Mecanismos de autoconfiguração e auto-otimização
para arquiteturas virtualizadas que visam a provisão
de qualidade de serviço

Luis Hideo Vasconcelos Nakamura

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Luis Hideo Vasconcelos Nakamura

Mecanismos de autoconfiguração e auto-otimização para arquiteturas virtualizadas que visam a provisão de qualidade de serviço

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

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Marcos José Santana

USP – São Carlos
Junho de 2017

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

N163m

Nakamura, Luis Hideo Vasconcelos

Mecanismos de autoconfiguração e auto-otimização
para arquiteturas virtualizadas que visam a
provisão de qualidade de serviço / Luis Hideo
Vasconcelos Nakamura; orientador Marcos José Santana.
- São Carlos - SP, 2017.

217 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2017.

1. Computação em Nuvem. 2. Computação Autônoma.
3. Virtualização. 4. Ontologia. 5. Qualidade de
Serviço. I. Santana, Marcos José, orient. II. Título.

Luis Hideo Vasconcelos Nakamura

**Mechanisms of self-configuration and self-optimization for
virtualized architectures aiming at the provision of quality of
service**

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

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Marcos José Santana

USP – São Carlos
June 2017

Este trabalho é dedicado a todos aqueles que acreditaram e também tornaram possível a realização deste projeto. Em especial à Deus, a minha Esposa Ádria, meus pais Carmem e Pedro, aos meus irmãos Artur e Celia, ao meu orientador prof. Dr. Marcos Santana, aos professores, amigos, funcionários da USP/ICMC e a todos pesquisadores da área.

AGRADECIMENTOS

Agradeço inicialmente a Deus pela existência, por estar ao meu lado principalmente nos momentos mais difíceis, pelas oportunidades que colocou em meu caminho e por me conceder ânimo, força e coragem para superar os problemas e vencer mais uma etapa na minha vida. Agradeço a minha esposa, Ádria, que sempre me apoiou e incentivou durante esta jornada. Agradeço também a minha família, em especial aos meus pais, Pedro e Carmem, pelo carinho, apoio, paciência e compreensão.

Quero agradecer, em especial, ao meu orientador e amigo de romaria prof. Dr. Marcos Santana pela sua orientação e motivação como orientador. Pela sua dedicação e experiência transmitida como professor e amigo. E também pelo seu apoio em momentos de decisão. Quero agradecer aos professores: Regina Santana, Sarita Bruschi, Júlio Estrella, Paulo Souza, Jo Ueyama, Francisco Mônico, Cláudio Toledo e Stephan Reiff-Marganiec pelos conselhos e conhecimentos transmitidos que muito contribuíram com ideias e sugestões, sempre prontos para participar, colaborar e contribuir.

Agradeço muito aos meus amigos do LaSDPC (Laboratório de Sistemas Distribuídos e Programação Concorrente), pelo apoio, simpatia, hospitalidade e amizade. Um forte abraço a estes meus amigos, em ordem alfabética: Adriana Centurion, Ariel Dias, Bruno Guazzelli, Bruno Faical, Bruno Tardiolle, Carlos Ferreira, Dionísio Leite, Edvard Oliveira, Edwin Choquehuanca, Geraldo, Heitor, Leandro Mano, Leonildo Azevedo, Luiz Nunes, Paulo Eustáquio, Lourenço Pereira, Mário Machado, Maycon Leone, Pedro Prado, Renê Pinto, etc. Aos meus amigos e colegas da FATEC, Techs, CI&T e IBM que muito me incentivaram durante essa trilha que escolhi para minha vida.

Agradeço a Universidade de São Paulo e a todos os seus funcionários, especialmente aos do ICMC. Agradeço também ao Instituto Federal de São Paulo (IFSP) e aos seus servidores do câmpus Catanduva, pelo incentivo a auto-capacitação e todo o apoio prestado. Incluo nos meus agradecimentos, todos aqueles que contribuíram de alguma forma para a realização bem sucedida deste projeto.

Quero agradecer também à Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP - processo 11/12670-5) pelo apoio financeiro concedido para a realização deste trabalho.

*“Eu acredito, que às vezes são as pessoas que
ninguém espera nada que fazem as coisas que
ninguém consegue imaginar.”
(Alan Turing)*

RESUMO

NAKAMURA, L. H. V.. **Mecanismos de autoconfiguração e auto-otimização para arquiteturas virtualizadas que visam a provisão de qualidade de serviço**. 2017. 217 f. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

A proposta deste projeto de doutorado envolve a pesquisa sobre computação autônoma, focando na elaboração de mecanismos de autoconfiguração e auto-otimização para arquiteturas virtualizadas que buscam garantir a provisão de qualidade de serviço. Esses mecanismos fazem uso de elementos autônomos que são auxiliados por uma ontologia. Para isso, instrumentos de Web Semântica são utilizados para que a ontologia represente uma base de conhecimento com as informações dos recursos computacionais. Tais informações são utilizadas por algoritmos de otimização que, baseados em regras pré-definidas pelo administrador, tomam a decisão por uma nova configuração do sistema que vise a otimizar o desempenho. A configuração e a otimização geralmente envolvem elementos de *software* que precisam ser gerenciados pelos profissionais em Tecnologia da Informação (TI). Parte desse gerenciamento é composto de tarefas corriqueiras, por exemplo, monitorações, reconfigurações e verificações de desempenho. Tais tarefas demandam tempo e, portanto, geram custos e desgastes para os profissionais. Dessa forma, este projeto visa automatizar algumas dessas tarefas corriqueiras, facilitando o trabalho dos profissionais de TI e permitindo que eles foquem em tarefas mais críticas. Portanto, para alcançar esse objetivo foi realizado um estudo e a criação de mecanismos distribuídos baseados em Computação Autônoma e Web Semântica que permitem a configuração e otimização de recursos de forma automática. Os resultados individuais de cada mecanismo indicam que é possível alcançar um nível satisfatório de auto-configuração e auto-otimização para arquiteturas virtualizadas. O mecanismo de auto-configuração obteve melhores resultados com a abordagem de monitoração de recursos ao invés de utilizar previsões, já o mecanismo de auto-otimização provou que sua metodologia e algoritmo são aplicáveis na busca de uma configuração otimizada para atender ao SLA acordado.

