carga de trabalho distribuída</i>	26
4.3.2	<i>Integração dos módulos</i>	27
4.3.3	<i>Base de dados</i>	31
4.3.4	<i>Protótipo</i>	32
4.4	Execução de um experimento	34
4.5	Considerações Finais	38
5	AVALIAÇÃO EXPERIMENTAL	39
5.1	Considerações Iniciais	39
5.2	Cenário I	39
5.2.1	<i>Resultados</i>	42
5.2.2	<i>Análise dos resultados</i>	46

5.3	Cenário II	48
5.3.1	<i>Resultados</i>	50
5.3.2	<i>Análise dos resultados</i>	53
5.4	Cenário III	55
5.4.1	<i>Resultados</i>	56
5.4.2	<i>Análise dos resultados</i>	57
5.5	Considerações Finais	59
6	CONCLUSÕES	61
6.1	Trabalhos Futuros	62
	REFERÊNCIAS	65

INTRODUÇÃO

1.1 Contextualização

Service Oriented Architecture (SOA) permite o desenvolvimento de aplicações baseadas na integração de *Web Services*, os quais possuem funcionalidades específicas e podem ser reutilizados por várias aplicações. Com isso, os *Web Services* têm sido cada vez mais utilizados para disponibilizar as funcionalidades implantadas em SOA devido a sua capacidade de interoperabilidade e baixo acoplamento (TEIXEIRA *et al.*, 2010; SERRANO; HERNANTES; GALLARDO, 2014).

A computação em nuvem surgiu a partir da integração de SOA e outros campos da computação como uma nova proposta de fornecer aplicações, plataformas e recursos computacionais como serviço baseado no modelo de pagamento por uso (BUYYA; BROBERG; GOSCINSKI, 2011). Os *Web Services* são utilizados na prestação desses serviços e possibilitam a reutilização das funcionalidades por entidades externas integrando diferentes recursos computacionais, tais como, computadores, bases de dados e redes, fornecendo uma ligação lógica que pode ser invocada por um cliente (SOUZA *et al.*, 2013).

Os atributos de *Quality of Service* (QoS) são utilizados para mensurar a qualidade dos serviços oferecidos com o intuito de garantir sua eficiência e eficácia. De acordo com a *World Wide Web Consortium* (W3C), os atributos de QoS dos *Web Services* são: desempenho, confiabilidade, escalabilidade, capacidade, robustez, precisão, integridade, acessibilidade, disponibilidade, interoperabilidade e segurança (BOOTH *et al.*, 2004). A necessidade de entender as condições de execução de um serviço não define apenas a qualidade da funcionalidade fornecida, mas também possibilita que as garantias estabelecidas entre cliente e provedor, descritas em um *Service Level Agreement* (SLA), sejam cumpridas. Além disso, esses atributos são utilizados para seleção e composição de serviços por parte dos consumidores.

Entretanto, garantir os níveis acordados de QoS é um desafio, devido a natureza dinâmica

e distribuída utilizada na construção dessas aplicações, que dificulta a avaliação dos possíveis cenários de funcionamento (BANZAI *et al.*, 2010). As aplicações podem ser construídas por meio da composição de vários serviços, que são reutilizados em tempo de execução envolvendo diferentes provedores e que estão hospedados em nós geograficamente distribuídos. Isso faz com que tais aplicações sejam consideradas como uma caixa preta, sendo necessário avaliar os fluxos de execução de um serviço individualmente ou uma composição de serviços. Neste caso, os problemas com os atributos de *QoS* precisam ser considerados em cada serviço, bem como na composição de serviços (PETROVA-ANTONOVA; ILIEVA; STOYANOVA, 2013; KUEHNE *et al.*, 2015).

Na literatura, várias abordagens são aplicadas a estudos que buscam assegurar os níveis desejáveis de *QoS*. O planejamento de capacidade permite estudar as tendências de utilização dos recursos sob determinadas condições ao longo de um período. Assim o consumo e o gerenciamento dos recursos alocados aos *Web Services* são averiguados sobre uma determinada carga de trabalho (MENASCE; ALMEIDA, 2001; SMIT *et al.*, 2008). Os testes de desempenho, testes de carga e testes de *stress* são utilizados para garantir que os serviços possam funcionar corretamente com uma dada carga de trabalho (JIANG; HASSAN, 2015). A partir desses testes, os *benchmarks* surgem como uma opção que permitem comparar o desempenho entre provedores de serviços avaliando o custo benefício (TCHANA *et al.*, 2015).

Embora as abordagens mencionadas possam ser aplicadas a estudos de *QoS* em sistemas orientados a serviços, essas dependem diretamente da geração de cargas de trabalho. Para isso, um ambiente de experimentação, comumente, é composto por três principais componentes: o orquestrador de experimentos, os agentes responsáveis por gerar uma carga de trabalho e o sistema alvo de avaliação. Esses componentes estão associados a um conjunto de recursos computacionais, tais como, servidores e provedores de serviço. Como alternativa aos ambientes locais, que geralmente contém recursos limitados, o uso da computação em nuvem se torna atraente nessas circunstâncias devido a capacidade de provimento de recursos sob demanda.

Por outro lado, tanto nos ambientes locais quanto na nuvem, há possibilidade de falhas de *hardware*, *software*, comunicação e sincronização entre as partes envolvidas. Em função disso, ruídos podem descaracterizar a carga de trabalho imposta sobre o sistema alvo. Desse modo, a carga de trabalho gerada pode ser totalmente distinta da carga de trabalho especificada acarretando em análises e conclusões inconsistentes. Identificar a ocorrência dessas falhas durante um experimento e quantificar o efeito na avaliação de serviço é uma tarefa não trivial devido à complexidade do ambiente distribuído. As perdas nesse caso vão além da definição errada dos níveis de *QoS* do serviço avaliado. Os recursos envolvidos e as atividades de planejamento, execução e análise dos resultados também são fontes de perdas.

Ao considerar essas soluções oferecidas como serviços, no caso do *Benchmark as a Service* (BaaS) e *Test as a Service* (TaaS) as consequências podem ser ainda piores, pois a violação do SLA pode ocorrer em duas circunstâncias: na primeira, quando essas soluções são contratadas

para a avaliação de uma aplicação, visto que o cliente muitas vezes não possui a característica da carga de trabalho gerada em relação a especificada; Posteriormente, quando esse serviço que supostamente foi avaliado com uma carga de trabalho, passa a ser oferecido a clientes com níveis de *QoS* validados incorretamente. Nesse contexto, uma solução capaz de identificar e dimensionar o impacto das possíveis falhas ao avaliar um sistemas orientado a serviços é um desafio em virtude das adversidades desses ambientes.

1.2 Motivação e Objetivos

A ascensão da computação em nuvem faz com que o desenvolvimento de novos serviços seja constante. Porém, garantir o desempenho desses serviços após a implantação no ambiente distribuído e paralelo é desafiador. Os clientes esperam serviços sem nenhuma interrupção ou perda de dados. Assim os provedores devem garantir *QoS* para ganhar a confiança dos clientes e oferecê-los a satisfação esperada (BANZAI *et al.*, 2010; BATISTA *et al.*, 2015). Várias são as condições que podem acarretar em uma violação de SLA. Por exemplo, problemas de tráfego inesperado como no lançamento do sistema de saúde americano Obamacare (MORGAN; HUMER, 2013), falta de planejamento de capacidade das aplicações em períodos especiais, como na *Black Friday* (BLOOMBERG, 2015), falhas técnicas e de segurança como na rede Xbox Live (JONES, 2014).

Tais problemas podem ser detectados por meio do uso de abordagens que aplicam cargas de trabalho em um processo de pré-implantação. No entanto, as dificuldades estão na falta de métodos precisos de experimentos que explorem os problemas de disponibilidade, confiabilidade e escalabilidade antes de disponibilizar as aplicações para o mundo real (SAROJADEVI, 2012). A possibilidade de desenvolver, aplicar e avaliar novos índices de desempenho em sistemas orientados a serviços que representem efetivamente uma determinada carga de trabalho imposta a aplicação é uma oportunidade de melhorar a qualidade das aplicações (SOUZA *et al.*, 2013).

Bai *et al.* (2011) apresentam uma comparação entre o ambiente de nuvem e o ambiente local para testes de carga. A principal diferença é que na nuvem os recursos teoricamente são ilimitados e escaláveis ao contrário de um ambiente local. Por outro lado, na nuvem o ambiente é menos controlável, sendo preciso gerenciar bem os recursos contratados para diminuir os custos. Os autores realizam um levantamento de várias abordagens e ferramentas de testes que utilizam a nuvem e destacam dois aspectos: 1) a qualidade introduzida na proposta, capaz de considerar a arquitetura dinâmica da nuvem; 2) A oportunidade de melhorar a capacidade dos sistemas de testes utilizando o serviço de infraestrutura da nuvem para superar limitações de custos e recursos.

Para Nunes *et al.* (2014), a principal vantagem de abordagens que utilizam protótipo para avaliação de desempenho é a possibilidade de resultados realísticos, ao contrário de abordagens baseadas em simulação ou modelagem analítica, em que a identificação de gargalos se torna

difícil. No trabalho desenvolvido pelos autores é proposta a PEESOS, um mecanismo para estudos de desempenho e QoS em um sistema orientado a serviço denominado WSARCH (ESTRELLA; SANTANA; SANTANA, 2011). Porém, esse mecanismo é limitado a WSARCH e possui problemas na geração da carga de trabalho. Apesar disso, tal trabalho é tido como motivação para a realização de estudos em outros sistemas orientados a serviços considerando a nuvem.

Nesse contexto, este trabalho tem como objetivo propor uma arquitetura para a realização de experimentos capaz de verificar as características da carga de trabalho. Tal arquitetura é chamada de *Planning and Execution for Experiments in Service Oriented Systems based on Cloud* - PEESOS-Cloud. Estudos realizados na literatura permitiram identificar os problemas associados à geração de carga de trabalho. Com isso, uma arquitetura genérica e extensível foi desenvolvida capaz de ser aplicada em vários ambientes com diferentes objetivos. Um modelo de geração de carga de trabalho que permite verificar as características da carga gerada segundo as especificações do usuário no experimento foi estabelecido. Dado esse modelo, é possível verificar as características da carga de trabalho ao avaliar um sistema. Por último, um protótipo foi utilizado para avaliar a arquitetura proposta e aplicado a estudos de caso.

1.3 Estrutura da Monografia

Esta monografia está organizada da seguinte forma: o Capítulo 2 apresenta uma fundamentação teórica dos temas abordados nesse projeto objetivando a compreensão do trabalho realizado. O Capítulo 3 apresenta soluções com diferentes objetivos que utilizam a geração de carga de trabalho distribuída, evidenciando a contribuição perante a literatura. O Capítulo 4 apresenta o trabalho desenvolvido. O Capítulo 5 apresenta uma avaliação experimental com resultados e discussões. Finalmente, o Capítulo 6 apresenta as conclusões e os trabalhos futuros.

FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

Este Capítulo apresenta uma fundamentação teórica dos principais temas abordados neste projeto de pesquisa. Na Seção 2.2 é apresentado os conceitos de SOA *Web Services*. Subsequente, a Seção 2.3 discute sobre a computação em nuvem, como os serviços são oferecidos e os desafios ainda presentes. Por último, a Seção 2.4 apresenta a Avaliação de Desempenho, uma abordagem ligada ao estudo de sistemas computacionais e é aplicada nesse projeto.

2.2 SOA e *Web Services*

Arquitetura Orientada a Serviços é um paradigma utilizado para o desenvolvimento e integração de sistemas que auxiliam as empresas organizarem e utilizarem recursos distribuídos envolvendo outras partes. Para isso, SOA estabelece como estes serviços são oferecidos, descobertos, e se interagem em diferentes tipos de sistema. SOA não é um tipo de tecnologia específica, apenas define padrões voltados para a integração e reutilização de serviços em ambientes heterogêneos (ERL, 2007). Em uma SOA tradicional, basicamente três tipos de componentes são conhecidos, conforme apresentado na Figura 1.

O provedor de serviço é responsável por oferecer o serviço tornando-o disponível. Consequentemente, o serviço tem que ser publicado na rede para se tornar disponível para consumidores ou outros serviços. O provedor de serviço envia para o registrador de serviço (que também é conhecido como *Universal Description Discovery and Integration* (UDDI)) as informações úteis para que o serviço seja publicado (PAPAZOGLU, 2003).

Assim, o registrador de serviço ou UDDI mantém informações como tipo de serviço, URL, como o serviço pode ser invocado e todas as funcionalidades que o serviço é capaz de proporcionar ao consumidor de serviço. Ao fim, quando um consumidor de serviço deseja usar

Figura 1 – Elementos de uma SOA básica. Adaptado de (BRANCA; ATZORI, 2012)

um serviço, ele tem como base informações contidas no registrador de serviço para comunicação com o provedor de serviço (PAPAZOGLU, 2003; BRANCA; ATZORI, 2012).

Para o desenvolvimento de uma SOA, alguns princípios devem ser seguidos (ERL, 2007). São eles:

- **Padronização do contrato dos serviços:** o contrato apresenta a proposta e a capacidade do serviço contratado. O mesmo estabelece informações como restrições técnicas, requisitos necessários, entre outras informações para que o serviço seja disponibilizado por uma interface.
- **Baixo acoplamento dos serviços:** o baixo acoplamento do serviço estabelece que o serviço deve ter o máximo de independência com outros serviços para realização de uma tarefa;
- **Abstração dos serviços:** este princípio estabelece que seja ocultado o máximo de informações de um serviço, fornecendo apenas o necessário para o uso, preservando o baixo acoplamento descrito anteriormente;
- **Reusabilidade dos serviços:** é um princípio fortemente desejável em SOA. Este princípio estabelece que serviços sejam utilizáveis para o maior número de propósitos possíveis;
- **Autonomia dos serviços:** é desejável que o serviço tenha o maior grau de independência possível, o que facilita o gerenciamento e manutenção;
- **Ausência de estado dos serviços:** manter informações de estado podem comprometer o bom funcionamento de um serviço. Por isso, é desejável que um serviço seja totalmente sem estado;

- **Descoberta dos serviços:** informações para a descoberta do serviço precisam estar facilmente disponíveis para que os mesmos sejam descobertos e assim utilizados;
- **Composição dos serviços:** a capacidade de um serviço combinar com outro serviço para atingir um objetivo final é essencial para alcançar os objetivos de SOA.

Atualmente, a tecnologia mais associada à implementação de SOA tem sido os *Web Services* (ERL, 2007). A organizadora de padrões W3C define *Web Services* como aplicações disponibilizadas por um URI, cuja interfaces e ligações são definidas, descritas e descobertas utilizando uma linguagem padrão como *eXtensible Markup Language* (XML). Estas aplicações podem interagir entre si por meio de mensagens baseadas em XML transmitidas pela Internet (World Wide Web Consortium, 2004).

Os *Web Services* são fundamentais para a prestação de serviços remotos. Pois, permitem a integração entre computadores, bases de dados e redes, criando uma ligação lógica que pode ser invocado por cliente distantes, independente da plataforma (SOUZA *et al.*, 2013). Para isso, protocolos como por exemplo, *Universal Description, Discovery and Integration* (UDDI), *Web Services Description Language* (WSDL) e *Simple Object Access Protocol* (SOAP), são utilizados respectivamente, para descoberta, descrição e troca de mensagem em *Web Services* (ALONSO *et al.*, 2010). Com isso, empresas tem a possibilidade de integrar e utilizar diferentes serviços por meio da composição de serviços de forma transparente para o usuário (KUEHNE *et al.*, 2015).

Essa seção apresentou conceitos relacionados a SOA e a *Web Services* e as características presentes nessas tecnologias. A próxima seção apresenta os conceitos de computação em nuvem.

2.3 Computação em Nuvem

O Instituto Nacional de Padrões e Tecnologias (do inglês, *National Institute of Standards and Technology* (NIST:2011)) define a computação em nuvem como: “um modelo que permite o acesso à rede de forma ubíqua, conveniente e sob demanda, a um conjunto compartilhado de recursos computacionais configuráveis que podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou interação com o provedor de serviços” (MELL; GRANCE, 2011).

A computação em nuvem surgiu a partir da evolução e integração de vários campos da computação, incluindo *utility computing*, computação distribuída, computação em grade, *Web Services* e arquitetura orientada a serviços (WEISS, 2007; MOTAHARI-NEZHAD; STEPHENSON; SINGHAL, 2009). A nuvem pode ser considerada como um conjunto de recursos que fornecem serviços para os clientes sob demanda, no qual o consumidor não tem conhecimento preciso de onde estes serviços estão sendo executados. Estes recursos são explorados na Internet por meio de um modelo de pagamento por uso conhecido como *pay-per-use* (RIMAL; CHOI; LUMB, 2009).

Ao contratar um serviço em nuvem, inicialmente os atributos de QoS são identificados. Em seguida, os níveis destes atributos são definidos, ocasionando em uma negociação entre o provedor e o consumidor. Por último, o resultado desta negociação é descrito em um SLA. Desde então, é esperado o cumprimento desse contrato sem que haja violação dos níveis de QoS definidos (VAQUERO *et al.*, 2008; PEIXOTO *et al.*, 2010).

Segundo o (MELL; GRANCE, 2011) a computação em nuvem é composta por cinco características essenciais, três modelos de serviço, e quatro modelos de implantação. A Figura 2 representa essa definição graficamente.