Palavras-chave: Computação em Nuvem, Computação Autônoma, Virtualização, Ontologia, Qualidade de Serviço.

ABSTRACT

NAKAMURA, L. H. V. **Mechanisms of self-configuration and self-optimization for virtualized architectures aiming at the provision of quality of service.** 2017. 217 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2017.

The purpose of this PhD project involves the research about autonomic computing, focusing on the development of self-configuration and self-optimization mechanisms for virtualized architectures that aims to ensure the provision of Quality of Service. These mechanisms make use of autonomous elements that are aided by an ontology. Therefore, Semantic Web tools are used in order to allow the ontology to represent a knowledge base with information of the computational resources. Such information is used by optimization algorithms that take the decision of choosing a new configuration that aims at optimizing the architecture performance based on rules predefined by the administrator. The configuration and optimization usually involve elements of software that must be managed by professionals in the Information Technology (IT) field and part of this management is composed of common tasks, for example, monitoring tests, reconfigurations and performance evaluations. These tasks take some time and therefore generate costs and distress to the professionals. Thus, this project aims at automating some of these common tasks, facilitating the work of IT professionals and allowing them to focus on more critical tasks. Therefore, to achieve this goal a study was performed and distributed mechanisms based on Autonomic Computing and Semantic Web were created allowing the configuration and optimization of resources automatically. The individual results of each mechanism indicate that it is possible to achieve a satisfactory level of self-configuration and self-optimization for virtualized architectures. The self-configuration mechanism has achieved better results with the resource monitoring approach rather than using predictions and the self-optimization mechanism has proven that its methodology and algorithm are applicable in the search for an optimized configuration to meet the SLA agreed.

Keywords: Cloud Computing, Autonomic Computing, Virtualization, Ontology, Quality of Service.

LISTA DE ILUSTRAÇÕES

Figura 1 – Grafo dos componentes básicos de um arquivo RDF. (NAKAMURA <i>et al.</i> , 2011).	34
Figura 2 – Arquitetura da Web Semântica (BERNERS-LEE, 2000)	35
Figura 3 – Arquitetura alternativa da Web Semântica (ANTONIOU; HARMELEN F., 2008)	35
Figura 4 – Propriedades da Computação Autônoma – (Autogerenciamento) - Adaptado de (JACOB <i>et al.</i> , 2004).	43
Figura 5 – Arquitetura para um Elemento Autônomo (Adaptado de (KEPHART; CHESS, 2003) e (CHENG; LEON-GARCIA; FOSTER, 2008)).	44
Figura 6 – Mudança de paradigma da computação. Extraído de (VOAS; ZHANG, 2009).	50
Figura 7 – Modelos de Implementações de Nuvens baseada nas definições do NIST.	55
Figura 8 – Modelos de serviços da Computação em Nuvem - Adaptação de (BUYYA; BROBERG; GOSCINSKI, 2011) e (INTEL-CORPORATION, 2010).	57
Figura 9 – Comparação entre a virtualização completa e a paravirtualização. Adaptado de (CHANTRY, 2009) e (ANTONOPOULOS; GILLAM, 2010).	61
Figura 10 – A convergência de vários avanços em áreas que conduziram ao advento da Computação em Nuvem. Traduzido de (BUYYA; BROBERG; GOSCINSKI, 2011).	64
Figura 11 – Possíveis situações de provisão de recursos, capacidade de atendimento e demanda por serviços para uma empresa fictícia. Baseada em (ARMBRUST <i>et al.</i> , 2010).	66
Figura 12 – Modelo de referência conceitual da computação em nuvem na visão do NIST. Traduzida de (LIU <i>et al.</i> , 2011).	67
Figura 13 – Visão Abstrata da utilização dos mecanismos em uma arquitetura virtualizada.	86
Figura 14 – Adaptador Elástico e outros elementos para o gerenciamento automático de recursos computacionais virtualizados (DIAS, 2015).	88
Figura 15 – <i>Framework</i> FAMaV no contexto de um arquitetura em nuvem. (NEVES <i>et al.</i> , 2014).	92
Figura 16 – Hierarquia de Classes da IaaSOnt.	103
Figura 17 – Tipos de Gerenciamento propostos neste projeto (Management).	105
Figura 18 – Tipos de Políticas propostos neste projeto (Policy).	105
Figura 19 – Algumas propriedades de objetos (em azul) e dados (em verde) da IaaSOnt.	109

Figura 20 – Modelo Conceitual da arquitetura virtualizada utilizando os mecanismos (NAKAMURA <i>et al.</i> , 2014).	112
Figura 21 – Exp0 : Memória - Sem mecanismo de reconfiguração (Traduzido de (DIAS <i>et al.</i> , 2014)).	119
Figura 22 – Exp0 : Disco - Sem mecanismo de reconfiguração (Traduzido de (DIAS <i>et al.</i> , 2014)).	119
Figura 23 – Exp1 : Memória - Com mecanismo de reconfiguração baseado em monitoração (Traduzido de (DIAS <i>et al.</i> , 2014)).	120
Figura 24 – Exp1 : Disco - Com mecanismo de reconfiguração baseado em monitoração (Traduzido de (DIAS <i>et al.</i> , 2014)).	120
Figura 25 – Exp2 : Memória - Com mecanismo de reconfiguração baseado em previsão (Traduzido de (DIAS <i>et al.</i> , 2014)).	122
Figura 26 – Exp2 : Disco - Com mecanismo de reconfiguração baseado em previsão (Traduzido de (DIAS <i>et al.</i> , 2014)).	122
Figura 27 – Exp3 : Memória - Reconfiguração baseado em ambas abordagens (Traduzido de (DIAS <i>et al.</i> , 2014)).	123
Figura 28 – Exp3 : Disco - Reconfiguração baseado em ambas abordagens (Traduzido de (DIAS <i>et al.</i> , 2014)).	123
Figura 29 – Exp4 : FAMaV x Virsh - Funções de gerenciamento simples (Adaptado de (NEVES <i>et al.</i> , 2014)).	124
Figura 30 – Implementação da Função <i>Restart</i> .	125
Figura 31 – Exp4 : FAMaV x Virsh - Funções em listas (Adaptado de (NEVES <i>et al.</i> , 2014)).	125
Figura 32 – FAMaV x Virsh - Funções de gerenciamento complexas (Adaptado de (NEVES <i>et al.</i> , 2014)).	126
Figura 33 – Resultado do Tempo de resposta do modelo determinístico e do μ AG para um cenário com 10 até 100 VMs (Extraído de (NAKAMURA <i>et al.</i> , 2017)).	132
Figura 34 – Resultados da avaliação da parte de web semântica e ontologia.	133

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo Determinístico	99
Algoritmo 2 – Algoritmo μ AG	99

LISTA DE TABELAS

Tabela 1 – Ferramentas de virtualização em servidores (ANTONPOULOS; GILLAM, 2010).	61
Tabela 2 – Lista Resumida de Trabalhos Relacionados	84
Tabela 3 – Comparação entre as funções do FAMaV e da Virsh (NEVES <i>et al.</i> , 2014).	93
Tabela 4 – Etapas das Metodologias utilizadas como base para a criação da IaaSOnt (NOY; MCGUINNESS, 2001) (FERNÁNDEZ; GÓMEZ-PÉREZ; JURISTO, 1997)	101
Tabela 5 – Relação Atributo MachineCapacity e Capacity	104
Tabela 6 – Fatores e Níveis para os experimentos Exp1 , Exp2 e Exp3	117
Tabela 7 – Tempo de resposta e número de combinações obtidas com o algoritmo determinístico para a execução de 10 cenários de 10 até 100 VMs disponíveis (NAKAMURA <i>et al.</i> , 2017).	130
Tabela 8 – Tempo médio para alcançar soluções ótimas com o algoritmo μ AG (AZEVEDO, 2016) (NAKAMURA <i>et al.</i> , 2017).	131
Tabela 9 – Médias dos tempos de execução de funções de processamento da ontologia com web semântica (em segundos).	132
Tabela 10 – Médias dos tempos de execução das consultas SQWRL: <i>QueryClientByID</i> e <i>QueryClientByEmailAndPwd</i> (em segundos).	133
Tabela 11 – Médias dos tempos de execução das consultas SQWRL: <i>QueryClientAndVMsC-PUOverLoad</i> e <i>QueryClientAndQoSFromSLA</i> (em segundos).	133
Tabela 12 – Médias dos tempos de inclusão da regra SWRL: <i>RuleLoadedVM</i> (em segundos).	134
Tabela 13 – Tempo de resposta e taxa de sucesso para a população com 20 indivíduos.	163
Tabela 14 – Tempo de resposta e taxa de sucesso para a população com 10 indivíduos.	164
Tabela 15 – Tempo de resposta e taxa de sucesso para a população com 5 indivíduos.	164

LISTA DE ABREVIATURAS E SIGLAS

AE		<i>Autonomic Element</i>
AM		<i>Autonomic Manager</i>
API		<i>Application Programming Interface</i>
APIs		<i>Application Programming Interfaces</i>
AWS		<i>Amazon Web Services</i>
B2B		<i>Business-To-Business</i>
B2C		<i>Business-To-Customer</i>
BaaS		<i>Business as a Service</i>
CA		<i>Computação Autônoma</i>
CEO		<i>Chief Executive Officer</i>
CLI		<i>Command Line Interface</i>
DaaS		<i>Data as a Service</i>
DaaS		<i>[Development, Database, Desktop] as a Service</i>
DDoS		<i>Distributed Denial of Service</i>
DoS		<i>Denial of Service</i>
EA		<i>Elementos Autônomos</i>
EC2		<i>Elastic Compute Cloud</i>
EMP		<i>Erro Médio Percentual</i>
EQM		<i>Erro Quadrático Médio</i>
FaaS		<i>Framework as a Service</i>
FAMaV	..	<i>Virtual Machines Administration Tool</i>
GA		<i>Gerenciador Autônomo</i>
GAE		<i>Google App Engine</i>
HaaS		<i>Hardware as a Service</i>
HPC		<i>High Performance Computing</i>
IaaS		<i>Infrastructure as a Service</i>
IBM		<i>International Business Machine</i>
IC		<i>Iniciação Científica</i>
IPMaaS	..	<i>Identity and Policy Management as a Service</i>
JNA		<i>Java Native Access</i>
JVM		<i>Java Virtual Machine</i>