Figura 2 – Modelo visual da definição de computação em nuvem proposta pelo NIST - Adaptado de (ALLIANCE, 2011)

Quanto as cinco características, o NIST as definem como (MELL; GRANCE, 2011):

- **Autoatendimento sob demanda:** refere-se ao fato da nuvem permitir os clientes requererem recursos como rede, processamento e armazenamento, quando necessário sem interação humana com o provedor de serviços;
- **Ampla acesso à rede:** O amplo acesso a rede está relacionado a disponibilidade de acesso a nuvem por meio da Internet sem distinção do dispositivo por meio de *software* ou *hardware*;
- **Agrupamento de recursos:** estabelece que os clientes possam especificar pelo menos em alto nível informações sobre a localização do serviço contratado, como por exemplo, o país, estado ou *datacenter*;

- **Elasticidade rápida:** possibilita o aumento dos recursos computacionais sob a condição de pagamento, apresentando-se de maneira ilimitada para o cliente. Tal operação é totalmente transparente para o cliente e pode ser realizada a qualquer momento.
- **Medição de serviço:** os serviços devem ser mensuráveis, otimizando o uso de recursos de acordo com os níveis apropriados de cada serviço.

Na computação em nuvem três modelos são comumente utilizados e amplamente difundidos. Esses serão descritos com mais detalhes, porém, um termo coletivo tem sido utilizado para representar um número de coisas que podem ser oferecidas como serviço na computação, tais como: *software*, plataforma, infraestrutura, segurança, dados, sensores, dentre outros. Esse modelo é conhecido como Anything as a Service (XaaS) (RIMAL; CHOI; LUMB, 2009; PERERA *et al.*, 2013).

Os três principais modelos de serviços estão organizados da seguinte forma: na base tem-se a Infraestrutura como Serviço (do inglês, *Infraestrutura As A Service* (IaaS)), no nível superior a IaaS, o modelo de Plataforma como Serviço, (do inglês *Platform As A Service* (PaaS)) e por último o modelo de Software como Serviço (do inglês, *Software As A Service* (SaaS)). Cada modelo é um pacote de soluções independentes que funcionam de forma integrada, onde o IaaS fornece a base para os demais serviços. Na construção de um modelo de serviço de nuvem é necessário a combinação lógica desses três modelos de serviços (VAQUERO *et al.*, 2008; FURHT, 2010). O NIST apresenta as seguintes definições sobre tais modelos:

- **SaaS:** permite o consumidor acessar as aplicações a partir de vários dispositivos, como por exemplo, um navegador *Web* ou uma interface de programa, clientes de e-mail e servidores de e-mail. Neste caso, o consumidor não gerencia e nem controla a infraestrutura subjacente da nuvem, apenas, e em alguns casos específicos, é possível definir configurações da aplicação (RIMAL; CHOI; LUMB, 2009; MELL; GRANCE, 2011).
- **PaaS:** oferecem ao consumidor a capacidade de desenvolver ou implantar as aplicações na infra-estrutura de nuvem utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. Como por exemplo, plataformas .NET, Java e SGBDs. O consumidor não gerencia e nem controla a infraestrutura subjacente da nuvem, apenas as aplicações desenvolvidas (RIMAL; CHOI; LUMB, 2009; MELL; GRANCE, 2011).
- **IaaS:** recursos de infraestrutura são disponibilizados como serviços, tais como, capacidade de processamento, armazenamento e redes fornecidas ao consumidor, em que o mesmo tem a possibilidade de implantação e execução de sistemas operacionais e aplicativos. Um exemplo são as máquinas virtuais (MVs) e o espaço de armazenamento contratado. Neste modelo o consumidor não possui o gerenciamento ou controle da infraestrutura de nuvem,

apenas do sistema operacional, aplicativos, e recursos de redes limitados (RIMAL; CHOI; LUMB, 2009; MELL; GRANCE, 2011).

Os vários tipos de serviços oferecidos em nuvem podem ser implantados em três tipos de nuvem. Esses são definidos da seguinte forma:

- **Nuvem privada:** a infraestrutura da nuvem é exclusivamente utilizada por uma única organização por meio de vários consumidores. A mesma pode ser gerida pela própria organização ou terceirizada, podendo existir ou não no local da organização (MELL; GRANCE, 2011). É um modelo que permite a modificação da infraestrutura de acordo com as necessidades do cliente (FURHT, 2010).
- **Nuvem comunitária:** a infraestrutura da nuvem é exclusivamente utilizada por uma comunidade específica de consumidores de várias organizações. Comumente, essas organizações possuem interesses em comum, como por exemplo: missão, requisitos e políticas de segurança. Pode ser gerida por parte destas organizações ou todas, terceiros, podendo ou não existir no local da organização (FURHT, 2010; MELL; GRANCE, 2011).
- **Nuvem pública:** a infraestrutura da nuvem é disponibilizada ao público em geral. A mesma pode pertencer e ser gerenciada por uma empresa, organização acadêmica, governo ou uma combinação destes (MELL; GRANCE, 2011). Neste modelo de nuvem os serviços podem ser gratuitos, sendo interessante para usuários que desejam flexibilidade em serviços temporários que implicam na redução de custos (FURHT, 2010).
- **Nuvem híbrida:** a infraestrutura da nuvem é uma composição de dois ou mais modelos de implantação, podendo pertencer a entidades únicas ou unidas por tecnologias que podem ser proprietárias (MELL; GRANCE, 2011). Também pode ser adequada quando em uma nuvem pública o desempenho não é satisfatório, tornando-se necessário alternar para nuvem privada (FURHT, 2010).

As características essenciais, os modelos de serviços e de implantação da nuvem estão associados a uma tecnologia denominada virtualização. A próxima subseção apresenta como essa tecnologia está fortemente associada a nuvem.

2.3.1 Virtualização

A característica da nuvem de permitir que serviços sejam ofertados aos consumidores por demanda exige que os recursos sejam alocados dinamicamente. Com isso, os provedores de nuvem buscam alocar recursos com o intuito de maximizar aqueles utilizáveis e minimizar os custos operacionais e de infraestrutura (FERREIRA *et al.*, 2014; LEITE *et al.*, 2016). Dessa forma, o uso da virtualização é fundamental. Essa técnica permite particionar um único *host* em

vários outros denominados de MVs, na qual cada uma dessas MVs oferece um ambiente similar a de uma máquina física independente, cada qual com seu sistema operacional, aplicativos e serviços de rede (WHITE; PILBEAM, 2010).

Os monitores de máquinas virtuais são quem possibilitam alocar e gerenciar as MVs tornando transparente toda a manipulação necessária na nuvem (HEAD *et al.*, 2010). Isso ocorre nos *data centers*, que podem ser considerados como um conjunto de servidores onde em cada um deles há um conjunto de MVs (BUYYA; RANJAN; CALHEIROS, 2010). As principais vantagens em se utilizar a virtualização são (MENASCÉ, 2005; SAHOO; MOHAPATRA; LATH, 2010):

- **Flexibilidade** pela possibilidade de alocar várias MVs a um *host* físico de maneira independente. Realizar operações de *start*, *stop*, *resume*, *boot* nas MVs. Manipulação do recursos alocadas às MVs, como a quantidade de memória, CPU e disco;
- **Confiabilidade e disponibilidade** devido às várias instâncias de MVs em execução. Caso uma falhe a outra pode ser alocada rapidamente, além de fácil recuperação por meio de uma cópia de *backup*. Há também a possibilidade de migração de forma que o tempo de inatividade dos serviços se tornam transparente aos clientes;
- **Escalabilidade** porque a adição e remoção de MVs é mais fácil do que de *hosts* físicos;
- **Utilização do hardware** aumenta devido ao compartilhamento dos recursos entre as MVs hospedadas em um único *host* físico;
- **Confiabilidade** porque serviços podem ser instanciados em MVs separadas e independentes. Assim, caso ocorra uma falha em um dado serviço, essa não irá afetar os outros em questão de *hardware* ou *software*;
- **Redução de custos** devido a maximização do uso de recursos, tornando servidores menores e em maior quantidade. Além da redução de custos com energia, espaço físico e mão de obra;
- **Adaptabilidade** em relação a variação de cargas de trabalho. Isso porque é possível alocar mais máquinas virtuais ou manipular os recursos de uma determinada máquina virtual;
- **Balanceamento de carga** uma vez que é possível trabalhar com replicação e migração de MVs com intuito de melhorar o desempenho;
- **Aplicações legadas** que precisam ser migradas podem explorar a flexibilidade e a possibilidade de migração em ambientes heterogêneos.

A virtualização possui muitas vantagens. Por outro lado, também há desvantagens em utilizá-la, das quais destacam-se (MENASCÉ, 2005; SAHOO; MOHAPATRA; LATH, 2010):

- **Sobrecarga** do *host* físico que hospeda as MVs pode interferir no desempenho das aplicações;
- **Segurança** apesar de uma vantagem também pode ser vista como desvantagem. Se o *host* físico tem a segurança comprometida, as MVs hospedadas neste *host* estão vulneráveis;
- **Ponto único de falha** do *host* físico é um risco apesar das MVs estarem desacopladas ao equipamento. Dessa forma, caso haja uma falha de *hardware* a máquina virtual será desligada ou reiniciada;
- **Interface de gerenciamento** está ligada a plataforma de virtualização. Isso pode gerar problemas relacionados a utilização de diferentes plataformas no mesmo ambiente.

Essa subseção apresentou a tecnologia base para a computação em nuvem e suas vantagens e desvantagens. A seção a seguir irá apresentar os conceitos de Avaliação de desempenho.

2.4 Avaliação de Desempenho

Os projetistas de sistemas computacionais estão interessados em fazer uma avaliação do desempenho destes sistemas com o objetivo de obter um melhor desempenho a um menor custo. Com isso, obter uma melhor relação custo benefício e definir quando é melhor utilizar um sistema de determinada forma para diferentes aplicações (JAIN, 1991).

Para aplicar a avaliação de desempenho, uma série de procedimentos devem ser seguidos. Isso envolve a necessidade de compreender o sistema alvo a ser avaliado, bem como definir e aferir as métricas, a carga de trabalho e o tipo de técnica utilizada. Jain (1991) apresenta um conjunto de passos para realizar a avaliação de desempenho:

- **Definir os objetivos da avaliação e o sistema alvo:** O primeiro em um projeto de avaliação de desempenho é indicar os objetivos do estudo e caracterizar o sistema;
- **Listar os serviços e as possíveis saídas:** Um sistema possui serviços que podem possuir várias saídas, desejadas ou não. É preciso listar essas possibilidades;
- **Selecionar métricas:** As métricas correspondem aos critérios que permitem definir o desempenho do sistema. Essas métricas são associadas nas saídas. Por exemplo, métricas relacionadas com velocidade, número de falhas, vazão, tempo de resposta e disponibilidade dos serviços;
- **Listar os parâmetros:** É preciso listar todos os parâmetros que interferem no desempenho do sistema. Esses parâmetros podem ser relacionados ao sistema, tais como características de *hardware* e *software* ou relacionados à carga de trabalho.

- **Selecionar fatores:** A lista de parâmetros pode ser dividida em duas partes: aqueles que variam durante a avaliação e aqueles que são fixos. Os parâmetros que variam são chamados de fatores e os seus valores são chamados níveis.

Diferentes modelos de planejamento de experimentos podem ser utilizados, cada qual com suas vantagens e desvantagens. Os mais utilizados são:

- **Planejamento simples:** Projeto simples, em que um fator é fixado, variando os níveis a fim de observar o quanto esses valores influenciam no desempenho. Sua principal vantagem é um menor número de experimentos e facilidade no projeto, porém não é estatisticamente eficiente por não permitir um estudo completo em relação aos fatores. Em um número k de fatores e n_i de níveis, resulta em um número de experimentos conforme a Equação 2.1:

$$n = 1 + \sum_{i=1}^k (n_i - 1) \quad (2.1)$$

- **Planejamento fatorial completo:** Um planejamento desse tipo exige uma combinação de todos os níveis de todos os fatores. Pode encontrar o efeito de todos os fatores independentes de serem primários ou secundários, além de suas interações, ou seja, é possível a realização de uma análise completa. O problema desse tipo de experimento é o custo elevado do estudo, principalmente se o número de replicações for elevado. Há três maneiras de reduzir a complexidade desse tipo de planejamento de experimento:

- **Redução dos níveis de cada fator:** Uma das alternativas mais recomendadas, em alguns casos a utilização de apenas dois níveis por fator é o suficiente, chegando em número de experimentos reduzido conhecido como 2^k ;
- **Redução do número de fatores:** Reduzir o número de fatores e aumenta o número de níveis a fim de se chegar em resultados conclusivos;
- **Uso de fatorial parcial:** Essa técnica é considerada um modelo de planejamento de experimento distinto e será apresentado no próximo tópico.

Um experimento utilizando um projeto de planejamento fatorial completo com k fatores e n_i níveis resulta em número de experimentos conforme a Equação 2.2:

$$n = \prod_{i=1}^k n_i \quad (2.2)$$

- **Planejamento fatorial parcial:** conforme apresentado, no planejamento fatorial completo tem-se um nível de complexidade maior devido a quantidade de experimentos. Uma das técnicas de redução dessa complexidade é a realização do planejamento fatorial parcial, em que parte dos experimentos são excluídos quando se sabe que não há interação ou

a mesma não impacta de forma significativa no desempenho do sistema. A diminuição de níveis e/ou fatores torna o projeto mais fácil, porém apresenta riscos aos resultados, principalmente se não for utilizada alguma metodologia de escolha dos fatores.

Realizado o planejamento de experimentos, como alternativa de técnicas de avaliação de desempenho tem-se a aferição e a modelagem. A seleção da técnica envolve características e a disponibilidade do sistema a ser avaliado. Além disso, aspectos como o tempo, os recursos disponíveis e o nível de precisão desejado no estudo. Isso porque em ambientes modelados, a simulação pode ser utilizada como uma solução que demanda menos recursos, por outro lado, um menor nível de precisão. Já a aferição fornece informações mais precisas, porém, demanda mais recursos e que o sistema esteja em pleno funcionamento ainda que em protótipo (JAIN, 1991).

2.5 Considerações Finais

Este Capítulo apresentou os conceitos fundamentais para o projeto de pesquisa desenvolvido. Nos sistemas orientados a serviços e na computação em nuvem é possível identificar como os desafios dos serviços em nuvem estão relacionados a proposta apresentada. Na avaliação de desempenho, como a proposta desse projeto inclui conceitos da avaliação de desempenho utilizando aferição para contribuir com estudos sob os desafios em sistemas orientados a serviços disponíveis na nuvem.

TRABALHOS RELACIONADOS

3.1 Considerações Iniciais

Para a manutenção de QoS dos serviços oferecidos, diferentes abordagens que atuam em várias frentes da computação sobre perspectivas diferentes tem sido utilizadas. Na literatura é possível identificar propostas relacionadas aos testes de *software*, avaliação de desempenho e *benchmarks*. Além disso, empresas especializadas surgiram na última década, como (SOASTA, 2015; ITKO, 2015; IGATE, 2015; SMARTBEAR, 2015; LOADIMPACT, 2015; LOADSTORM, 2015). Porém, tais empresas partem de soluções proprietárias e consequentemente vão contra ao estudo realizado neste trabalho. Sendo assim, este capítulo descreve alguns trabalhos encontrados na literatura que se identificam com a proposta deste trabalho mas que possuem focos diferentes sob à geração de carga de trabalho distribuída para avaliar sistemas orientados a serviços.

3.2 Discussão

Dillenseger (2009) apresenta a importância de tratar problemas de capacidade e dimensionamento dos sistemas computacionais, uma vez que esses podem ocasionar em diversos prejuízos. Os autores então propõem um *framework* flexível a diferentes sistemas computacionais para geração de carga de trabalho distribuída. A arquitetura do CLIF *framework* é composta por injetores de carga distribuídos que geram carga para o sistema alvo, sondas que mensuram métricas de desempenho tanto no sistema alvo quanto nos injetores de carga, e componentes que armazenam e permite analisar tais métricas. Uma avaliação do *framework* foi realizada com foco na flexibilidade, desempenho e escalabilidade a partir de estudos de casos. Em relação à flexibilidade, os autores apresentam algumas aplicações práticas em que o CLIF pode ser utilizado. Quanto ao desempenho e escalabilidade os autores não apresentam os resultados, mas discutem alguns pontos, tais como: a independência dos componentes não é um fator limitante nessas questões, e a influência negativa da *Java Virtual Machine* (JVM) em um processo de

avaliação já que todo o *framework* é desenvolvido sobre a plataforma Java. A proposta tem como principal vantagem ser flexível e aberta a novas propostas como [Tchana et al. \(2013\)](#), que apresentou um *benchmark* construído a partir do CLIF *framework*.

[Tchana et al. \(2015\)](#) enfatizam que apesar da computação em nuvem prover os recursos necessários para testes com cargas de trabalho, o desafio está em alocar, implantar e gerenciar estes recursos de forma eficiente. Assim, testes distribuídos com carga de trabalhos devem considerar a sobrecarga dos injetores de carga, ocasionando em cenários diferentes do especificado e medições errôneas, além do desperdício de recursos alocados que não são utilizados. Com base no CLIF *framework* ([DILLENSEGER, 2009](#); [TCHANA et al., 2013](#)) os autores propõem uma solução de BaaS auto escalável para geração de carga de trabalho. Para isso, eles desenvolveram um protocolo que adiciona e remove injetores de carga no CLIF *framework* de acordo com o perfil da carga. Experimentos foram realizados considerando três cargas de trabalho com diferentes comportamentos em relação a variação do número de injetores necessários ao longo do experimento. Os resultados apresentados mostram que as políticas de adicionar e remover injetores automaticamente implementadas reduzem o custo e mantém a efetividade da proposta.