KB *Knowledge Base*
MAC Módulo de Auto-Configuração
MAO Módulo de Auto-Configuração
MB *MegaBytes*
MHW Modelo de Holt-Winter
MME Média Móvel Exponencial
MR *Managed Resource*
MVs Máquinas Virtuais
NaaS *Network as a Service*
NIST *National Institute of Standards and Technology*
OaaS *Organization as a Service*
OWL *Web Ontology Language*
PC *Personal Computer*
QoS *Quality of Service*
RDF *Resource Description Framework*
RDFS *RDF Schema*
RG Recurso Gerenciável
ROI *Return on Investment*
S3 *Simple Storage Service*
SaaS *Software-as-a-Service*
SLA *Service Level Agreement*
SO Sistema Operacional
SOA *Service Oriented Architecture*
SPARQL . *Simple Protocol and RDF Query Language*
SQL *Structured Query Language*
SWRL ... *Semantic Web Rule Language*
UMs Unidades monetárias
URI *Uniform Resource Identifier*
vCPU *virtual CPU*
VMM *Virtual Machine Monitor*
VPN *Virtual Private Network*
W3C *World Wide Web Consortium*
Web *World Wide Web*
XaaS *Everything as a Service*
XML *eXtensible Markup Language*

SUMÁRIO

1	INTRODUÇÃO	27
1.1	Considerações Iniciais	27
1.2	Contextualização	28
1.3	Motivação	29
1.4	Objetivo	30
1.5	Metodologia	30
1.6	Estrutura	31
2	WEB SEMÂNTICA E ONTOLOGIA	33
2.1	Considerações Iniciais	33
2.2	Web Semântica	33
2.3	Ontologia	35
2.3.1	A OWL - Web Ontology Language	36
2.4	Considerações Finais	38
3	COMPUTAÇÃO AUTÔNOMA	41
3.1	Considerações Iniciais	41
3.2	Embasamento Teórico	42
3.3	Conceitos Pertinentes ao Projeto	44
3.3.1	Autoconfiguração (Self-Configuration)	45
3.3.2	Auto-otimização (Self-Optimization)	46
3.4	Considerações Finais	46
4	COMPUTAÇÃO EM NUVEM	49
4.1	Considerações Iniciais	49
4.2	Embasamento Teórico	49
4.3	Modelos de Computação em Nuvem	54
4.4	Modelos de Serviços na Computação em Nuvem	55
4.5	Características da Computação em Nuvem	58
4.5.1	Características Essenciais	58
4.5.2	Virtualização	59
4.5.3	Computação Elástica	62
4.5.4	Computação Autônoma	63
4.5.5	Mercado e QoS	63

4.6	Arquitetura da Computação em Nuvem	66
4.7	Desafios da Computação em Nuvem	68
4.7.1	<i>Segurança</i>	68
4.7.2	<i>Escalabilidade</i>	69
4.7.3	<i>Provisionamento de serviços automatizados</i>	69
4.7.4	<i>Modelos de Custo e Cobrança</i>	69
4.7.5	<i>Acordos de nível de serviço (SLA)</i>	70
4.7.6	<i>Migração de Máquinas Virtuais</i>	70
4.7.7	<i>Questões políticas e jurídicas</i>	71
4.7.8	<i>Transparência e Dependência dos Provedores</i>	71
4.7.9	<i>Computação Verde</i>	71
4.8	Considerações Finais	72
5	TRABALHOS RELACIONADOS	73
5.1	Considerações Iniciais	73
5.2	Web Semântica	75
5.3	Computação Autônoma	78
5.4	Computação Autônoma, Computação em Nuvem e Web Semântica	81
5.5	Considerações Finais	84
6	MECANISMOS DE AUTOCONFIGURAÇÃO E AUTO-OTIMIZAÇÃO	85
6.1	Considerações Iniciais	85
6.2	Módulo de AutoConfiguração - MAC	86
6.2.1	<i>Adaptador Elástico</i>	87
6.2.1.1	<i>AutoConfiguração baseada em Monitoração</i>	88
6.2.1.2	<i>AutoConfiguração baseada em Previsão</i>	88
6.2.1.3	<i>Retorno do Investimento (ROI - Return on Investment)</i>	90
6.2.2	<i>Ferramenta de Administração de Máquinas Virtuais (FaMaV)</i>	91
6.3	Módulo de Auto-Otimização - MAO	93
6.3.1	<i>Metodologia para o cumprimento dos SLAs</i>	95
6.3.2	<i>Algoritmos de Otimização</i>	97
6.4	Componentes Comuns aos Módulos	100
6.4.1	<i>Ferramenta de balanceamento de carga</i>	100
6.4.2	<i>Módulo de Inferência (ModInf)</i>	100
6.4.2.1	<i>A Ontologia laasOnt</i>	100
6.4.2.2	<i>Regras e Consultas para a Ontologia laasOnt</i>	108
6.5	Organização e Funcionamento	112
6.6	Considerações Finais	113
7	AVALIAÇÃO DOS MECANISMOS	115