Segundo [Snellman, Ashraf e Porres \(2011\)](#) devido ao crescimento do tamanho e da complexidade das *Rich Internet Application* (RIA) e da infraestrutura em nuvem que hospeda esses aplicativos, é preciso considerar testes de desempenho e a escalabilidade destas aplicações. Diferente das abordagens que executam testes distribuídos por meio de uma requisição HTTP, RIAs são tecnologias assíncronas que possuem processamento do lado do servidor e do cliente. Sendo assim, o método de geração de carga de trabalho baseado em uma requisição HTTP não funciona. Dessa forma é necessário considerar as ações do usuário em um nível mais elevado, que é a interação deste usuário com o navegador Web e consequentemente com o RIA. O desafio está em gerar uma carga de trabalho eficaz representando um amplo número de navegadores simultâneos realizando interação com estas aplicações. Porém, representar a execução de múltiplos navegadores é uma tarefa dispendiosa e cara, pela complexidade de gerenciá-los com a sua interface gráfica e a demanda de recursos.

Os autores então propõem um *framework* chamado ASTORIA para automatizar e simplificar a avaliação de desempenho e escalabilidade de RIAs. Esse *framework* utiliza *scripts* que automatizam a geração de testes com uma *Application Programming Interface* (API) para um navegador sem interface gráfica. Essa solução permite alocar um maior número de clientes distribuídos a partir de máquinas virtuais e reduzir custos para realização das avaliações. Um estudo de caso foi realizado em um RIA com um número máximo de 1000 usuários concorrentes alocados em 20 máquinas virtuais da Amazon avaliando o tempo de resposta do sistema alvo.

Em [Oberle e Szabo \(2015\)](#) os autores apresentam as possibilidades de testes que podem ser realizados utilizando a nuvem para gerar carga de trabalho. A partir disso, os autores propõem uma arquitetura em forma de protótipo com foco em TaaS. O protótipo desenvolvido permite especificar o cenário de teste contendo o serviço a ser testado e os nós distribuídos a serem

utilizados. Um estudo de caso foi realizado com o intuito de validar a arquitetura proposta. Para isso, foi utilizado um *Web Service* implementado com o *Jersey framework*. Os resultados mostraram que o protótipo é funcional e permite mensurar métricas do ambiente considerado.

Cunha, Mendonca e Sampaio (2013) relatam a importância em avaliar os provedores de *Infrastructure as a Service* (IaaS) antes de adquirir os serviços. Para apoiar esta atividade vários trabalhos permitiram avaliar ambientes de IaaS. Entretanto, os autores questionam que poucos trabalhos permitem a automatização do processo de avaliação desses ambientes voltado para avaliação de IaaS. Com isso, eles apresentam um ambiente programável para avaliação de desempenho de IaaS utilizando uma linguagem declarativa *Crawl* que permite especificar as informações necessárias para execução dos casos de testes de desempenho. Algumas destas são os número de provedores, métricas e configurações das Máquinas Virtuais (MVs). Para gerar carga de trabalho, um *benchmark* chamado FABAN7 foi utilizado. Experimentos foram realizados considerando os provedores de IaaS EC2 e Rackspace com intuito de avaliá-los e demonstrar o funcionamento do ambiente utilizando dois níveis de carga. Na primeira carga, considerada demanda baixa, o número de usuários concorrentes eram de 25 e 150, enquanto que na segunda, considerada moderada o número de usuários variava de 200 a 800. Os resultados permitiram avaliar a relação de custo/desempenho dos provedores considerando a carga de trabalho aproveitando melhor os recursos contratados.

No mesmo contexto, Chhetri *et al.* (2013) apresentam o Smart Cloud Bench, um *benchmark* de IaaS que permite selecionar instâncias de provedores de nuvem de acordo com requisitos de SLA desejados a partir de um módulo de seleção de provedores. A partir disso, as instâncias são implantadas com imagens pré construídas que contém os pacotes necessários para o provedor receber requisições. Então os agentes de testes, responsáveis por gerar carga de trabalho executam as requisições e ao final as métricas reportadas para o usuário. Como prova de conceito, um estudo de caso foi realizado avaliando um sistema de *e-commerce* de livraria implementado pelo TPC-W *benchmark*. O estudo de caso contemplou o uso de diferentes tipos de instâncias de três provedores em nuvem: Amazon EC2, GoGrid, e Rackspace. Os resultados permitiram avaliar quais instâncias de quais provedores eram adequadas para determinados níveis da carga de trabalho imposta.

Nunes *et al.* (2014) apresentam a dificuldade em testar o desempenho em arquiteturas orientadas a serviços devido a quantidade de componentes envolvidos e a interação entre eles. Com isso, os autores propõem a PEESOS (NUNES *et al.*, 2014; NUNES *et al.*, 2015), uma ferramenta para auxiliar no planejamento e execução de experimentos gerando carga de trabalho distribuída. Em um estudo de caso os autores apresentam uma avaliação de desempenho de uma arquitetura orientada a serviço denominada WSARCH que hospedava um *web service* com característica *CPU-Bound*. Os resultados permitiram identificar um gargalo de desempenho no servidor de *Log* da WSARCH, componentes responsável por manter informações de QoS e registro das requisições recebidas pelos provedores de serviço.

Motivados a executar testes com carga de trabalho mais com os benefícios de utilizar a nuvem, Yan *et al.* (2012) propuseram uma plataforma para teste de carga distribuído na nuvem com nome de WSTaaS para o teste de carga em *Web Services*. Tal plataforma é capaz de implantar, configurar e escalonar as partes envolvidas para teste de *Web Services* na nuvem. Experimentos foram realizados comparando a abordagem proposta em relação a outras presentes na literatura que utilizam apenas um nó para representar vários clientes. Os resultados mostraram que problemas como a sobrecarga do único nó em relação a largura de banda da rede e o número de *threads* limitam a capacidade do teste e comprometem a análise final.

Budai e Goldschmidt (2014) apresentaram um *framework* distribuído para geração de carga de trabalho com foco na avaliação de serviços hospedados na nuvem com intuito de estudar o impacto das interferências presentes na infraestrutura contratada sobre o desempenho das aplicações. Para isso, uma arquitetura desenvolvida é composta por agentes e um controlador. Os agentes alocados em máquinas virtuais são responsáveis pela geração de carga de trabalho no sistema alvo. As carga de trabalhos são configuradas por meio de mensagens entre o controlador de experimentos e os agentes. Um estudo de caso foi realizado com o objetivo de avaliar o MySQL *cluster* e identificar fatores limitantes de desempenho. Com base nesse estudo foi possível identificar que o MySQL *cluster* escala em relação ao número de nós, mas não no tamanho da base de dados.

A Tabela 1 apresenta algumas características sumarizadas nesses trabalhos.

Tabela 1 – Comparação entre os trabalhos relacionados

Trabalho	Objetivo	Geração de carga de trabalho	Automatização do processo	Considera a carga de trabalho
(DILLENSEGER, 2009)	Framework para teste de carga em ambientes computacionais.	Local	✓	✗
(TCHANA <i>et al.</i> , 2015)	Criar injetores de carga de trabalho auto escaláveis para testes.	Local	✓	✗
(SNELLMAN; ASHRAF; PORRES, 2011)	Testes de desempenho em RIAs disponíveis na nuvem.	Nuvem	✓	✗
(OBERLE; SZABO, 2015)	TaaS utilizando recursos da nuvem para gerar carga de trabalho.	Nuvem	✓	✗
(CUNHA; MENDONCA; SAMPAIO, 2013)	Avaliar o desempenho de aplicações hospedadas em IaaS	Nuvem	✗	✗
(CHHETRI <i>et al.</i> , 2013)	Geração de carga de trabalho na nuvem para avaliar sistemas.	Nuvem	✓	✗
(NUNES <i>et al.</i> , 2014)	Testes com carga de trabalho em <i>Web services</i> .	Local	✓	✗
(YAN <i>et al.</i> , 2012)	Testes de <i>Web services</i> com carga de trabalho gerada na nuvem	Nuvem	✓	✗
(BUDAI; GOLDSCHMIDT, 2014)	Geração de carga de trabalho na nuvem para avaliar sistemas.	Nuvem	✓	✗
PEESOS-Cloud	Arquitetura para a realização de experimentos capaz de verificar as características da carga de trabalho.	Nuvem Local	✓	✓

Os trabalhos apresentados utilizam e propõem alternativas para gerar e aplicar cargas de trabalhos sobre diferentes contextos. De acordo com o objetivo é possível identificar quais trabalhos são flexíveis e podem ser utilizados em vários contextos. Por outro lado, esses trabalhos não consideram como a carga de trabalho é gerada e com essa impacta no sistema alvo. A importância de avaliar tal característica é evidenciada no próximo Capítulo, onde a Seção 4.2 apresenta a caracterização do problema e os impactos na avaliação de um sistema orientado a

serviço.

Em [Dillenseger \(2009\)](#) a proposta tem como principal vantagem ser flexível e aberta a novas possibilidades como em ([TCHANA et al., 2015](#)) que apresentou um *benchmark* construído a partir do CLIF *framework*. Porém também não foi avaliada a carga apesar dos autores apresentarem e discutirem a influência negativa da JVM. Os autores recomendam dedicar uma sub rede para o ambiente que utiliza o CLIF *framework*. No entanto, como apresentamos, mesmo que esta rede seja local ainda há a possibilidade de ruídos na carga de trabalho gerada. [Tchana et al. \(2015\)](#) também não consideram a carga. Além disso, há toda uma dinamicidade de alocar MVs em tempo de execução, o que pode prejudicar ainda mais o ambiente utilizado para gerar carga.

Apesar da inovação em relação ao *framework* capaz de avaliar RIA, [Snellman, Ashraf e Porres \(2011\)](#) consideram 1000 usuários concorrentes em 20 MVs sem quantificar o impacto sobre a carga gerada neste cenário. Ao oferecer um tipo de serviço na nuvem é preciso garantir o acordo firmado com o cliente por meio do SLA. Em [Cunha, Mendonca e Sampaio \(2013\)](#) um número de usuários concorrentes foram definidos sem descrição dos recursos utilizados para simular estes nós. [Oberle e Szabo \(2015\)](#) utilizam à geração de carga de trabalho distribuída a oferecer TaaS, sem garantir ao cliente que a carga desejada foi aplicada corretamente sobre o sistema testado. Assim como ([CHHETRI et al., 2013](#)), que aplicam cargas de trabalho comparando diferentes provedores em cenários diferentes, sem garantia que a comparação seja justa a ponto das cargas impostas serem equivalentes.

[Nunes et al. \(2014\)](#) apresentam o trabalho mais semelhante deste projeto, porém não avalia a carga, mesmo com a proposta de utilizar a nuvem e clientes colaborativos. O *framework* apresentado em ([BUDAI; GOLDSCHMIDT, 2014](#)) também vai de encontro a proposta aqui apresentada, porém, a carga também não é considerada. [Yan et al. \(2012\)](#) questionam a importância de verificar como a carga de trabalho gerada em nuvem se comporta devido aos riscos em realizar testes de carga com *Web Services* considerando apenas um nó para simular vários clientes. Com isso, eles propõem uma solução de seleção de nós de acordo com a sua capacidade e utilização, comparando com outras soluções na literatura que não consideram as limitações do nó que gera a carga. Porém, os autores não consideram outros tipos de problema, como a falha deste nó, bem como a carga final imposta ao servidor.

A proposta deste trabalho de mestrado, denominada PEESOS-Cloud pode ser considerada genérica e extensível, pois a geração de carga a um sistema alvo é realizada por meio de uma *Uniform Resource Identifier* (URI) que contém parâmetros da aplicação quando necessário. O ambiente para geração de carga distribuída considerado pode ser local ou na nuvem. Quando considerado o ambiente em nuvem, é importante verificar a carga de trabalho resultante, pois a natureza dinâmica da nuvem e restrições de acesso por meio da Internet podem comprometer o processo de avaliação de desempenho dependendo do contexto. Além disso, os trabalhos apresentados tem como objetivo automatizarem o processos de avaliação alocando os recursos necessários de forma a implantar, gerenciar e coletar as informações necessárias. Essa é uma

característica importante, pois utilizar recursos na nuvem requer um gerenciamento preciso para evitar custos desnecessários devido a má utilização dos recursos.

3.3 Considerações Finais

Neste Capítulo foi apresentado trabalhos da literatura e que estão relacionados a proposta deste projeto. Além disso foi possível relacionar as propostas existentes e enfatizar as contribuições que este trabalho busca contemplar. O próximo Capítulo apresenta a proposta com base na modelagem do problema e na arquitetura desenvolvida.

PEESOS-CLOUD

4.1 Considerações Iniciais

Ao propor uma solução arquitetural para avaliação de sistemas orientados a serviço utilizando cargas de trabalhos com possibilidade de uso da nuvem requer considerar como essa carga é gerada pelos recursos responsáveis por simular os clientes. Posteriormente, com quais características ela chega no sistema alvo e, por último, avaliar o impacto que essa carga tem sob desempenho do sistema. Para isso, é necessário investigar quais potenciais problemas podem ocorrer durante o processo de execução dos experimentos e sumariá-los. Assim, apresentar uma proposta que permita identificar a ocorrência desses problemas e o impacto final sob a característica da carga de trabalho.

Este Capítulo apresenta o projeto desenvolvido. A Seção 4.2 resume e descreve os problemas que ocasionam em ruídos presentes na geração de carga de trabalho. Em seguida, a Seção 4.3 apresenta a proposta de uma arquitetura composta por módulos e um modelo para geração de carga de trabalho distribuída. A última Seção apresenta um protótipo que representa o funcionamento da arquitetura proposta.

4.2 Caracterização Do Problema Abordado

O teste com carga de trabalho denota qualquer carga de trabalho utilizada em estudos de desempenho, e pode ser classificada em real ou sintética (JAIN, 1991; FEITELSON, 2015). A partir desses dois tipos de carga, existe a possibilidade de execução e aplicação de forma centralizada ou distribuída. O diagrama da Figura 3 apresenta tal classificação.

A carga de trabalho real ou conhecida é oriunda de *traces* ou de qualquer base de informação que permita identificar as requisições que chegam em um dado sistema alvo. Uma vantagem é que desta carga de trabalho é uma representação mais próxima da realidade. Por outro

Figura 3 – Classificação simplificada sobre a geração de carga de trabalho.

lado, pode ser difícil obter e manipular *traces* devido a sua complexidade de armazenamento e quantidade de informações. A dificuldade em representar diferentes cenários é outra limitação de uma carga de trabalho real, o que limita a avaliação do sistema sobre uma única perspectiva (MENASCE; DOWDY; ALMEIDA, 2004). Como alternativa, tem-se a utilização de cargas de trabalho sintética, que podem ser construídas por meio de um algoritmo pseudoaleatório e parâmetros descritivos para criar um comportamento desejado ou ser extraída por meio da modelagem de uma carga real (MENASCE; DOWDY; ALMEIDA, 2004; XI *et al.*, 2011). A principal vantagem desse tipo de carga é a possibilidade de testar o sistema em diferentes condições e variações. Jain (1991) afirma que uma carga de trabalho real é algo que não pode ser reproduzido, ocorre uma única vez em um único momento. No entanto, essa carga pode ser observada e transformada em uma carga de trabalho sintética por meio de modelagens, permitindo a sua reprodução. Apesar desta divergência, neste trabalho, a classificação de carga de trabalho será dividida como sintética ou real, de forma que a real seja conhecida utilizando informações das requisições que chegam no sistema alvo.

Independentemente do tipo de carga de trabalho, há o esforço de tentar garantir a qualidade da mesma com objetivo de fazer com que a carga de trabalho gerada seja o mais similar possível da especificada. Na carga de trabalho executada de forma centralizada, a descaracterização devido a presença de ruídos ocorre em problemas como a sobrecarga nos componentes responsáveis por gerar a carga de trabalho, principalmente quando uma única

máquina desempenha o papel de representar muitos clientes (CHHETRI *et al.*, 2013; TCHANA *et al.*, 2015). Hsieh *et al.* (2014) mostra como o número excessivo de *threads* criadas nas instâncias que desempenham as gerações de carga influenciam negativamente na geração de carga de trabalho. Além desses problemas que podem ocorrer nas máquinas que hospedam o ambiente utilizado, ruídos de rede também podem ser adicionados, principalmente quando utilizado a nuvem em que os recursos tendem estar ainda mais distantes geograficamente. Para (ZHOU *et al.*, 2013), ao utilizar a nuvem como fonte de recursos, fatores como o desempenho do provedor de serviço e a comunicação e condições da rede podem influenciar na geração de carga de trabalho. Isso porque a qualidade da carga de trabalho gerada depende de como o sistema irá distribuir e coordenar os clientes que realizarão as requisições para o sistema alvo. Caso ocorra uma falha nos recursos contratados em tempo de execução, é difícil tomar uma decisão a tempo de não prejudicar a geração de carga de trabalho.

Este trabalho utiliza diretamente a geração de carga de trabalho sendo ela sintética ou real de forma distribuída como exibe a Figura 4.

Figura 4 – Exemplo de um ambiente de experimentos com geração de carga de trabalho

A Figura 5 apresenta um exemplo contendo uma carga de trabalho especificada junto a tentativa de execução denominada carga de trabalho gerada em 100 unidades de tempo, variando a quantidade de requisições por essa unidade de tempo durante a execução. É possível notar que em alguns momentos ocorrem comportamentos diferentes do esperado devido a incidência de ruídos. Inicialmente a carga de trabalho gerada pode ser equiparada à especificada, mas entre o intervalo de tempo 5 e 20 a carga de trabalho gerada é inferior a carga de trabalho especificada. Já entre o intervalo de tempo de 20 e 40, percebe-se que há uma tentativa de compensação, talvez pelo atraso na rede e a sobrecarga de algumas das partes envolvidas. Posteriormente tem-se o

equilíbrio até o instante de tempo 50. Novamente, há uma variação seguido de um equilíbrio entre as cargas de trabalho comparadas. Nos instantes finais, após o instante de tempo 75, percebe-se uma geração inferior junto com um momento final de equilíbrio sem tentativa de compensação.