7.1	Considerações Iniciais	115
7.2	Avaliação de Desempenho dos Componentes do Mecanismo de Auto-Configuração	115
7.2.1	<i>Configuração dos Ambientes de Experimentos</i>	116
7.2.2	<i>Planejamento de Experimentos</i>	117
7.2.3	<i>Resultados e Análise</i>	118
7.3	Avaliação de Desempenho dos Componentes do Mecanismo de Auto-Otimização	127
7.3.1	<i>Configuração do Ambiente de Experimentos</i>	127
7.3.2	<i>Planejamento dos Experimentos</i>	127
7.3.3	<i>Análise dos Resultados</i>	130
7.4	Considerações Finais	134
8	CONCLUSÕES	135
8.1	Considerações Iniciais	135
8.2	Discussão e Conclusões	136
8.3	Contribuições	141
8.3.1	<i>Produção Científica</i>	143
8.3.2	<i>Colaboração em projetos de pesquisa</i>	144
8.4	Trabalhos Futuros	144
REFERÊNCIAS		147
ANEXO A	METODOLOGIA DE "TUNNING" DO ALGORITMO μ AG	163
ANEXO B	DOCUMENTOS DE ESPECIFICAÇÕES DA ONTOLOGIA	165

INTRODUÇÃO

1.1 Considerações Iniciais

A proposta da *World Wide Web* (Web) por Tim Berners-Lee em 1989, foi um evento propulsor para uma alteração mundial no modo de comunicação, integração, disponibilização de informações, execução de negócios, entretenimento, publicidade e até mesmo na forma das pessoas se relacionarem.

Antes da *Web*, a Internet era um reduto de pesquisadores ligados às universidades, governo e indústrias (TANENBAUM, 2003). Porém, a *Web* fez com que esse cenário fosse alterado rapidamente e ela alcançou vários países ao redor do mundo, representando atualmente um dos principais meios de comunicação e reunindo uma das maiores fontes de informações disponíveis. Desde então, todo contexto que envolve a *Web* vem passando por diversas alterações e melhorias desde os protocolos de comunicação até as aplicações que nela residem. A nova versão da *Web* foi batizada de *Web 2.0* e engloba as aplicações de nova geração com conteúdo dinâmico e provendo mais iteratividade com os usuários (MURUGESAN, 2007).

Todos esses avanços e a crescente popularização da *Web* atraíram a atenção de diversas empresas que passaram a investir nesse ambiente virtual. Além disso, diversas outras nasceram desse meio (por exemplo, o Google) e rapidamente fizeram sucesso gerando divisas milionárias. Reforçando ainda mais o apelo comercial envolvido na *Web*, Schroth e Janner (2007) fizeram a seguinte afirmação: "*No contexto da Web 2.0, basicamente todos os usuários da Web podem ser considerados como um cliente em potencial*" (SCHROTH; JANNER, 2007). De fato, a *Web* tornou-se um veículo eficiente de propaganda, venda de produtos e prestação de serviços. Diversas formas de negócios, como por exemplo, *Business-To-Customer* (B2C), *Business-To-Business* (B2B) e *Software-as-a-Service* (SaaS) estão presentes na *Web*.

A prestação de serviços pela Internet está sendo explorada por diversas vertentes, desde simples consultas em bancos de dados até o aluguel de *softwares* ou mesmo infraestruturas

completas. O fenômeno da Computação em Nuvem, discutido mais adiante neste trabalho, retrata tal tendência que está sendo alvo de constantes investimentos e pesquisas tanto no meio corporativo quanto no acadêmico. Contudo, há ainda questões de como determinar quais serviços, dentre vários serviços disponíveis, atendem da melhor forma as necessidades dos clientes. Em outras palavras, é um desafio descobrir quais são os serviços que oferecem a qualidade de serviço (*Quality of Service* (QoS)), mesmo que o cliente descubra o melhor serviço é preciso que haja um mecanismo que busque garantir o seu funcionamento de forma eficiente e automática, evitando ou pelo menos diminuindo a intervenção do administrador em tarefas corriqueiras como a realização de monitoramentos, avaliações e execução de atividades que visam o desempenho e o bom funcionamento dos serviços prestados.

1.2 Contextualização

Em 2001 a *International Business Machine* (IBM) anunciava por meio de um manifesto um desafio a toda comunidade de TI (Tecnologia da Informação) e a academia. Nesse manifesto foi feita uma comparação da computação autônoma com o sistema nervoso autônomo. A ideia considera que, da mesma forma que o sistema autônomo humano é capaz de nos liberar de tarefas vitais e funções de baixo nível, deixando o nosso cérebro livre para pensar em outras atividades, a computação autônoma poderia deixar os administradores de sistemas livres de muitas rotinas de gerenciamento e tarefas operacionais corriqueiras (GANEK; CORBI, 2003).

Em 2003, Kephart e Chess destacaram em seu artigo "*The Vision of Autonomic Computing*" (A Visão da Computação Autônoma) que a complexidade dos sistemas computacionais deve evoluir de forma a se tornar um pesadelo. O motivo dessa preocupação é que a maior interconectividade e integração dos sistemas parece estar se aproximando dos limites da capacidade humana de gerenciamento (KEPHART; CHESS, 2003). O gerenciamento de recursos pode envolver tanto o *hardware* como o *software*, como por exemplo: sistemas operacionais, redes cabeadas ou sem fio, servidores, banco de dados, roteadores, aplicações modulares, máquinas virtuais, *Web Services*, entre outros (ZHAO; GAO; DUAN, 2009).