Figura 5 – Exemplo de comparação entre as cargas de trabalhos

Conforme visto no exemplo anterior, pode ser difícil quantificar o ruído dependendo da abordagem. Com a carga de trabalho sendo configurada individualmente em cada cliente é difícil saber quando uma requisição deve ser realizada, quando esta foi realizada, e por qual cliente ela foi realizada. Sendo assim, os possíveis problemas podem ocorrer:

- Requisição pode ser gerada corretamente e chegar no sistema alvo;
- Requisição pode ser gerada corretamente, mas não chegar no sistema alvo;
- Requisição pode ser gerada com atraso e chegar no sistema alvo;
- Requisição pode ser gerada com atraso e não chegar no sistema alvo;
- Requisição pode não ser gerada;

Obter essas informações em um experimento seria fundamental para avaliar e quantificar os ruídos que afetam a geração de carga de trabalho. Além disso, aplicar métricas que poderiam ser utilizadas para quantificar a diferença entre a característica da carga de trabalho que chega ao sistema alvo em relação a carga de trabalho que foi especificada. Assim, garantir que a carga de trabalho gerada seja exatamente igual a carga de trabalho especificada é uma tarefa não trivial. Contudo, é possível verificar a característica final da carga de trabalho gerada e imposta sobre o sistema alvo para ter uma análise sobre a carga real perante as métricas de desempenho.

Esta seção apresentou alguns problemas na geração de carga de trabalho dispostas na literatura. Face ao exposto, a próxima Seção descreve a arquitetura proposta considerando os pontos discutidos.

4.3 Arquitetura

A arquitetura desenvolvida tem como requisito básico gerar carga de trabalho distribuída em um sistema alvo para avaliação de sistema orientados a serviços. Nesse caso, três componentes estão presentes, sendo eles:

- **PEESOS-Cloud Server:** é um servidor que hospeda e gerencia o módulo principal da arquitetura. Todas as informações do experimento são configuradas neste módulo, como: carga de trabalho, serviço, clientes, sistema alvo e base de dados para manipulação de resultados.
- **Sistema alvo:** é o sistema alvo da avaliação que receberá as requisições dos clientes por meio de uma interface do serviço avaliado. Isso faz com que o ambiente do sistema alvo independa de tecnologias, basta fornecer uma URI para invocação do serviço. Métricas relacionadas a taxa de utilização dos recursos do sistema, tais como, CPU, memória e disco são mensuradas, bem como a possibilidade de uso de índices compostos de desempenho como apresentado em (SOUZA *et al.*, 2013).
- **Clientes:** são responsáveis por executar a carga de trabalho imposta sob o sistema alvo de avaliação simulando usuários reais e coletar métricas previamente definidas. Esses são alocados em *hosts* capazes de se comunicar com o servidor PEESOS-Cloud e com o sistema alvo. A carga é gerada por meio de requisições de um cliente a uma URI do serviço. As métricas coletadas podem ser o tempo de execução, o tempo de resposta, a taxa de requisições realizadas com sucesso e o resultado da requisição.

Para coordenação e gerenciamento das partes envolvidas, dois módulos foram desenvolvidos e organizados em camadas que atuam sob o ambiente. O primeiro módulo é o PEESOS-Cloud Server que executa em um servidor principal da arquitetura. O segundo é a aplicação PEESOS-Cloud Monitor que atua tanto nos clientes responsáveis por gerar carga quanto no sistema alvo da avaliação. Nos clientes, PEESOS-Cloud Monitor fornece uma interface de comunicação entre o cliente e o servidor PEESOS-Cloud Server, realizando atividades como implantar a aplicação cliente, que atua como um cliente para o serviço avaliado no sistema alvo. No sistema alvo, o PEESOS-Cloud Monitor gerencia a implantação do serviço e fornece informações do experimento ao módulo PEESOS-Cloud Server. A relação das entidades é representada na Figura 6.

Figura 6 – Abstração da arquitetura PEESOS-Cloud

A modelagem da arquitetura PEESOS-Cloud em dois módulos busca distribuir as tarefas de processamento e gerenciamento dos experimentos. Esta divisão atende a propriedade de baixo acoplamento da aplicação, uma importante característica para que a arquitetura proposta seja extensível a diferentes contextos e seja escalável. Além disso, o uso da aplicação cliente e de serviços a cada sistema alvo fornece flexibilidade em relação ao sistema a ser avaliado.

4.3.1 Modelo para geração de carga de trabalho distribuída

Além do modelo arquitetural para realização dos experimentos, é preciso definir como a carga de trabalho será aplicada pelos clientes. O modelo para geração de carga de trabalho distribuído deve ser genérico, tanto para cargas de trabalho sintéticas quanto para reais. Isso permite que diferentes cargas sejam utilizadas de acordo com o contexto. Além disso, esse modelo deve permitir verificar questões relacionadas a descaracterização da carga de trabalho independentemente do tipo de ruído presente.

Para isso, a proposta é baseada na decomposição de uma carga de trabalho a ser aplicada no sistema alvo em forma de série temporal. Independente da carga de trabalho ser sintética ou real, é possível expressar a chegada da mesma em um servidor em forma temporal contendo a quantidade de requisições por segundo. Dada uma série temporal que expressa a quantidade de requisições em instantes de tempo, a carga de trabalho w em um determinado instante de tempo t , representado por $w(t)$, possui uma demanda $d(t)$ com um possível ruído $n(t)$, podendo ser representada pela Equação 4.1.

$$w(t) = d(t) + n(t). \quad (4.1)$$

Com conhecimento a priori da demanda de requisições em um instante de tempo e o ruído a exploração de uma nova abordagem que permita gerar e avaliar uma carga de trabalho se torna possível. Com a necessidade de trabalhar com a característica da carga de trabalho imposta sobre o sistema alvo, a utilização de séries temporais permite estabelecer o número de requisições de acordo com o comportamento desejado. Por outro lado, é preciso decompor e sincronizar as requisições corretamente. Considere o fluxo de requisições apresentada na Tabela 2, em série temporal. Com isso, tem-se um conjunto de requisições por intervalo de tempo.

Tabela 2 – Carga de trabalho em forma de série temporal

Tempo	t_1	t_2	$t...$	$t_{(m-1)}$	t_m
Nº de Requisições	r_{t1}	r_{t2}	$r_{t...}$	$r_{t(m-1)}$	r_{tm}

A partir da série temporal, é realizada a decomposição em função da quantidade de requisições e de clientes. A Tabela 3 representa a quantidade de requisições e quantos clientes são necessários para representar a carga de trabalho em um instante de tempo t , e por último, qual cliente realizará requisição nestes instantes de tempo. Assim, a matriz designada “matriz de requisições”, com dimensão $m \times n$, tal que m é a quantidade total de requisições por instante de tempo t , dividida por um inteiro n clientes que irão separadamente executar o total de requisição. A matriz então é preenchida com 1 se há requisição do n -ésimo cliente. Caso contrário é preenchida com 0. A partir da matriz, cada cliente receberá a coluna c_n indicando em qual momento as requisições serão realizadas, onde cada linha representa os instantes de tempo e a demanda total de requisições para um determinado cliente.

Tabela 3 – Decomposição da carga de trabalho

Clientes/ Requisições/t	c_1	c_2	$c_{(...)}$	$c_{(n-1)}$	c_n
r_{t1}	$r_{t1,1}$	$r_{t1,2}$	$r_{t1,...}$	$r_{t1,(n-1)}$	$r_{t1,n}$
r_{t2}	$r_{t2,1}$	$r_{t2,2}$	$r_{t2,...}$	$r_{t2,(n-1)}$	$r_{t2,n}$
$r_{t(...)}$	$r_{t(...),1}$	$r_{t(...),2}$	$r_{t(...),...}$	$r_{t(...),(n-1)}$	$r_{t(...),n}$
$r_{t(m-1)}$	$r_{t(m-1),1}$	$r_{t(m-1),2}$	$r_{t(m-1),...}$	$r_{t(m-1),(n-1)}$	$r_{t(m-1),n}$
r_{tm}	$r_{tm,1}$	$r_{tm,2}$	$r_{tm,...}$	$r_{tm,(n-1)}$	$r_{tm,n}$

Ao decompor a série temporal da forma apresentada, pode-se utilizar uma carga de trabalho sintética ou real. Com o modelo para geração de carga distribuído entre os clientes, o próximo passo é integrar as partes envolvidas, conforme descrito na próxima Seção.

4.3.2 Integração dos módulos

A arquitetura proposta atua de forma independente e distribuída, sendo necessário tratar questões relacionadas à comunicação e ao sincronismo. Um servidor *Network Time Protocol* (NTP) é configurado em todo o ambiente. Isso não garante que o sincronismo seja perfeito, porém, mesmo com uma diferença, assume-se que as entidades estão sincronizadas. O Diagrama de

Sequência apresentado na Figura 7 descreve a ordem com que os eventos ocorrem nas entidades envolvidas.

Figura 7 – Diagrama de sequência da PEESOS-Cloud simplificado

É possível observar pelo diagrama de sequência que inicialmente tem-se um estágio que pode ser considerado como pré-experimento. Esse estágio ocorre até o evento de configuração da carga de trabalho em que os clientes e os provedores de serviços estão configurados. Desde então, o sincronismo se torna essencial para que o experimento execute conforme o esperado. É preciso que os clientes realizem as requisições no tempo adequado e os provedores estejam prontos para atender as requisições e mensurar a utilização dos recursos de desempenho do sistema. Todo esse processo é baseado em troca de mensagem utilizando *Sockets* e *JavaScript Object Notation* (JSON).

No módulo PEESOS-Cloud server, o pseudo-código 1 detalha o funcionamento do processo de comunicação entre os módulos da arquitetura. Como entrada ele recebe todas as informações do experimento, tais como, a carga de trabalho, o serviço, a aplicação cliente, o cliente e o provedores disponíveis para o experimento. Após a implantação do serviço e da aplicação cliente, é realizada a decomposição da carga de trabalho como explicada na Seção 4.3.1 referente ao modelo para geração de carga de trabalho distribuído. Em seguida, cada provedor recebe um tempo para monitorar os recursos. Posteriormente, cada cliente recebe sua carga de trabalho e esse mesmo tempo para que o experimento comece a ser realizado.

Algoritmo 1: Execução do experimento no servidor

```

input : Workload  $W$ , service  $s$ , applicationClient  $a$ , clientPool  $C$ , providePool  $P$ 
output : Experiment Result

1  /*Configura os clientes e provedores antes do experimento*/ for  $i \leftarrow 1$  to  $P$  do
2     $P_i \leftarrow \text{deploy } s \text{ provider};$ 
3  for  $i \leftarrow 1$  to  $C$  do
4     $C_i \leftarrow \text{deploy } a \text{ on the clients};$ 

5  /*Configura a carga de trabalho do experimento*/  $\text{matrixWorkload};$ 
6  for  $i \leftarrow 1$  to  $\text{size}(W)$  do
7    for  $j \leftarrow 1$  to  $W$  at instant  $i$  do
8       $\text{matrixWorkload}[i][j] \leftarrow 1;$ 

9  /*Atribui a carga de trabalho do experimento aos clientes*/ for  $i \leftarrow 1$  to
     $\text{maxValue}(W)$  do
10     for  $j \leftarrow 1$  to  $\text{size}(W)$  do
11        $C_i \leftarrow \text{matrixWorkload}[j][i];$ 

12 /*Sincroniza antes de iniciar o experimento*/  $\text{thresholdTime} \leftarrow 2000\text{ms};$ 
13  $\text{startTime} \leftarrow \text{currentTime} + (\text{thresholdTime} \times (P.\text{size}() + C.\text{size}()));$ 
14 for  $i \leftarrow 1$  to  $P$  do
15      $P_i \leftarrow \text{start the experiment } \text{startTime};$ 
16 for  $i \leftarrow 1$  to  $C$  do
17      $C_i \leftarrow \text{start the experiment } \text{startTime};$ 

```

O tempo de início de experimento é definido por meio de um cálculo envolvendo um limiar estabelecido e a quantidade de partes envolvidas, sendo esta a soma de provedores e clientes. Esse cálculo fornece um intervalo de tempo seguro para que nenhum cliente antecipe as requisições ou o provedor antecipe o monitoramento dos recursos. O valor desse limiar pode ser relacionado ao ambiente do experimento. Por exemplo, em um ambiente que todos os componentes da arquitetura estão hospedados localmente, um tempo de 2000ms pode ser o suficiente para troca de mensagens entre cada uma das partes envolvidas. Porém, ao considerar um ambiente distribuído geograficamente, como um experimento na nuvem, provavelmente que o tempo demandado seja maior. Durante a realização deste trabalho, esse número foi definido empiricamente pelos testes realizados nos experimentos. Caso este valor definido seja insuficiente em outros contextos, na análise dos resultados será possível identificá-los, pois os clientes e provedores iniciarão o experimento de forma assíncrona sendo possível verificar esse comportamento no *log* ao analisar os resultados.

O pseudo-código 2 do provedor é responsável por realizar a implantação do serviço no provedor, dada as configurações. Como por exemplo, implantar um arquivo *Web application Archive* (WAR) no diretório do servidor de aplicação. Após isso, inicializar o monitor de recursos considerando o tempo de início de experimento informado pelo PEESOS-Cloud Server.

Algoritmo 2: Execução do experimento no provedor

```

input : service  $s$ , startExperiment  $startTime$ 
output : Monitor Result

1 /*Configura os clientes e os provedores antes do experimento*/
2 while true do
3   switch commandServer do
4     case deployService
5       | Deploy  $s$  in the provider;
6     case StartExperiment
7       | Start monitor system in the  $startTime$ ;
8       | return results;

```

O pseudo-código 3, presente no PEESOS-Cloud Monitor dos clientes é responsável por implantar a aplicação cliente que realiza requisições ao sistema alvo e a carga de trabalho que será aplicada neste sistema alvo. Por último, são inicializadas as requisições considerando o tempo de início informado pelo PEESOS-Cloud Server.

Algoritmo 3: Execução do experimento nos clientes PEESOS-Cloud

```

input : Workload  $W$ , applicationClient  $a$ , startExperiment  $startTime$ 
output : Requests Result

1 /*Configura os clientes e os provedores antes do experimento*/
2 while true do
3   switch commandServer do
4     case deployApplicationClient
5       | Deploys the  $a$  on the client.;
6     case deployWorkload
7       | Deploys the  $W$  on the client;
8     case StartExperiment
9       | Start the experiment in the  $startTime$ ;
10      | return results;

```

A Figura 8 apresenta o fluxo de uma requisição e como as métricas são extraídas no sistema alvo. Ao observar o fluxo de uma requisição, o cliente da PEESOS-Cloud já possui parte da carga de trabalho que será a base para as requisições a serem realizadas e a aplicação cliente, que é um cliente para o serviço avaliado (por exemplo, um Cliente RESTful). De forma semelhante, o provedor possui o serviço implantado e um monitor de recursos capaz de mensurar as métricas definidas.

Assim, quando iniciado o experimento, as requisições são realizadas e os recursos mensurados. O primeiro tempo armazenado representa o instante em que a requisição é realizada no cliente. Em seguida, é armazenado o tempo que essa requisição chega ao serviço representada

Figura 8 – Fluxo de uma requisição e como as métricas são aferidas.

por “tempo de requisição no serviço”, assim como o tempo de execução da requisição no provedor de serviços. Por último, é armazenado o tempo de resposta final, que é quando a resposta final chega ao cliente. É importante destacar que todos esses tempos são mensurados em nível de aplicação, no caso do tempo de execução, esse é instrumentado no serviço. Ao final do experimento, todas as informações dos clientes e dos provedores são armazenadas na base de dados apresentada na próxima Seção.

4.3.3 Base de dados

Um banco de dados é necessário para armazenar informações dos experimentos para que as análises sejam realizadas. Para isso, um modelo lógico de dados foi definido e apresentado na Figura 9.

As tabelas presentes no modelo são descritas a seguir:

- **user:** armazena todas as informações de cadastro dos usuários do sistema;
- **experiment:** armazena informações de um experimento e todas as informações que o envolve;
- **workload:** armazena informações sobre as cargas de trabalho cadastradas no sistema disponíveis para serem utilizadas;
- **numberOfRequests:** armazena a quantidade de requisições por unidade de tempo de uma determinada carga de trabalho;
- **instance:** armazena informações das instâncias que podem ser utilizadas para um experimento;
- **monitorSystem:** armazena informações de desempenho de uma instância;

Figura 9 – Modelo entidade relacionamente base de dados PEESOS-Cloud

- **Service:** armazena os serviços disponíveis para serem utilizados;
- **LogRequest:** armazena informações sobre as requisições realizadas em um determinado experimento;

Para outros contextos, essa base de dados pode ser modificada conforme a necessidade. As informações podem variar de acordo com o tipo de sistema avaliado, bem como o objetivo da avaliação.

4.3.4 Protótipo

Um protótipo para avaliação da arquitetura proposta foi desenvolvido. A abordagem de desenvolvimento utilizada foi a prototipação, em que na primeira versão avaliações foram conduzidas e ajustes realizados, perfazendo o ciclo até que a proposta fosse contemplada (PRESSMAN, 2005). A Figura 10 representa a implantação da arquitetura composta pelos componentes.

Figura 10 – Funcionamento do protótipo implementado

Este cenário considera a disponibilidade de potenciais clientes. Desse modo, para avaliar um sistema os seguintes passos devem ser seguidos:

1. O usuário configura um experimento, por meio de *scripts*, definindo parâmetros como carga de trabalho a ser aplicada, serviço e sistema alvo da avaliação;
2. A aplicação PEESOS-Cloud Monitor do sistema alvo recebe informações do serviço a ser utilizado. Em seguida, verifica informações de como obter o serviço e então realiza a implantação no provedor de serviços de acordo com informações fornecidas pelo módulo PEESOS-Cloud Server;
3. Os potenciais clientes recebem informações sobre o experimento e então consultam um repositório para obter a aplicação cliente do respectivo sistema alvo;
4. De acordo com a característica da carga de trabalho, os clientes realizam as requisições para o sistema alvo;
5. Resposta da requisição e métricas de interesse do cliente tal como, tempo de execução sem rede, são enviadas para os clientes. Neste mesmo instante os provedores estão sendo monitorados em relação a taxa de utilização de recursos;

6. Nessa fase com o experimento já encerrado, o PEESOS-Cloud Monitor dos provedores enviam os resultados de monitoramento para o PEESOS-Cloud server;
7. Assim como nos provedores, o PEESOS-Cloud Monitor envia os resultados das requisições para o PEESOS-Cloud Server com o experimento já finalizado;
8. Todas as informações são armazenadas na base de dados, pronta para análise do experimento realizado.

Em todas as atividades o *feedback* das operações é avaliado para o controle de possíveis erros. O desenvolvimento do protótipo foi fundamental para verificação e ajustes da arquitetura.

4.4 Execução de um experimento

Nesta seção, um exemplo hipotético é utilizado para mostrar como a proposta deste projeto atua na avaliação de sistemas orientados a serviço considerando as características da carga de trabalho em função das métricas de desempenho. Como apresentado no Seção 2.4, a Avaliação de Desempenho envolve um conjunto de critérios que devem ser definidos para avaliar o desempenho de um sistema. Para esse exemplo, não foi realizado um planejamento de experimentos, apenas os dados de entrada de um único cenário é descrito e posteriormente demonstrado como os dados resultantes do experimento podem ser analisados.

Ao considerar um experimento com clientes já alocados, basta executar o PEESOS-Cloud Monitor em modo cliente. Os dados de entrada da Tabela 4 são fornecidos como entrada no PEESOS-Cloud Server.

Tabela 4 – Descrição e exemplo dos parâmetros de entrada para o experimento

Parâmetro	Descrição	Exemplo
Carga de trabalho	Unidade de tempo Vetor de requisições	1000 ms [3,4,5,3,6,7,0,6,5,1]
Sistema alvo	Dados da instância para implantação do serviço	IP: 10.0.0.100 Usuário: root Senha: 123 Diretório: /opt/tomcat/
Web Service	Serviço a ser implantado	fatorial.war
Aplicação Cliente	Cliente para o serviço implantado	cliente.jar

Após a execução dos experimentos os resultados são armazenados na base de dados onde informações podem ser obtidas por meio de consulta na base de dados. A Tabela 5 apresenta o *log* do experimento.

Primeiro, uma análise sobre o *log* das requisições do experimento deve ser realizada. De acordo com as informações presentes na base de dados do experimento, pode-se analisar as seguintes métricas:

Tabela 5 – Exemplo: Log das requisições do experimento provenientes da base de dados

Tempo da requisição no cliente	Tempo da requisição no serviço	Tempo de execução	Tempo de resposta	Status da resposta	Nome da máquina	IP
10:00:02.084	10:00:02.100	51	82	Ok	Desk-01	10.0.0.1
10:00:02.085	10:00:02.102	56	88	Ok	Desk-02	10.0.0.2
10:00:02.085	10:00:02.102	60	90	Ok	Desk-03	10.0.0.3
10:00:03.086	10:00:03.102	58	87	Ok	Desk-01	10.0.0.1
10:00:03.088	10:00:03.103	55	83	Ok	Desk-02	10.0.0.2
10:00:03.092	10:00:03.108	61	89	Ok	Desk-03	10.0.0.3
10:00:03.102	10:00:03.120	66	87	Ok	Desk-04	10.0.0.4
10:00:04.185	10:00:04.200	65	88	Ok	Desk-01	10.0.0.1
10:00:04.215	10:00:04.232	50	78	Ok	Desk-02	10.0.0.2
10:00:04.390	10:00:04.405	64	93	Ok	Desk-03	10.0.0.3
10:00:04.430	10:00:04.454	53	89	Ok	Desk-04	10.0.0.4
10:00:04.532	10:00:04.543	63	85	Ok	Desk-05	10.0.0.5
10:00:05.532	10:00:05.556	67	82	Ok	Desk-01	10.0.0.1
10:00:05.634	10:00:05.650	58	84	Ok	Desk-02	10.0.0.2
10:00:05.650	10:00:05.676	70	102	Ok	Desk-03	10.0.0.3
10:00:06.024	10:00:06.040	68	96	Ok	Desk-01	10.0.0.1
10:00:06.065	10:00:06.080	55	86	Ok	Desk-02	10.0.0.2
10:00:06.208	10:00:06.223	64	93	Ok	Desk-03	10.0.0.3
10:00:06.330	10:00:06.343	69	99	Ok	Desk-04	10.0.0.4
10:00:06.541	10:00:06.564	64	94	Ok	Desk-05	10.0.0.5
10:00:06.650	10:00:06.667	55	87	Ok	Desk-06	10.0.0.6
10:00:06.420	10:00:07.435	68	91	Ok	Desk-01	10.0.0.1
10:00:06.530	10:00:07.556	62	92	Ok	Desk-02	10.0.0.2
10:00:07.538	10:00:07.559	59	83	Ok	Desk-03	10.0.0.3
10:00:07.644	10:00:07.654	67	86	Ok	Desk-04	10.0.0.4
10:00:07.750	10:00:07.765	67	85	Ok	Desk-05	10.0.0.5
10:00:07.860	10:00:07.854	59	87	Ok	Desk-06	10.0.0.6
10:00:07.860	10:00:07.888	62	91	Ok	Desk-07	10.0.0.7
10:00:09.547	10:00:09.556	58	86	Ok	Desk-01	10.0.0.1
10:00:09.742	10:00:09.756	59	88	Ok	Desk-02	10.0.0.2
10:00:09.840	10:00:09.856	67	86	Ok	Desk-03	10.0.0.3
10:00:09.860	10:00:09.878	67	85	Ok	Desk-04	10.0.0.4
10:00:09.873	10:00:09.887	59	91	Ok	Desk-05	10.0.0.5
10:00:09.876	10:00:09.900	62	85	Ok	Desk-06	10.0.0.6
10:00:10.108	10:00:10.123	58	82	Ok	Desk-01	10.0.0.1
10:00:10.215	10:00:10.232	64	89	Ok	Desk-02	10.0.0.2
10:00:10.330	10:00:10.343	55	87	Ok	Desk-03	10.0.0.3
10:00:10.415	10:00:10.432	54	84	Ok	Desk-04	10.0.0.4
10:00:10.440	10:00:10.456	53	85	Ok	Desk-05	10.0.0.5
10:00:11.015	10:00:11.032	54	85	Ok	Desk-01	10.0.0.1

- **Tempo médio de execução e resposta:** Ambos os tempos podem ser analisados por meio da média e medidas de dispersão. O tempo de execução desconsidera o tempo da rede e o tempo de resposta é a soma do tempo de execução mais o tempo de rede. Analisar o tempo de resposta pode ser interessante em sistemas de tempo real que exigem baixa latência na rede. No exemplo utilizado, a Tabela 6 apresenta a média e o desvio padrão para o tempo de execução e de resposta. Ao comparar a média e o desvio padrão pode-se observar que a variação foi relativamente pequena. Isso caracteriza que o sistema se comportou bem com a carga de trabalho imposta. No entanto, em escalas maiores pode haver uma variação

maior durante a execução do experimento, como o caso de recursos alocados na nuvem. O uso de outras métricas baseadas em medidas de dispersão, *boxspot* e demais técnicas estatísticas e de matemática podem ajudar a compreender e definir o comportamento do sistema com maiores detalhes.

Tabela 6 – Média e desvio padrão do tempo de execução e de resposta

	Média	Desvio Padrão
Tempo de Execução	60,65	10,908
Tempo de Resposta	87,75	14,484

- **Característica da carga de trabalho:** Com as informações da Tabela 8 pode-se analisar a carga de trabalho em diferentes aspectos, tais como:

- **Carga de trabalho gerada pelos clientes:** Dado o tempo da primeira e última requisição, pode-se estabelecer qual o tempo de duração das requisições, para o exemplo utilizado, foi de 10 segundos. Ao agrupar as requisições a cada 1000ms, tempo entre as requisições definido do experimento, pode-se analisar a carga de trabalho que foi gerada pelos clientes. Neste caso, obtém-se a carga de trabalho exposta na Tabela 7. Isso leva a conclusão que nenhum cliente falhou na geração de carga. Os clientes alocados na PEESOS-Cloud podem ser diferenciados pelo nome e IP da máquina. Isso permite verificar qual cliente falhou caso ocorra a descaracterização da carga no momento da geração. Com isso, um estudo pode ser feito considerando a capacidade de cada máquina em hospedar um número n de clientes.

Tabela 7 – Carga de trabalho exemplo hipotético

Tempo (s)	1	2	3	4	5	6	7	8	9	10
Requisições/s	3	4	5	3	6	7	0	6	5	1

- **Carga de trabalho imposta sobre o sistema alvo:** Com as informações do tempo em que a requisição chegou no serviço, pode-se agrupar as requisições e obter o carga de trabalho resultante, que no exemplo considerado, é representado na Tabela 7. Apesar desse vetor mostrar que a carga imposta sobre o sistema alvo é idêntica a carga gerada pelos clientes, em outros cenários de larga escala dificilmente isso acontecerá. O uso da rede faz com que atrasos nas requisições ocorram e cheguem fora de ordem. Esse e outros problemas já foram discutidos anteriormente na definição do problema. Consequentemente, uma análise mais robusta dos dados deve ser feita como, por exemplo, avaliar o número médio de requisições considerando o tempo de duração do experimento, taxa de atendimento de requisições por segundo, comparar a distribuição da carga de trabalho quando essa já está bem caracterizada e considerar o tipo de carga imposta em função das métricas de desempenho do sistema. Essas e outras análises com base na literatura estatística e experimental podem ser considerados com intuito de obter conclusões de desempenho do sistema avaliado.

- **Status da resposta:** Como a arquitetura proposta é flexível, pode-se instrumentar o código com o objetivo de verificar o *status* da resposta de acordo com o serviço avaliado. Ao combinar essa métrica com tempo de execução e tempo de resposta, pode-se verificar a taxa de atendimento do sistema.

Com a análise das requisições e métricas de desempenho do sistema, o próximo passo é avaliar o comportamento do provedor de serviços. A Tabela 8 apresenta os dados obtidos no monitoramento dos recursos do provedor durante o tempo de experimento em que é possível observar o tempo de início das requisições em relação ao tempo de monitoramento dos recursos do provedor, sendo possível compará-los. Além das métricas consideradas neste exemplo, métricas de desempenho compostas sobre o sistema podem ser implementadas, como apresentado em (SOUZA *et al.*, 2013).

Tabela 8 – Consumo de recursos durante o tempo de experimento

Tempo (s)	Utilização dos recursos monitorados		
	CPU	Memória	Disco
10:00:01	2%	15%	10%
10:00:02	34%	16%	11%
10:00:03	46%	16%	11%
10:00:04	56%	16%	12%
10:00:05	32%	16%	12%
10:00:06	67%	16%	12%
10:00:07	74%	16%	12%
10:00:08	5%	16%	12%
10:00:09	62%	18%	12%
10:00:10	53%	18%	12%
10:00:11	14%	18%	12%
10:00:12	6%	18%	12%

Com os dados obtidos no *log* das requisições é possível associar algumas métricas, tais como, o tempo de execução e número de requisições atendidas, a taxa de utilização dos recursos do provedor. Assim, é possível realizar o cruzamento de informações que permita identificar cenários indesejáveis. Além disso, verificar como os recursos computacionais do sistema alvo se comportam durante o experimento, conforme a Tabela 9 que mostra a média e o desvio padrão da CPU, memória e disco mensurados durante o experimento.

Tabela 9 – Média e desvio padrão dos recursos mensurados

CPU		Memória		Disco	
Média	Desvio Padrão	Média	Desvio Padrão	Média	Desvio Padrão
40,45	23,27	16,50	1,02	11,63	0,64
44,30	21,55	16,50	1,02	11,63	0,64

Para a carga de trabalho exemplificada, o tempo de duração das requisições é de 10 segundos. Esse tempo é calculado por meio da quantidade de valores na série temporal e a escala de tempo utilizada, neste caso são 10 valores com intervalos de tempo de 1000ms. Sendo assim, os recursos são mensurados em um período anterior e posterior ao tempo de execução da carga. De acordo com o tipo do sistema avaliado, pode-se assumir de 10% a 20% antes e depois do tempo de monitoramento. Esses períodos são denominados como período de *warm-up* e *cool*

down que respectivamente, representam o tempo de aquecimento e resfriamento do sistema perante a carga de trabalho imposta. Para o cálculo apresentado foi desprezado o primeiro valor da Tabela 8 por que tempos não estavam inclusos no tempo de início das requisições, conforma a Tabela 5.

Esse exemplo hipotético mostra a quantidade de informações que podem ser extraídas de um único cenário considerado. A análise dessas informações possibilitam também amenizar os impactos do ambiente distribuído, como os problemas relacionados a sincronismo, comunicação e falhas das partes envolvidas. Quando realizado um planejamento de experimento completo, em que fatores e níveis são considerados, os vários cenários de experimentos fornecem informações para análise sobre o sistema avaliado sobre diferentes aspectos. Com isso, ajustes podem ser realizados e o sistema pode se comportar o próximo do desejado em relação a uma carga de trabalho imposta.

4.5 Considerações Finais

Este Capítulo discutiu os problemas que ocasionam ruídos na geração de carga de trabalho e propôs uma arquitetura para avaliação de sistemas orientados a serviço. Para desenvolvimento dessa arquitetura, denominada PEESOS-Cloud, foi preciso estabelecer um modelo para geração de carga de trabalho e uma forma de comunicação entre as partes envolvidas. Um protótipo foi desenvolvido para ser utilizado na avaliação experimental de todo o trabalho proposto. Por último, um exemplo hipotético foi apresentado com intuito de mostrar como métricas extraídas do sistema podem ser utilizadas na avaliação de desempenho de diferentes sistemas computacionais. O próximo Capítulo apresenta uma avaliação experimental contendo estudos de caso que demonstram a aplicação do projeto desenvolvido.

AVALIAÇÃO EXPERIMENTAL

5.1 Considerações Iniciais

Este Capítulo apresenta como a proposta desenvolvida permite avaliar um sistema orientado a serviço de acordo com a característica da carga de trabalho. A metodologia adotada no planejamento, execução e análise dos experimentos seguem (JAIN, 1991), no qual o modelo de experimentos utilizado foi o fatorial completo. Foram realizadas 10 replicações de cada experimento para se estabelecer um nível de confiança de 95%. Três cenários foram definidos variando a localização dos componentes da arquitetura em ambientes locais e distribuídos, permitindo verificar como a arquitetura proposta se comportava nesses ambientes. Em todos os cenários, o objetivo é avaliar o desempenho de uma aplicação que possui característica *CPU-Bound*, disponibilizada como serviço RESTful. Cada cenário possui um ambiente, planejamento de experimentos e análises específicas, sendo descritos nas respectivas subseções. Em todos os experimentos, a carga de trabalho foi analisada. Para isso, considerou-se a característica da carga de trabalho utilizada. Assim, ao executar os experimentos para todas as combinações e replicações, a característica da carga de trabalho resultante foi analisada e comparada com a da carga de trabalho utilizada. Por último, foi analisado o impacto na carga de trabalho sob a avaliação realizada.

5.2 Cenário I

Neste cenário, o ambiente de experimentos considerado é composto por um provedor e a arquitetura PEESOS-Cloud hospedada em uma rede local, como mostra a Figura 11.

A Tabela 10 apresenta os fatores e níveis utilizados no experimento. O fator relacionado à média de requisições por segundo condiz com a característica da carga de trabalho que possui duração de 300 segundos. Para cada um dos níveis, as cargas de trabalho que possuem médias

Figura 11 – Ambiente de experimentos do cenário I

aproximadas são apresentadas, respectivamente, nas Figuras 12(a), 12(b) e 12(c). Os valores dos níveis correspondentes a média de requisições por segundo foram passados como parâmetros para uma função de distribuição de Poisson. Desta forma, a carga de trabalho montada por essa distribuição de probabilidade, para caso, possui 300 segundos referentes a quantidade de replicação. O fator relacionado ao modo de geração de carga de trabalho refere-se a como os clientes da PEESOS-Cloud foram instanciados. No modo centralizado, todos os clientes eram hospedados em um único nó físico, isto é, a geração e carga foi realizada de maneira centralizada. Ao passo que no modo distribuído os clientes eram distribuídos em 5 máquinas. Cada cliente desempenhava o papel de realizar uma requisição por segundo. Assim, é possível verificar o desempenho da aplicação considerando a carga de trabalho em diferentes perspectivas.

Tabela 10 – Fatores e níveis dos experimentos

Fator	Níveis
Média requisições/s	15, 30, 45
Modo de geração de carga	Centralizado, Distribuído

As variáveis de resposta analisadas foram:

- **Carga de trabalho gerada pelos clientes:** Verificar se a carga de trabalho estabelecida foi gerada corretamente pela arquitetura PEESOS-Cloud;

Figura 12 – Cargas de trabalho utilizada no experimento.