Nesse contexto, diversos pesquisadores da indústria e academia deram início a estudos sobre computação autônoma e como empregá-la nos problemas atuais. Tais problemas exigem o controle e autogestão de sistemas computacionais como, por exemplo: Computação em Nuvem (*Cloud Computing*), Computação em Grade (*Grid Computing*), robótica, controle de sistemas de tempo real, nanotecnologia, etc. Nesse nicho de sistemas é possível acrescentar aqueles que precisam garantir a qualidade dos serviços prestados. Essa garantia envolve toda a infraestrutura de *hardware* e também todos os elementos de *software* empregados. Ambos devem estar funcionando de forma harmônica com o objetivo de fornecer a qualidade acordada entre a empresa prestadora e o cliente consumidor de serviços.

1.3 Motivação

Segundo Dobson et. al, (2010) a computação autônoma é citada por alguns críticos como sendo, simplesmente, uma síntese dos resultados de outras áreas. Porém, esses mesmos autores afirmam que este tipo de visão ignora a amplitude do conjunto de sistemas (DOBSON et al., 2010). Realmente, a computação autônoma pode envolver uma gama de conceitos de diferentes áreas, como por exemplo: sistemas distribuídos, tolerância a falhas, avaliação de desempenho, estatística, segurança, inteligência artificial, comunicação em redes, sensores, sistemas embarcados, teoria de controle, banco de dados, qualidade de serviço, etc. Contudo, esses conceitos devem estar envolvidos e relacionados entre si de forma que possam garantir o funcionamento autônomo de sistemas computacionais. Isso significa que não basta apenas agrupar e utilizar conceitos de diversas áreas, é preciso também relacioná-los de uma forma inspirada na biologia do sistema nervoso autônomo humano a fim de atingir os objetivos.

A computação autônoma gira em torno de soluções que atendam basicamente a quatro objetivos (KHALID et al., 2009): Autoconfiguração (*Self-Configuration*), Auto-otimização (*Self-Optimization*), Autocura (*Self-Healing*) e Autoproteção (*Self-Protection*). Esses objetivos são abrangentes e permitem que sejam empregados em várias áreas da computação. Logo, uma área em potencial que pode explorar tais conceitos é a grande área de sistemas distribuídos com um enfoque mais particular em arquiteturas que visam a provisão de QoS.

Desde a proposta inicial da computação autônoma até os dias atuais, houve uma grande evolução dos sistemas computacionais. Parte dessa evolução está relacionada ao avanço do *hardware* e das técnicas de virtualização que, quando relacionadas, podem garantir um melhor aproveitamento dos recursos disponíveis. O avanço nesses dois nichos da computação e em alguns outros como, por exemplo, o aumento da largura de banda e a flexibilidade no acesso às aplicações da Web por meio de dispositivos com recursos limitados e/ou móveis, favoreceram a criação de um novo paradigma ou modelo de computação que foi rotulado como "Computação em Nuvem".

De modo geral, o conceito da Computação em Nuvem envolve muito mais que o avanço e a combinação de recursos e técnicas computacionais, há um forte apelo econômico e comercial no uso de modelos de prestação e venda de serviços com um enfoque comercial destacando os benefícios desse tipo de computação. Este trabalho destina um capítulo completo sobre esse assunto, porém é importante salientar que os mecanismos de virtualização presentes na Computação em Nuvem permitem uma flexibilidade de configuração da infraestrutura. Dessa forma, arquiteturas virtualizadas e, conseqüentemente, as arquiteturas computacionais disponíveis na "Nuvem" (*Web/Internet*), podem ser dimensionadas visando uma melhor otimização da arquitetura como um todo. Contudo, surgem questões de quando é o melhor momento e como realizar esse dimensionamento computacional. Acrescenta-se ainda, questões que envolvem o atendimento a um cliente específico da arquitetura com a QoS adequada e também questões relacionadas aos custos e lucros tanto na óptica do cliente quanto na do provedor de serviço

respectivamente.

A infraestrutura computacional virtualizada e fornecida como um serviço - *Infrastructure as a Service* (IaaS) - concentra a principal área de estudo deste trabalho. No tocante à qualidade de serviço e questões financeiras ou econômicas há diversos desafios que se estendem desde configurações de limites e valores de atributos de qualidade definidos em contratos (Tempo de Resposta, Disponibilidade, Custo, etc.) até decisões que permitem determinar quais são os clientes mais importantes ou lucrativos para a prestadora de serviço (prioridades). Além disso, podem-se acrescentar políticas de atendimento às requisições dos usuários visando um melhor aproveitamento da infraestrutura e buscando um melhor retorno do investimento. Contudo, tais informações devem ser aglomeradas e processadas para auxiliar na tomada de uma decisão determinante na realização ou não dos processos de otimização e configuração da arquitetura.

1.4 Objetivo

A proposta deste trabalho é reunir tais informações em uma ontologia e, por meio de recursos e mecanismos da Web Semântica, conseguir inferir uma decisão que outrora era do prestador de serviço em conjunto com o administrador do sistema, ou seja, decisões financeiras e/ou técnicas simples serão automatizadas. Portanto, o objetivo deste projeto é a criação de mecanismos baseados em conceitos da computação autônoma que garantam a autoconfiguração e auto-otimização de arquiteturas virtualizadas que visam a provisão de qualidade de serviço. Os mecanismos de auto-otimização buscam encontrar a melhor configuração para atender as demandas e os mecanismos de autoconfiguração realizam a reconfiguração da infraestrutura de forma automática baseada em informações ou critérios. Dessa forma, pretende-se que tais mecanismos cumpram o seu papel de forma eficiente e com o mínimo de dependência humana e atendendo aos requisitos dos clientes de forma mais personalizada.