- **Carga de trabalho processada pelo provedor:** Verificar a característica da carga de trabalho imposta sobre o sistema alvo. É importante destacar que a aferição é feita em nível de aplicação.
- **Tempo de execução no provedor:** Verificar o tempo de execução do serviço em diferentes níveis de carga;
- **Taxa de utilização média da CPU:** Verificar a utilização da CPU do provedor de serviço em diferentes níveis de carga;

A Tabela 11 apresenta a especificação da máquina virtual que hospedava o servidor de aplicação do serviço avaliado. A aplicação cuja característica é *CPU-Bound* calcula a operação fatorial do número 1000. Esse valor foi definido empiricamente por meio de testes preliminares, considerando principalmente os recursos computacionais utilizados.

O servidor físico que hospeda essa máquina virtual possui especificação de acordo com a Tabela 12.

O ambiente computacional em que o servidor PEESOS-Cloud e os clientes foram instanciados é descrito na Tabela 13.

Tabela 11 – Especificação do provedor de serviço no cenário I

Configuração do Provedor	
Processador	1 vCPU
Memória	1 GB
Sistema Operacional	Linux Ubuntu Server 14.04.1 64 bits LTS
Aplicações	Apache Axis2 1.6 com Tomcat 9.0

Tabela 12 – Especificação do servidor físico da MV no cenário I

Recurso	Especificação
Processador	Intel® Core™2 Quad Processor Q9400
Memória	8 GB RAM DDR3
Disco	HD 160GB Seagate SATA II
Sistema Operacional	Linux Ubuntu Server 14.04.1 LTS – 64 Bits
Switch	HP V1910-48G (JE009A) w/ 48 ports 10/100/1000Mbps

Tabela 13 – Especificação do ambiente da PEESOS-Cloud

Recurso	Especificação
Processador	Intel® Core™ I7 Processor 4790 3.60GHZ
Memória	32 GB RAM DDR3
Disco	HD 2TB SATA III 7200RPM
Sistema Operacional	Linux Ubuntu Server 14.04.1 LTS – 64 bits
Switch	HP V1910-48G (JE009A) w/ 48 ports 10/100/1000Mbps

5.2.1 Resultados

O gráfico da Figura 13 apresenta a carga de trabalho gerada pela arquitetura PEESOS-Cloud sob a aplicação avaliada. O eixo X mostra o modo de geração de carga agrupado pelas cargas de trabalho com as médias de requisições. Enquanto que o eixo Y apresenta a média de requisições por segundo com um intervalo de confiança de 95% de cada carga de trabalho em relação ao experimento.

É possível observar que quando o modo de geração de carga de trabalho é distribuído, 100% das requisições são geradas corretamente em todos os experimentos considerando o tempo total da carga de trabalho de 300 segundos. Por outro lado, quando o modo de geração de carga é centralizado, para as cargas com médias de 30 e 45, a característica do número médio de requisições por segundo é violado. Neste caso, para a carga de trabalho com média de 30 requisições por segundo gerada centralizada, uma carga inferior com média de 23,70 requisições por segundo foi gerada pelos clientes da PEESOS-Cloud. Já a carga de trabalho com média de 45 requisições por segundo de modo centralizado, uma carga também inferior, com média de 20,70 requisições por segundo foi gerada pelos clientes da PEESOS-Cloud. Em ambos os casos, foram considerados as requisições até que os recursos do nó físico exaurissem ocasionando na falha dos clientes da PEESOS-Cloud.

No gráfico da Figura 14 observa-se a carga de trabalho processada pelo provedor de

Figura 13 – Carga de trabalho gerada pela PEESOS-Cloud - Cenário I

Figura 14 – Carga de trabalho processada pelo provedor - Cenário I

serviços. Em relação a carga que foi gerada, isto é, que partiu dos clientes da PEESOS-Cloud, na maioria dos experimentos o provedor atendeu todas as requisições, exceto quando a carga

utilizada possui média de 45 requisições por segundo e é gerada de modo distribuído. Neste, as requisições não foram processadas de acordo com a carga processada no provedor do serviço na qual foi processado em média 7,68 requisições por segundo durante o tempo de monitoramento.

Ao observar a carga de trabalho com média de 15 requisições por segundo, a média da carga de trabalho processada foi de 14,44 e 14,52 requisições por segundo, para os modos de geração de carga centralizado e distribuído, respectivamente, sendo estatisticamente iguais. Ao considerar a carga de trabalho com média de 30 requisições por segundo, a média da carga de trabalho processada foi de 23,62 e 29,62 requisições por segundo, havendo superioridade para a carga gerada de forma distribuída. De maneira similar para carga com média de 45 requisições por segundo, a média foi de 20,69 requisições e 7,68 para as cargas processadas de forma centralizada e distribuída, respectivamente.

Além do uso da média de requisições por segundo, a Figura 15 apresenta as Função de distribuição acumulada (CDF) da carga de trabalho processada para cada experimento, contendo as replicações com as médias, desvio padrão e quantidade de amostras. É possível observar pela distribuição acumulada, exceto os experimentos representados nas Figuras 15(c), 15(e) e 15(f), que as cargas de trabalho apresentam um comportamento semelhante. As curvas das replicações e da carga especificada estão sobrepostas na área do gráfico em grande parte e aproximadamente 50% dos valores são inferior e os outros 50% superiores à média de requisições da carga de trabalho utilizada. Ainda nesses experimentos é possível observar o tempo de execução da carga, ou seja, o tempo total em que houve requisições onde a média variou pouco. De maneira geral, em todos os experimentos o tempo de processamento de toda a carga de trabalho no provedor foi um pouco maior do que os 300 segundos de carga gerada. Por isso, é importante destacar que mesmo a média das cargas de trabalho processadas sendo semelhantes as cargas de trabalho especificadas, há uma pequena diferença estatística. Ao considerar o tempo de 300 segundos da execução da carga de trabalho e a variação média do tempo de execução para as replicações, representadas pelo número de amostras, a diferença é inferior a 2,33%, calculado por meio do intervalo de confiança das replicações e a média da carga especificada.

Nos experimentos que apresentaram problemas na geração de carga, na quais as distribuições acumuladas estão presentes nas Figuras 15(c) e 15(e), é possível observar que houve uma maior variação. Neste caso, nota-se que quase 100% das requisições realizadas por segundo possuem valor inferior ao da média de requisições por segundo da carga utilizada. De maneira similar, na Figura 15(f) em que o provedor não foi capaz de processar as requisições tornando-se inoperante. Neste experimento, verifica-se também que o tempo de processamento de toda a carga de trabalho no provedor foi bem inferior aos 300 segundos, com uma pequena variação.

Dada a carga de trabalho que foi gerada e a carga de trabalho que foi processada pelo provedor de serviços, no gráfico da Figura 16 é possível observar o tempo de execução do serviço fatorial no provedor.

Para a carga de trabalho com média de 15 requisições por segundo, os tempos médios de

Figura 15 – CDF das cargas de trabalho processadas pelo provedor - Cenário I

execução do serviço foram de 128,50 e 131,50 milissegundos, para os modos de geração de carga centralizado e distribuído, respectivamente, sendo estatisticamente iguais. Ao considerar a carga de trabalho com média de 30 requisições por segundo, os tempos médios de execução do serviço foram de 312,06 e 375,34 milissegundos, sendo um tempo superior para a carga gerada de forma distribuída. Por último, ao verificar os tempos de execução para a carga que possui média de 45 requisições por segundo, esses foram de 321,20 e 7717 milissegundos, com uma diferença significativa entre os modos de geração de carga centralizado e distribuído, respectivamente.

Ao final, por ser uma aplicação *CPU-Bound* verificou-se a taxa média de utilização da

Figura 16 – Tempo de execução do serviço fatorial - Cenário I

CPU durante os experimentos, conforme apresentado no gráfico da Figura 17.

Para a carga de trabalho com média de 15 requisições por segundo, a utilização média da CPU foi de 73,10% e 72,82% sendo estatisticamente equivalente, para os modos de geração de carga centralizado e distribuído, respectivamente. Quando considerada a carga de trabalho com média de 30 requisições por segundo, a taxa de utilização média da CPU foi de 92,31% e 93,69%, para os modos de geração de carga centralizados e distribuídos, respectivamente, sendo estatisticamente equivalentes. Por outro lado, a taxa de utilização média da CPU para a carga de trabalho foi de 94,59% e 94,21%, para os modos de geração de carga centralizados e distribuídos, respectivamente, sendo também estatisticamente equivalentes.

5.2.2 Análise dos resultados

Os resultados apresentados possibilitam observar como o modo de geração de carga de trabalho pode interferir na característica final da carga. Ao avaliar a média, é possível identificar gargalos durante a geração da carga de trabalho considerando o modo centralizado e uma média de 30 e 45 requisições. Isso porque a sobrecarga no nó físico faz com que os clientes da PEESOS-Cloud responsáveis pela geração da carga falhem ao gerar as requisições. Ainda, é importante destacar que para os cenários em que houve a descaracterização devido a ruídos presentes na carga especificada, os tempos médios de execução foram estatisticamente equivalentes, enfatizando que a descaracterização da carga de trabalho ocorreu no momento da geração da carga. Além disso, há uma diferença notória ao comparar a descaracterização das

Figura 17 – Taxa de utilização média da CPU - Cenário I

cargas de trabalho especificadas em relação as cargas de trabalho processadas, diminuindo a média de requisições em 21,71% e 54,17% para as cargas de 30 e 45, respectivamente geradas de forma centralizada. Todos esses aspectos são confirmados pela CDF que caracteriza a carga de trabalho processada que já possui os ruídos em cada replicação, permitindo a comparação com as cargas especificadas.

A pequena diferença apresentada de 2,33% nas cargas geradas das especificadas, para os cenários em que não houve problema na geração da carga, ocorre devido um aumento no tempo de execução e monitoramento da requisições por segundo que o provedor processou. Neste caso, há uma pequena diferença na média da carga de trabalho processada devido as variações nas replicações. Independente do nível de carga, há uma tendência para que esse pequeno deslocamento ocorra. Todo o processo de execução da arquitetura envolve troca de mensagens e processamento das informações que é feito em nível de aplicação.

Ao observar o impacto das cargas de trabalho geradas sobre o provedor de serviço por meio da carga final processada, é possível notar, exceto para o cenário em que a carga com 45 requisições em média com geração de carga distribuída, que as cargas foram processadas conforme o esperado. No cenário específico da carga de trabalho com média de 45 requisições por segundo gerada de forma distribuída, o impacto da carga de trabalho foi maior do que o máximo suportado pelo provedor de serviços, que após um primeiro período experimental, não conseguiu responder às solicitações. Como resultado, apenas uma média de 17,01% das requisições foram atendidas. O alto valor no tempo de execução, neste caso, é causado pelo tempo de espera na

fila enquanto a requisição está sendo processada. Como cada pedido gera um processo Java que tem de ser escalonado para execução pelo sistema operacional, este experimento produziu um número maior de requisições simultâneos por segundo, que o sistema pode lidar.

Assim, o servidor de aplicação utilizado para oferecer o serviço mostrou-se ser instável, enquanto tais experimentos foram realizadas. Os recursos alocados em VM para a carga processada, resultou em fatores como *Timeout*. Além disso, a necessidade de mais recursos para atender a demanda de requisições simultâneas resultou numa falha da parte do provedor ao atingir o limite de processamento, em que apenas alguns dos 300 segundos de experimentos foram executados. Isto significa que é necessário adotar políticas de escalabilidade e técnicas de otimização.

Ao verificar a taxa de utilização média da CPU, para carga de trabalho com um menor número de requisições, neste caso média de 15 requisições por segundo, a taxa de utilização já apresentou-se elevada. Para a carga de 30 e 45 requisições por segundo, a utilização média da CPU foi equivalente estatisticamente. Apesar de essa taxa de utilização não chegar a 100%, é importante observar que tempos de *warm-up* e *cool-down* influenciaram sob as médias, principalmente quando a taxa de utilização média da CPU não foi de 100% para a carga de trabalho com média de 45 requisições por segundo gerada de forma distribuída. Isso acontece pela necessidade de considerar o tempo de execução de acordo com a última requisição processada, que pode ser analisada no *log*. Neste caso, esse período envolve o tempo de *cool-down* em alguns dos experimentos.

5.3 Cenário II

Neste cenário, o ambiente de experimentos considerado é composto por um provedor hospedado na Nuvem, os clientes e o servidor da PEESOS-Cloud hospedada em uma rede local. A Figura 18 apresenta o ambiente considerado.

O planejamento de experimentos é definido na Tabela 14.

Tabela 14 – Fatores e níveis dos experimentos

Fator	Níveis
Média de requisições/s	15, 30, 45
Modo de geração de carga	5 Nós, 10 Nós

O fator relacionado à média de requisições por segundo condiz com a característica das cargas de trabalho que possuem duração de 300 segundos. Essas já foram apresentadas nas Figuras 12(a), 12(b) e 12(c). O fator relacionado ao modo de geração de carga de trabalho refere-se a como os clientes da PEESOS-Cloud foram instanciados. Neste caso, os clientes foram divididos em 5 e 10 nós. Assim, é possível verificar se a quantidade de nós que realizam as requisições, sem que haja sobrecarga, impacta na avaliação de desempenho das aplicações. O

Figura 18 – Ambiente de experimentos do cenário II

número de 5 e 10 nós físicos foram definidos porque no Cenário I apresentado na subseção mostrou que com 5 nós físicos não há sobrecarga na geração das cargas de trabalho no contexto da infraestrutura e ambiente utilizado.

As variáveis de resposta analisadas foram as mesmas do Cenário I: Carga de trabalho gerada pelos clientes; Carga de trabalho processada pelo provedor; Taxa de utilização média da CPU; Tempo de execução no provedor.

O provedor utilizado na Nuvem foi alocado na Amazon EC2, em uma instância do tipo t2.micro¹ que possui como propósito o uso geral de aplicações. As especificações encontradas sobre a instância utilizada bem como dos servidores físicos da Amazon EC2 t2.micro são apresentadas Tabela 15.

Tabela 15 – Especificação da Amazon EC2 t2.micro

Recurso	Especificação
Processador do Servidor Físico	Intel Xeon 3.3 GHz
Processador MV t2.micro	1 vCPU
Memória MV t2.micro	1 GB
Disco	30 GB SSD

Apesar de informar a frequência e a fabricante do processador, mais detalhes não são fornecidos no site oficial da Amazon para instâncias do tipo t2.micro¹. O ambiente computacional

¹ <http://aws.amazon.com/pt/ec2/instance-types/>

em que o servidor PEESOS-Cloud e os clientes foram instanciados já foi descrito na Tabela 13.

5.3.1 Resultados

O gráfico da Figura 19 apresenta a carga de trabalho gerada pela arquitetura PEESOS-Cloud sob a aplicação avaliada. O eixo X mostra o modo de geração de carga agrupado pelas cargas de trabalho com as médias de requisições. Enquanto que o eixo Y apresenta a média de requisições por segundo com um intervalo de confiança de 95% de cada carga de trabalho em relação ao cenário.

Figura 19 – Carga de trabalho gerada pela PEESOS-Cloud - Cenário II

É possível observar que independente dos fatores e níveis definidos, 100% das requisições são geradas corretamente, considerando o tempo total da carga de trabalho de 300 segundos. Sendo assim, em nível de aplicação é possível observar que os recursos alocados para geração de carga foram suficiente em todos os casos.

No gráfico da Figura 20 observa-se a carga de trabalho processada pelo provedor de serviços. Em relação a carga que foi gerada, isto é, que partiu dos clientes da PEESOS-Cloud, todas as requisições foram atendidas. Porém, nos experimentos houve uma pequena variação entre 303 e 308 segundos durante o período de requisição. Apesar dessas pequenas variações, a média das cargas de trabalho possíveis de serem comparadas foram estatisticamente equivalentes independente do tipo de geração de carga e 100% das requisições foram processadas.

Figura 20 – Carga de trabalho processada pelo provedor - Cenário II

Na carga de trabalho com média de 15 requisições por segundo, a média da carga de trabalho processada foi de 14,53 e 14,51 requisições por segundo, para os modos de geração de carga centralizado e distribuído, respectivamente, sendo estatisticamente iguais. Ao considerar a carga de trabalho com média de 30 requisições por segundo, a média da carga de trabalho processada foi de 29,57 e 29,59 requisições por segundo, sendo estatisticamente equivalentes. De maneira similar para carga com média de 45 requisições por segundo, a média foi de 44,10 requisições e 44,18 para as cargas de trabalho processadas de forma centralizada e distribuída, respectivamente.

A Figura 21 apresenta as distribuições acumuladas para cada experimento e replicações deste cenário. Neste caso, é possível observar que em todos os cenários as cargas de trabalho apresentam um comportamento semelhante, com uma pequena variação. As curvas das replicações e da carga especificada estão sobrepostas em grande parte da área do gráfico e aproximadamente 50% dos valores são inferior e os outros 50% superiores à média de requisições da carga de trabalho utilizada. Quando observado os pontos extremos da carga de trabalho, isto é, carga com média de 15 e 45 requisições por segundo, nota-se uma pequena diferença na variação da cauda das curvas devido a diferença no desvio padrão. Observa-se que a média tem uma maior variação, ocasionando uma pequena diferença estatística. Sendo assim, há uma diferença inferior novamente, de aproximadamente 2,33%, calculado por meio do intervalo de confiança das replicações e à média das cargas processadas considerando o tempo da carga especificada de 300 segundos e a variação.

Figura 21 – CDF das cargas de trabalho processadas pelo provedor - Cenário II

Com a carga de trabalho processada sobre o provedor de serviços verificada, o gráfico da Figura 20 apresenta o tempo de execução do serviço no provedor da Amazon EC2. Para a carga de trabalho com média de 15 requisições por segundo, os tempos médios de execução foram de 40,38 e 43,70 milissegundos, para à geração de carga centralizada e distribuída, respectivamente. Quanto à carga de trabalho com média de 30 requisições por segundo, os tempos médios foram de 55,20 e 60,83 milissegundos para à geração de carga centralizada e distribuída, respectivamente. De maneira análoga, para a carga com média de 45 requisições por segundo, os tempos foram de 131,48 e 140,06 milissegundos. Independentemente do modo de geração de carga, todos os tempos de execuções agrupados pela média de requisições por segundos foram estatisticamente

equivalentes.

Figura 22 – Tempo de execução do serviço fatorial - Cenário II

Por último, é avaliada a taxa de utilização média da CPU durante todo o tempo de experimento. Para a carga de trabalho que possui média de 15 requisições por segundo, a taxa de utilização média foi de 34,01% para o modo de geração de carga centralizado e 35,54% para o modo de geração distribuído. Ao considerar a carga de trabalho com média de 30 requisições por segundo, a taxa de utilização média para o modo de geração de carga centralizado e distribuído foram de 44,12% e 46,75%, respectivamente. Por último, para a carga de trabalho com média de 45 requisições por segundo, a taxa de utilização média foi de 70,85% e 73,68% para o modo de geração centralizado e distribuído, respectivamente. Em todos os cenários considerando as cargas de trabalho com média iguais a taxa de utilização média foi estatisticamente equivalente.

5.3.2 Análise dos resultados

Ao considerar a carga de trabalho gerada pela arquitetura PEESOS-Cloud e a carga final recebida pelo provedor, essas cargas foram estatisticamente equivalentes apesar de ocorrer uma pequena variação na carga final no provedor. Com a utilização de um ambiente dinâmico, característica inerente da Internet, os resultados podem ser considerados satisfatórios. Isso faz com que o processo de avaliação da aplicação seja sólido, pois a carga processada foi semelhante à gerada no experimento, conforme apresentado na CDF de cada experimento. Porém, um aumento no tempo de execução de toda a carga de trabalho fez com que a média de requisições por segundo diminuísse, ocasionando novamente em uma diferença de 2,33%. Como já discutido

Figura 23 – Taxa de utilização média da CPU - Cenário II

anteriormente, o processo de troca de mensagens e o processamento das informações feito em nível de aplicação acaba gerando essa diferença.

É importante verificar que o fator relacionado ao modo de geração de carga não alterou a característica da carga, sendo as médias de requisições por segundo estatisticamente equivalentes. Esse fato chama atenção, pois com o uso da rede é possível que ocorra a descaracterização em níveis maiores. Para esse cenário de experimento, é possível verificar que podem ser utilizados 5 ou 10 nós para gerar as cargas de trabalho consideradas. Caso esses recursos fossem alocados na nuvem, a definição da capacidade de cada nó alocado ajudaria a reduzir custos de forma significativa. Pois neste caso, 50% dos recursos contratados seriam economizados, um número considerável para qualquer escala de experimentos. Para que fosse possível estabelecer limites de um nó a característica da carga de trabalho, a presença de rajadas na carga de trabalho, os recursos computacionais do nós, e até mesmo o ambiente de experimento teriam de ser considerados.

Em todos os experimentos, o tempo de execução do serviço foi inferior à 153 milissegundos com uma variação considerável. Em um tempo de execução tão pequeno, é preciso considerar a influência de toda arquitetura e camada de *software* presente no provedor de serviços. Entre essas que podem interferir em tempos de execução tão pequenos como os apresentados, é importante destacar a virtualização, JVM, Tomcat e diferentes níveis de cache. Além disso, o provedor é hospedado em uma nuvem pública, com a ideia de compartilhamento de recursos, onde a interferência da virtualização e estratégias de alocações de recursos podem influenciar no desempenho das aplicações.

5.4 Cenário III

Neste cenário, o ambiente de experimentos considerado é composto por um provedor hospedado na Nuvem e a arquitetura PEESOS-Cloud hospedada em uma rede local. Porém, a rede que interliga os clientes e o provedor é um *link* direto com capacidade de 10 Gb/s. O provedor foi hospedado na CloudUSP² localizada na Universidade de São Paulo na cidade de São Paulo SP - Brasil e os clientes numa rede local em outro Campus da USP localizado na cidade de São Carlos SP - Brasil, interligados pelo *link* da Universidade de São Paulo. A Figura 24 apresenta o ambiente considerado.

Figura 24 – Ambiente de experimentos do cenário III

Para esse cenário, variou-se a média de requisições por segundo das cargas de trabalho, sendo 15, 30 e 45 durante os 300 segundos de execução. Essas já foram apresentadas nas Figuras 12(a), 12(b) e 12(c), respectivamente. Os clientes da PEESOS-Cloud foram distribuídos em número fixo de 10 nós. As variáveis de resposta analisadas foram as mesmas do Cenários I e II: Carga de trabalho gerada pelos clientes; Carga de trabalho processada pelo provedor; Taxa de utilização média da CPU; Tempo de execução no provedor.

A Tabela 16 apresenta a especificação da máquina virtual que hospedava a aplicação. A mesma aplicação cujo característica é *CPU-Bound* foi utilizada, porém com a operação fatorial do número 3000.

² <http://cetirp.sti.usp.br/>

Tabela 16 – Especificação do provedor de serviço no cenário III

Recurso	Especificação
Processador	8 vCPUs
Memória	32 GB
Disco	50 GB
Sistema Operacional	Linux Ubuntu Server 14.04.1 64 bits LTS
Aplicações	Apache Axis2 1.6 com Tomcat 9.0

Já o servidor físico que hospeda essa máquina virtual possui especificação de acordo com a Tabela 17.

Tabela 17 – Especificação do servidor físico da MV no cenário III

Recurso	Especificação
Processador	Cisco UCS B230 M2 com 2 Processadores Intel® Xeon® E7-2800 10 Cores
Memória	512GB RAM DDR3 RDIMMs
Disco	Storage Netapp FAS3270C 15 PB
Sistema Operacional	Citrix XenServer 6.2
Switch	Cisco Nexus 7000 Series

5.4.1 Resultados

O gráfico da Figura 25 apresenta a carga de trabalho gerada pela arquitetura PEESOS-Cloud. O eixo X possui as cargas de trabalho, enquanto que o eixo Y, a média de requisições por segundo com um intervalo de confiança de 95% de cada carga de trabalho em relação ao cenário.

Figura 25 – Carga de trabalho gerada pela PEESOS-Cloud - Cenário III

Independente do nível de carga de trabalho considerado, todas as requisições foram geradas corretamente pela arquitetura PEESOS-Cloud. Ao observar a carga de trabalho processada,

como mostra o gráfico da Figura 26, as requisições foram processadas pelo provedor da mesma forma. Porém, há uma diferença inferior novamente, de aproximadamente 2,33%, calculado por meio do intervalo de confiança das replicações e à média da carga especificada.

Figura 26 – Carga de trabalho processada pelo provedor - Cenário III

Na carga de trabalho com média de 15 requisições por segundo a média da carga de trabalho processada foi de 14,55 requisições por segundo. Ao considerar a carga de trabalho com média de 30 requisições por segundo, a média da carga de trabalho processada foi de 29,57 requisições por segundo. Por último, para a carga de trabalho com média de 45 requisições por segundo, a média foi de 44,21 requisições por segundo.

A Figura 27 apresenta as distribuições acumuladas para cada experimento deste cenário. Em geral, todos os experimentos apresentam pequenas variações e todas as replicações convergem para uma mesma característica. Ao observar as curvas das replicações e da carga especificada, essas estão sobrepostas perante a área do gráfico. Além disso, aproximadamente 50% dos valores são inferior e os outros 50% superiores à média de requisições da carga de trabalho utilizada.

Após verificar a característica da carga gerada e processada, observa-se o tempo médio de execução no gráfico da Figura 28. O tempo médio de execução foi de 61,26 milissegundos para a carga de trabalho com média de 15 requisições por segundo, 83,51 milissegundos para a carga de trabalho com média de 30 requisições por segundo e 188,92 milissegundos para a carga de trabalho com média de 45 requisições por segundo.

5.4.2 Análise dos resultados

Os resultados apresentados mostram que para todas as cargas de trabalho geradas, uma carga equivalente foi processada, apesar da diferença inferior de aproximadamente 2,33% devido a pequena variação no tempo de processamento da carga de trabalho no provedor, calculado

Figura 27 – CDF das cargas de trabalho processadas pelo provedor - Cenário III

Figura 28 – Tempo de execução do serviço fatorial - Cenário III

por meio do intervalo de confiança das replicações e à média da carga especificada. O *link* de alta velocidade entre a USP-São Carlos e a USP-São Paulo possui ligação direta. Isso pode ter contribuído para que não houvesse ruídos na rede em o suficiente para descaracterizar a carga de trabalho.

Figura 29 – Taxa de utilização média da CPU - Cenário III

O tempo de execução para todos os cenários foi inferior a 198 milissegundos, com uma maior variação para a carga de trabalho com média de 45 requisições por segundo. As cargas inferiores a essa apresentam menor impacto no provedor de serviços. Para verificar isso, basta observar a taxa de utilização média da CPU, que apresenta um maior deslocamento para a um maior número de requisições por segundo.

5.5 Considerações Finais

Neste Capítulo foram apresentados alguns cenários em que a proposta deste trabalho foi aplicada. A avaliação experimental mostrou como a identificação de uma característica relacionada as cargas de trabalho permite revelar questões relacionadas a esse projeto, tais como problemas na geração da carga, presença de ruídos que à descaracterizasse, ausência e o desperdício dos recursos alocados. Além disso, por se tratar de um sistema distribuído, a arquitetura proposta identificou como os fatores inerentes nesses ambientes podem ter influência no desempenho. Quanto ao desempenho da aplicação, apesar dos cenários diferentes, o comportamento do tempo de execução foi parecido mesmo que em dimensões diferentes, em que a aplicação avaliada possui um comportamento exponencial. Isso corre devido a infraestrutura, ambiente e parâmetros da aplicação não serem os mesmos, o que dificulta a comparação em virtude da diferença de poder computacional.

CONCLUSÕES

O crescimento da computação em nuvem associado a um grande número de serviços oferecidos exige cada vez mais eficiência por parte dos provedores ao fornecer suas funcionalidades. Com a necessidade de projetar e avaliar serviços com a qualidade desejada, muitas propostas são apresentadas na literatura. Porém, um aspecto que tem sido negligenciado nessas propostas é como as cargas de trabalhos utilizadas nestes estudos são aplicadas. Este trabalho de mestrado teve como desafio identificar problemas presentes na literatura que fossem de suma importância ao planejar e executar experimentos em sistemas orientados a serviços.

O desenvolvimento desse projeto demonstrou como a carga de trabalho pode influenciar na avaliação desses ambientes. Assim, com objetivo de considerar o impacto da carga de trabalho durante uma avaliação, a arquitetura PEESOS-Cloud foi proposta. Tal arquitetura permite aplicar uma carga de trabalho durante um experimento e após a execução da mesma, realizar uma comparação com a carga inicial dando sustentação para a análise a ser realizada. Isso é possível devido ao modelo para geração de cargas de trabalho proposto e desenvolvido na arquitetura PEESOS-Cloud.

Uma avaliação experimental foi realizada com uma aplicação sintética em forma de serviço em diferentes cenários, considerando que a carga de trabalho pudesse ter impactos diferentes. Os resultados mostraram como a arquitetura PEESOS-Cloud abordou tais aspectos, enfatizando a importância de considerar a carga de trabalho final na avaliação das aplicações. Além disso, foi possível estudar limites de desempenho das aplicações de forma consistente evidenciando as contribuições esperadas.

Sendo assim, as principais contribuições alcançadas no desenvolvimento desse projeto são:

- Identificação e sumarização dos problemas originadores de ruídos na geração de cargas de trabalho distribuída. Com isso, novas propostas podem ser desenvolvidas.

- Projeto de uma arquitetura flexível e genérica a diferentes contextos, composta por módulos independentes e um modelo e geração de carga de trabalho. Essa pode ser implementada independente de tecnologias e linguagens.

6.1 Trabalhos Futuros

O desenvolvimento desse projeto de mestrado buscou atender aos requisitos que ocasionassem em uma proposta extensível e genérica, com possibilidade de aplicação em diversos contextos. Isso fez com que potenciais trabalhos futuros foram identificados. Alguns são citados e descritos a seguir:

- **Caracterização de cargas de trabalhos reais:** Na literatura, é possível encontrar diversos trabalhos que caracterizam cargas de trabalhos de aplicações que se tornam populares. A demanda para esse tipo de trabalho é constante, pois aplicações são lançadas frequentemente, podendo ter uma demanda alta e necessariamente vão precisar de ser escaladas. Alguns trabalhos utilizam uma metodologia para caracterizar uma carga de trabalho de uma aplicação modelando-a matematicamente, modelar o ambiente que essa aplicação está inserida, propor alternativas para melhoria de desempenho, e aplicar a carga de trabalho caracterizada sobre as alternativas e tomar uma decisão de melhoria. Para isso, é necessário uma solução que permita aplicar essa carga de trabalho caracterizada para realização do tradicional planejamento de capacidade. Neste contexto, a arquitetura PEESOS-Cloud pode ser aplicada a diferentes estudos relacionados de desempenho de aplicações. Um exemplo foi um trabalho realizado por (GONCALVES *et al.*, 2015) que caracterizou a carga de trabalho do Dropbox por meio de *logs*. Um gerador para essa carga de trabalho foi desenvolvido e utilizado na validação de uma proposta de cache com melhoria significativa na arquitetura do Dropbox. Porém, a solução geradora de carga não se preocupa com a qualidade da carga de trabalho desempenhada, sendo o problema abordado neste projeto de mestrado. O exemplo citado destaca o potencial de aplicação desse projeto em estudos semelhantes;
- **Visualização de dados:** A quantidade de informações fornecidas por um experimento faz com que a aplicação de técnicas de visualização de dados seja uma oportunidade. Assim, a detecção de anomalias e características presentes durante um experimento, tais como *outliers*, *cool-down* e *warm-up*, permitiriam uma visualização das informações de maneira transparente;
- **Mineração de dados:** As técnicas de aprendizado de máquina e agrupamento atuais poderiam ser utilizadas para auxiliar na avaliação e validação de novas propostas. Por exemplo, políticas de escalonamento e classificação de requisições nos provedores de serviços. Para que essas possam ser desenvolvidas e validadas, é preciso que uma carga

de trabalho seja caracterizada e aplicada sobre a aplicação, tornando possível utilizar a proposta aqui desenvolvida;

- **Desenvolvimento e validação de métricas compostas de QoS:** O desenvolvimento de novos índices de QoS que representem as aplicações de maneira mais realística é uma alternativa para estudar o desempenho das aplicações. Neste caso, uma alternativa possível para desenvolver e validar um índice de QoS composto é aplicar uma carga de trabalho com as características desejadas e mensurar as métricas adequadas;
- **Desenvolvimento de REA:** Os Recursos Educacionais Abertos (REA) são ferramentas utilizadas para o ensino e aprendizagem. Grandes grupos de pesquisas nas universidades brasileiras tem atuado na área de avaliação de desempenho, entre eles, o LaSDPC no ICMC-USP. Neste contexto, a arquitetura PEESOS-Cloud pode ser associada a materiais didáticos e aplicadas ao ensino na área de avaliação de desempenho. Em paralelo, o desenvolvimento de uma GUI associada a aplicações de técnicas de usabilidade utilizando heurísticas sólidas na literatura, tais como heurísticas de Nielsen ([NIELSEN, 1994](#));
- **Introdução a técnicas de tolerância a falha:** Alguns dos trabalhos apresentados e discutidos no Capítulo 3 estão preocupados com alocação de recursos com foco em prover tolerância a falhas na geração da carga. Durante o desenvolvimento desse projeto não houve a preocupação com tolerância a falhas. Porém, a proposta pode ser considerada em propor, avaliar e validar o desenvolvimento de técnicas de tolerância a falha. Com a implantação dessas técnicas é possível verificar a característica da carga final.
- **Desenvolvimento e avaliação da arquitetura PEESOS-Cloud em outras linguagens:** Com a ideia de validar a arquitetura proposta, o protótipo foi desenvolvido na linguagem Java. Porém, é interessante verificar a possibilidade de uma arquitetura mais robusta utilizando linguagens como a Erlang, que possui como forte característica a concorrência baseada em passagem de mensagens. Além disso, uma versão para *mobile* com foco em *Fog Computing* e a Internet das Coisas que realizam processamento em sensores e dispositivos móveis que podem atuar como provedores de serviço.
- **Modelos de negócios voltados para TaaS e BaaS:** Com a possibilidade de avaliar a característica da carga de trabalho e alocar recursos na nuvem sob demanda, uma oportunidade é desenvolver modelos de negócios que o consumidor paga pela qualidade da carga do serviço contratado. Além disso, o uso de técnicas de otimização para atribuição de tarefas podem ser utilizadas para minimizar os custos e minimizar os recursos utilizáveis preservando a geração da carga.

É importante ainda destacar que cada um desses trabalhos tem a viabilidade de ser desenvolvido em diferentes níveis, tais como, iniciação científica, mestrado e doutorado. Isso dependerá das técnicas e objetivos a serem alcançados.

REFERÊNCIAS

ALLIANCE, C. Security guidance for critical areas of focus in cloud computing v3. 0. **Cloud Security Alliance**, 2011. Citado 2 vezes nas páginas 15 e 8.