1.5 Metodologia

A metodologia adotada para o desenvolvimento deste projeto de doutorado envolveu inicialmente um estudo aprofundado sobre os principais conceitos da fundamentação teórica que embasou a proposta deste trabalho. Na sequência diversos trabalhos relacionado foram pesquisados na literatura disponível, visando ir de encontro ao estado da arte na área para realizar comparações entre os trabalhos e buscar o conhecimento necessário para cumprir os objetivos expostos atendendo a hipótese proposta. Por fim, na parte do desenvolvimento dos mecanismos e de suas avaliações houveram colaborações com outros trabalhos de mestrados e iniciações científicas que implementaram e avaliaram algumas das propostas e soluções previstas neste projeto de doutorado.

1.6 Estrutura

Os capítulos restantes desta tese de doutorado estão organizados da seguinte forma:

- No **Capítulo 2** são apresentados brevemente alguns conceitos básicos de Web Semântica e Ontologias. O conhecimento de ambos os assuntos é relevante para atingir os objetivos deste projeto de pesquisa. A explicação do conceito de ontologia é realizada de forma sucinta e utilizando-se um exemplo baseado em domínio de aplicação. É apresentada também uma breve revisão bibliográfica sobre Web Semântica, abordando os principais fundamentos e ferramentas relacionadas ao escopo deste projeto.
- No **Capítulo 3** é feita uma revisão sobre Computação Autônoma explorando os seus conceitos dos quais alguns são aplicados no contexto deste projeto.
- No **Capítulo 4** são apresentados os conceitos de Computação em Nuvem, como são definidos, os principais tipos de computação em nuvem, os seus modelos de serviços, exemplos, riscos e benefícios e outras considerações relevantes.
- O **Capítulo 5** é constituído por uma revisão dos trabalhos relacionados disponíveis na literatura, com o intuito de expor o estado da arte e elucidar ao leitor sobre como os temas abordados nesta pesquisa estão sendo explorados pela comunidade academia e indústria.
- O **Capítulo 6** é reservado para a discussão dos mecanismos propostos neste trabalho. Esse Capítulo também inclui a metodologia empregada no desenvolvimento da ontologia e uma apresentação dos componentes e as ferramentas utilizados na criação deste projeto.
- No **Capítulo 7** são discutidos alguns resultados de desempenho dos mecanismos de auto-configuração e auto-otimização. Portanto, nesse Capítulo são apresentados os ambientes de execução e os planejamentos dos experimentos, bem como a análise dos resultados obtidos.
- O **Capítulo 8** é destinado às conclusões resultantes do desenvolvimento deste projeto. Ainda são apresentadas as contribuições geradas para a área de pesquisa e também as propostas para trabalhos futuros visando melhorias e possíveis novas contribuições.
- Finalmente são apresentadas as **Referências Bibliográficas** utilizadas para a elaboração desta tese.

WEB SEMÂNTICA E ONTOLOGIA

2.1 Considerações Iniciais

Este capítulo apresenta uma breve revisão sobre Web Semântica e Ontologia que são assuntos relacionados a este projeto. É importante salientar que ambos os assuntos visam apenas a complementar o trabalho proposto fornecendo uma maneira de organizar e utilizar dados e metadados relevantes às questões de QoS e de tomada de decisão na parte administrativa dos mecanismos propostos. Portanto, apesar da importância desses assuntos nesses aspectos específicos, eles não concentram o foco deste trabalho, sendo utilizados apenas alguns dos recursos disponíveis nessas áreas de pesquisa para atingir alguns dos objetivos deste projeto de doutorado.

2.2 Web Semântica

Quando a Web Semântica foi proposta por Tim Berners-Lee et. al. em 2001, tinha como objetivo principal ser uma extensão da Web para, não apenas prover o conteúdo, mas também dar significado sobre aquele conteúdo para as máquinas. Dessa forma, a Web Semântica seria capaz de permitir que os computadores fossem capazes de “compreender” a semântica dos documentos e dados ([BERNERS-LEE; HENDLER; LASSILA, 2001](#)).

A *World Wide Web Consortium* (W3C) tem incentivado as pesquisas em Web semântica. Ela propõe uma visão na qual a informação seja entregue de forma explícita, proporcionando que as máquinas consigam processar e integrar informações de uma maneira mais fácil ([W3C, 2004b](#)).

Uma das primeiras propostas para representar as informações na Web foi sugerida pela W3C e consiste na utilização de um *framework* denominado *Resource Description Framework* (RDF). Esse *framework* permite adicionar informações sobre determinado recurso na Web, por

Figura 1 – Grafo dos componentes básicos de um arquivo RDF. (NAKAMURA *et al.*, 2011).

exemplo, uma página da Web. Nesse exemplo, um arquivo RDF é associado à página e ele possui informações sobre ela organizadas de forma que possam ser interpretadas semanticamente por agentes computacionais. Dessa forma, é possível adquirir não apenas o conteúdo, mas também informações capazes de atribuir uma semântica ao recurso (W3C, 2004b).

A estrutura da expressão em RDF é uma coleção baseada em triplas compostas por Sujeito, Predicado e Objeto, onde um conjunto de triplas é denominado grafo RDF (W3C, 2004b). A Figura 1 exibe os três componentes básicos de um arquivo RDF, no qual a Página 1 é o Sujeito, a Página 2 é o Objeto e uma propriedade em comum (*creator* – criador) é o Predicado. Isto implica que tanto a Página 1 quanto a 2 foram criadas pela mesma pessoa (*Luis Nakamura*) e há uma relação entre elas.