ALONSO, G.; CASATI, F.; KUNO, H.; MACHIRAJU, V. **Web Services: Concepts, Architectures and Applications**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 3642078885, 9783642078880. Citado na página 7.

BAI, X.; LI, M.; CHEN, B.; TSAI, W.-T.; GAO, J. Cloud testing tools. In: **Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on**. [S.l.: s.n.], 2011. p. 1–12. Citado na página 3.

BANZAI, T.; KOIZUMI, H.; KANBAYASHI, R.; IMADA, T.; HANAWA, T.; SATO, M. D-cloud: Design of a software testing environment for reliable distributed systems using cloud computing technology. In: **Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on**. [S.l.: s.n.], 2010. p. 631–636. Citado 2 vezes nas páginas 2 e 3.

BATISTA, B. G.; ESTRELLA, J. C.; FERREIRA, C. H. G.; FILHO, D. M. L.; NAKAMURA, L. H. V.; REIFF-MARGANIEC, S.; SANTANA, M. J.; SANTANA, R. H. C. Performance evaluation of resource management in cloud computing environments. **PLoS ONE**, Public Library of Science, v. 10, n. 11, p. e0141914, 11 2015. Disponível em: <<http://dx.doi.org/10.1371/journal.pone.0141914>>. Citado na página 3.

BLOOMBERG, J. **The Slow, The Crashed, The Out Of Stock: Black-Friday Fail Twitter Report**. 2015. [Online; posted November 27,2015]. Disponível em: <<http://www.forbes.com/sites/jasonbloomberg/2015/11/27/the-slow-the-crashed-the-out-of-stock-blackfriday-fail-twitter-report/4e00ddb87335>>. Citado na página 3.

BOOTH, D.; HAAS, H.; MCCABE, F.; NEWCOMER, E.; CHAMPION, M.; FERRIS, C.; ORCHARD, D. **Web Services Architecture**. 2004. World Wide Web Consortium, Note NOTE-
ws-arch-20040211. Citado na página 1.

BRANCA, G.; ATZORI, L. A survey of soa technologies in ngn network architectures. **Communications Surveys Tutorials, IEEE**, v. 14, n. 3, p. 644–661, Third 2012. ISSN 1553-877X. Citado 2 vezes nas páginas 15 e 6.

BUDAI, P.; GOLDSCHMIDT, B. Performance analysis of cloud-based application. In: **Large-Scale Scientific Computing**. [S.l.]: Springer, 2014. p. 476–483. Citado 2 vezes nas páginas 18 e 19.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. **Cloud Computing Principles and Paradigms**. [S.l.]: Wiley Publishing, 2011. ISBN 9780470887998. Citado na página 1.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In: **Proceedings of the**

10th International Conference on Algorithms and Architectures for Parallel Processing - Volume Part I. Berlin, Heidelberg: Springer-Verlag, 2010. (ICA3PP'10), p. 13–31. ISBN 3-642-13118-2, 978-3-642-13118-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-13119-6_2>. Citado na página 11.

CHHETRI, M.; CHICHIN, S.; VO, Q. B.; KOWALCZYK, R. Smart cloudbench – automated performance benchmarking of the cloud. In: **Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on.** [S.l.: s.n.], 2013. p. 414–421. Citado 4 vezes nas páginas 17, 18, 19 e 23.

CUNHA, M.; MENDONCA, N.; SAMPAIO, A. A declarative environment for automatic performance evaluation in iaas clouds. In: **Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on.** [S.l.: s.n.], 2013. p. 285–292. Citado 3 vezes nas páginas 17, 18 e 19.

DILLENSEGER, B. Clif, a framework based on fractal for flexible, distributed load testing. **annals of telecommunications - annales des télécommunications**, Springer-Verlag, v. 64, n. 1-2, p. 101–120, 2009. ISSN 0003-4347. Disponível em: <<http://dx.doi.org/10.1007/s12243-008-0067-9>>. Citado 4 vezes nas páginas 15, 16, 18 e 19.

ERL, T. **SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl).** Upper Saddle River, NJ, USA: Prentice Hall PTR, 2007. ISBN 0132344823. Citado 3 vezes nas páginas 5, 6 e 7.

ESTRELLA, J. C.; SANTANA, R. H. C.; SANTANA, M. J. **WSARCH: An Architecture for Web Services Provisioning with QoS Support - Performance Challenges.** [S.l.]: Saarbrücken : VDM Verlag Dr. Müller GmbH & Co, 2011. [Http://www.amazon.com/WSARCH-Architecture-Provisioning-Performance-Challenges/dp/3639378245](http://www.amazon.com/WSARCH-Architecture-Provisioning-Performance-Challenges/dp/3639378245). Citado na página 4.

FEITELSON, D. G. **Workload modeling for computer systems performance evaluation.** [S.l.]: Cambridge University Press, 2015. Citado na página 21.

FERREIRA, C. H. G.; RIBEIRO, J. B.; J., W. D. B.; ESTRELLA, J. C.; FILHO, D. M. L.; MACIEL, M. L. P. Identificação de gargalos de desempenho em ambientes virtuais para uso em computação em nuvem. In: **XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação.** [S.l.: s.n.], 2014. p. 13. Citado na página 10.

FURHT, B. Cloud computing fundamentals. In: **Handbook of cloud computing.** [S.l.]: Springer, 2010. p. 3–19. Citado 2 vezes nas páginas 9 e 10.

GONCALVES, G.; DRAGO, I.; SILVA, A. P. Couto da; VIEIRA, A. B.; ALMEIDA, J. M. Analyzing the impact of dropbox content sharing on an academic network. In: IEEE. **Computer Networks and Distributed Systems (SBRC), 2015 XXXIII Brazilian Symposium on.** [S.l.], 2015. p. 100–109. Citado na página 62.

HEAD, M. R.; KOCHUT, A.; SCHULZ, C.; SHAIKH, H. Virtual hypervisor: Enabling fair and economical resource partitioning in cloud environments. In: IEEE. **Network Operations and Management Symposium (NOMS), 2010 IEEE.** [S.l.], 2010. p. 104–111. Citado na página 11.

HSIEH, S.-J.; LUO, G.-H.; YUAN, S.-M.; CHEN, H.-W. A flexible public cloud based testing service for heterogeneous testing targets. In: **Network Operations and Management Symposium (APNOMS), 2014 16th Asia-Pacific.** [S.l.: s.n.], 2014. p. 1–3. Citado na página 23.

IGATE. **QA and Testing Services**. 2015. Disponível em: <<http://www.igate.com/industries/product-engineering-solutions/isv/offerings/services/qa-and-testing-services>>. Citado na página 15.

ITKO. **ITKO LISA Functional Testing**. 2015. Disponível em: <<http://www.itko.com/solutions/functional.jsp>>. Citado na página 15.

JAIN, R. **The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: Wiley, 1991. I-XXVII, 1-685 p. ISBN 978-0-471-50336-1. Citado 5 vezes nas páginas 12, 14, 21, 22 e 39.

JIANG, Z.; HASSAN, A. A survey on load testing of large-scale software systems. **Software Engineering, IEEE Transactions on**, PP, n. 99, p. 1–1, 2015. ISSN 0098-5589. Citado na página 2.

JONES, G. **Xbox One fans warned of new Xbox Live outages after December online issues**. 2014. [Online; posted December 7, 2014]. Disponível em: <<http://www.express.co.uk/entertainment/gaming/544198/Xbox-One-Microsoft-Xbox-Live-down-December-Christmas-offline>>. Citado na página 3.

KUEHNE, B. T.; ESTRELLA, J. C.; NUNES, L. H.; OLIVEIRA, E. Martins de; NAKAMURA, L. H.; FERREIRA, C. H. G.; SANTANA, R. H. C.; REIFF-MARGANIEC, S.; SANTANA, M. J. Awscs-a system to evaluate different approaches for the automatic composition and execution of web services flows. **PLoS ONE**, Public Library of Science, v. 10, n. 6, p. e0127677, 06 2015. Disponível em: <<http://dx.doi.org/10.1371/journal.pone.0127677>>. Citado 2 vezes nas páginas 2 e 7.

LEITE, D.; PEIXOTO, M.; BATISTA, B.; TARDIOLE, B.; FERREIRA, C.; SEGURA, D. A utilização da função affinity na manutenção de qos nos serviços de nuvem: uma comparação de desempenho entre os virtualizadores xen e kvm. In: **SBRC 2016 - XIV WORKSHOP DE COMPUTAÇÃO EM CLOUDS E APLICAÇÕES (WCGA)**. [S.l.: s.n.], 2016. Citado na página 10.

LOADIMPACT. **On Demand Load Testing for Developers e Testers**. 2015. Disponível em: <<https://loadimpact.com/>>. Citado na página 15.

LOADSTORM. **Load Test your app with LoadStorm to measure scalability**. 2015. Disponível em: <<http://loadstorm.com/>>. Citado na página 15.

MELL, P. M.; GRANCE, T. **SP 800-145. The NIST Definition of Cloud Computing**. Gaithersburg, MD, United States, 2011. Citado 4 vezes nas páginas 7, 8, 9 e 10.

MENASCÉ, D. A. Virtualization: Concepts, applications, and performance modeling. In: **Int. CMG Conference**. [S.l.: s.n.], 2005. p. 407–414. Citado na página 11.

MENASCE, D. A.; ALMEIDA, V. **Capacity Planning for Web Services: Metrics, Models, and Methods**. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130659037. Citado na página 2.

MENASCE, D. A.; DOWDY, L. W.; ALMEIDA, V. A. F. **Performance by Design: Computer Capacity Planning By Example**. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN 0130906735. Citado na página 22.

MORGAN, D.; HUMER, C. **Web traffic, glitches slow Obamacare exchanges launch**. 2013. [Online; posted October 1, 2013]. Disponível em: <<http://www.reuters.com/article/2013/10/01/us-usa-healthcare-idUSBRE98T14R20131001>>. Citado na página 3.

MOTAHARI-NEZHAD, H. R.; STEPHENSON, B.; SINGHAL, S. Outsourcing business to cloud computing services: Opportunities and challenges. **IEEE Internet Computing**, v. 10, 2009. Citado na página 7.

NIELSEN, J. **Usability engineering**. [S.l.]: Elsevier, 1994. Citado na página 63.

NUNES, L. H.; ESTRELLA, J. C.; FERREIRA, C. H. G.; NAKAMURA, L. H. V.; LIBARDI, R. M.; OLIVEIRA, E. M. de; KUEHNE, B. T.; SOUZA, P. S. L.; SANTANA, R. H. C.; SANTANA, M. J.; REIFF-MARGANIEC, S. Dca-services: A distributed and collaborative architecture for conducting experiments in service oriented systems. **International Journal of Services Computing**, v. 3, 2015. Disponível em: <<http://www.hipore.com/ijsc/2015/IJSC-Vol3-No4-2015-pp14-28.pdf>>. Citado na página 17.

NUNES, L. H.; NAKAMURA, L. H. V.; KUEHNE, B. T.; OLIVEIRA, E. M. de; LIBARDI, D. O.; RAFAEL, M.; ADAMI, L. J.; ESTRELLA, J. C.; REIFF-MARGANIEC, S. Peesos: A web tool for planning and execution of experiments in service oriented systems. In: **IEEE. Web Services (ICWS), 2014 IEEE International Conference on**. [S.l.], 2014. p. 606–613. Citado 4 vezes nas páginas 3, 17, 18 e 19.

OBERLE, T.; SZABO, C. An architectural prototype for testware as a service. In: **Applied Machine Intelligence and Informatics (SAMI), 2015 IEEE 13th International Symposium on**. [S.l.: s.n.], 2015. p. 15–19. Citado 3 vezes nas páginas 16, 18 e 19.

PAPAZOGLU, M. Service-oriented computing: concepts, characteristics and directions. In: **Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on**. [S.l.: s.n.], 2003. p. 3–12. Citado 2 vezes nas páginas 5 e 6.

PEIXOTO, M. L.; SANTANA, M. J.; ESTRELLA, J. C.; TAVARES, T. C.; KUEHNE, B. T.; SANTANA, R. H. A metascheduler architecture to provide qos on the cloud computing. In: **IEEE. Telecommunications (ICT), 2010 IEEE 17th International Conference on**. [S.l.], 2010. p. 650–657. Citado na página 8.

PERERA, C.; ZASLAVSKY, A. B.; CHRISTEN, P.; GEORGAKOPOULOS, D. Sensing as a service model for smart cities supported by internet of things. **CoRR**, abs/1307.8198, 2013. Disponível em: <<http://arxiv.org/abs/1307.8198>>. Citado na página 9.

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**. [S.l.: s.n.], 2013. p. 1–2. ISSN 2151-1349. Citado na página 2.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. [S.l.]: Palgrave Macmillan, 2005. Citado na página 32.

RIMAL, B.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. In: **INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on**. [S.l.: s.n.], 2009. p. 44–51. Citado 3 vezes nas páginas 7, 9 e 10.

SAHOO, J.; MOHAPATRA, S.; LATH, R. Virtualization: A survey on concepts, taxonomy and associated security issues. In: **Computer and Network Technology (ICCNT), 2010 Second International Conference on**. [S.l.: s.n.], 2010. p. 222–226. Citado na página 11.

SAROJADEVI, H. Performance testing: Methodologies and tools. **Journal of Information Engineering and Applications**, v. 2, n. 3, p. 53–61, 2012. Citado na página 3.

SERRANO, N.; HERNANTES, J.; GALLARDO, G. Service-oriented architecture and legacy systems. **Software, IEEE**, v. 31, n. 5, p. 15–19, Sept 2014. ISSN 0740-7459. Citado na página 1.

SMARTBEAR. **Software Testing, Monitoring, Developer Tools**. 2015. Disponível em: <<http://smartbear.com/products/testing/>>. Citado na página 15.

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**. New York, NY, USA: ACM, 2008. (CASCON '08), p. 11:144–11:156. Disponível em: <<http://doi.acm.org/10.1145/1463788.1463803>>. Citado na página 2.

SNELLMAN, N.; ASHRAF, A.; PORRES, I. Towards automatic performance and scalability testing of rich internet applications in the cloud. In: IEEE. **Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on**. [S.l.], 2011. p. 161–169. Citado 3 vezes nas páginas 16, 18 e 19.

SOASTA. **Load Testing and Performance Monitoring**. 2015. Disponível em: <<http://www.soasta.com/>>. Citado na página 15.

SOUZA, P. S. L.; SANTANA, R. H. C.; SANTANA, M. J.; ZALUSKA, E.; FAICAL, B. S.; ESTRELLA, J. C. Load index metrics for an optimized management of web services: A systematic evaluation. **PLoS ONE**, Public Library of Science, v. 8, n. 7, p. e68819, 07 2013. Disponível em: <<http://dx.doi.org/10.1371/journal.pone.0068819>>. Citado 5 vezes nas páginas 1, 3, 7, 25 e 37.

TCHANA, A.; DILLENSEGER, B.; PALMA, N. D.; ETCHEVERS, X.; VINCENT, J.-M.; SALMI, N.; HARBAOUI, A. Self-scalable benchmarking as a service with automatic saturation detection. In: EYERS, D.; SCHWAN, K. (Ed.). **Middleware 2013**. Springer Berlin Heidelberg, 2013, (Lecture Notes in Computer Science, v. 8275). p. 389–404. ISBN 978-3-642-45064-8. Disponível em: <http://dx.doi.org/10.1007/978-3-642-45065-5_20>. Citado na página 16.

TCHANA, A.; PALMA, N. D.; DILLENSEGER, B.; ETCHEVERS, X. A self-scalable load injection service. **Software: Practice and Experience**, John Wiley and Sons, Ltd, v. 45, n. 5, p. 613–632, 2015. ISSN 1097-024X. Disponível em: <<http://dx.doi.org/10.1002/spe.2236>>. Citado 5 vezes nas páginas 2, 16, 18, 19 e 23.

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**. [S.l.: s.n.], 2010. p. 358–365. ISSN 1062-922X. Citado na página 1.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds: towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, ACM, New York, NY, USA, v. 39, n. 1, p. 50–55, dez. 2008. ISSN 0146-4833. Citado 2 vezes nas páginas 8 e 9.

- WEISS, A. Computing in the clouds. **Computing**, v. 16, 2007. Citado na página 7.
- WHITE, J.; PILBEAM, A. A survey of virtualization technologies with performance testing. **arXiv preprint arXiv:1010.3233**, 2010. Citado na página 11.
- World Wide Web Consortium. Web services architecture. 2004. [Http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/](http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/). Citado na página 7.
- XI, H.; ZHAN, J.; JIA, Z.; HONG, X.; WANG, L.; ZHANG, L.; SUN, N.; LU, G. Characterization of real workloads of web search engines. In: **Workload Characterization (IISWC), 2011 IEEE International Symposium on**. [S.l.: s.n.], 2011. p. 15–25. Citado na página 22.
- YAN, M.; SUN, H.; WANG, X.; LIU, X. Ws-taas: A testing as a service platform for web service load testing. In: **Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on**. [S.l.: s.n.], 2012. p. 456–463. ISSN 1521-9097. Citado 2 vezes nas páginas 18 e 19.
- ZHOU, J.; LI, S.; ZHANG, Z.; YE, Z. Position paper: Cloud-based performance testing: Issues and challenges. In: **Proceedings of the 2013 International Workshop on Hot Topics in Cloud Services**. New York, NY, USA: ACM, 2013. (HotTopsCS '13), p. 55–62. ISBN 978-1-4503-2051-1. Disponível em: <http://doi.acm.org/10.1145/2462307.2462321>. Citado na página 23.