O RDF é baseado em *eXtensible Markup Language* (XML) o que permite a representação de relações entre objetos, possuindo melhores funcionalidades para representação semântica (NAKAMURA, 2012). Por esse motivo o RDF permite que agentes computacionais interpretem o significado dos dados por meio das *tags* associadas a eles (MARTIMIANO, 2006). O Esquema RDF (*RDF Schema* (RDFS)) define classes e propriedades que podem ser usadas para descrever as classes, propriedades e outros recursos. Portanto, ele é o responsável pela representação da respectiva semântica (W3C, 2004c).

Contudo, alguns pesquisadores notaram algumas limitações no RDF e RDFS. Por exemplo, Heflin (2001) relata que o RDF não possui um mecanismo para uma definição geral de axiomas (HEFLIN, 2001). Dessa forma, foi necessária a criação da linguagem *Web Ontology Language* (OWL), que estende o conceito do RDF para criação de ontologias.

A Figura 2 apresentada por Tim Berners-Lee em 2000 expõe uma arquitetura em pilha, denominada de "camadas do bolo" (*layer cake*) para explicar os diferentes protocolos e os desafios da Web Semântica (BERNERS-LEE, 2000). Uma das camadas mais importantes é a camada que representa a ontologia, pois é nela que está o conhecimento. A Figura 3 mostra um modelo atualizado de pilha da Web Semântica no qual as camadas "abstratas" propostas por Berners-Lee são representadas por algumas das atuais linguagens utilizadas para o desenvolvimento como,

por exemplo, a linguagem OWL para a criação de ontologias (NAKAMURA, 2012).

Figura 2 – Arquitetura da Web Semântica (BERNERS-LEE, 2000)

Figura 3 – Arquitetura alternativa da Web Semântica (ANTONIOU; HARMELEN F., 2008)

2.3 Ontologia

O emprego de ontologias para facilitar a representação das informações semânticas é adotado, evitando a redundância de informações e provendo uma representação formal de uma base de conhecimento. A criação e utilização de agentes computacionais capazes de interpretar as informações e realizar inferências é uma das grandes vantagens da Web Semântica, pois com a utilização de inteligência artificial para realizar inferências em regras existentes na ontologia é possível fazer com que estes agentes sejam capazes de responder a perguntas e a tomarem decisões. Outra vantagem da Web Semântica é que a representação do conhecimento não é centralizada como tipicamente é a maioria dos sistemas (BERNERS-LEE; HENDLER; LASSILA, 2001).

A informação que compõe uma Ontologia pode ser distribuída, pois os elementos que compõe a estrutura semântica (**Sujeito-Predicado-Objeto**) são representados por URIs¹. Por exemplo, pode-se ter um Sujeito de uma determinada base de conhecimento relacionado com o Objeto de outra; para evitar que exista a ambiguidade de dois elementos que possuem o mesmo significado cada um em uma base (exemplo: CEP e Código Postal) o uso de ontologias é essencial, pois através de uma representação formal é possível indicar que o elemento "CEP" da Base 1 está relacionado com o elemento "Código Postal" da Base 2 e deixar de forma explícita que esse relacionamento é equivalente. Dessa forma, o agente computacional (também conhecido como racionalizador - *reasoner*) "entenderá" que aqueles elementos de bases distintas são equivalentes. Diversos pesquisadores consideram as diversas vantagens geradas pela Web Semântica sendo

¹ URI (*Uniform Resource Identifier*) é um mecanismo de endereçamento e associação de nomes a recursos na Web (BERNERS-LEE *et al.*, 1998). Este mecanismo é único em toda a Web, por exemplo, a URL é um elemento deste mecanismo.

realizadas pesquisas em diversas vertentes², como por exemplo: na área de defesa militar³, comércio eletrônico⁴, combate ao câncer⁵, em institutos de padrões e tecnologia⁶, etc.

2.3.1 A OWL - Web Ontology Language

A OWL é capaz de prover maior interpretação do conteúdo da Web pelas máquinas (computadores) do que com XML, RDF e RDF *Schema*, disponibilizando um vocabulário adicional com uma semântica formal (W3C, 2004b). Com a OWL é possível adicionar vocabulários mais ricos para descrever as classes, os relacionamentos entre classes, comparação entre classes, restrições de cardinalidade e as características das propriedades (MARTIMIANO, 2006). O sucesso da linguagem OWL fez com que ela se tornasse o padrão utilizado pela comunidade de Web Semântica, especialmente para descrever ontologias (ISOTANI *et al.*, 2009).

A primeira versão da especificação da OWL foi aprovada como recomendação pela W3C em fevereiro de 2004 (W3C, 2004b) e incluía três tipos de linguagens (sublinguagens) da família OWL que são: *OWL Lite*, *OWL DL (Description Logics)* e *OWL Full* com diferentes níveis de expressividade (CHEN; WU; CUDRE-MAUROUX, 2012). Cada uma dessas subdivisões pode ser empregada em um propósito específico, basicamente a OWL Lite provê uma simples definição de hierarquia de classes e com poucas restrições de propriedades. A *OWL Full* é a mais completa, apresenta todos os construtores disponíveis sem restrição e é independente da sintaxe do RDF. Contudo, devido a sua abrangência, não há garantia que um *software* seja capaz de interpretar a ontologia criada com essa sublinguagem. A *OWL DL* é um meio termo entre a *OWL Lite* e a *OWL Full*, e está baseada em lógicas de descrição (DL). Essa sublinguagem de OWL oferece funcionalidades expressivas e permite o uso dos construtores da OWL, porém possui algumas restrições, mas mantém a completude computacional (capacidade de ser processada) e a tomada de decisão em um tempo finito (W3C, 2004b) (NAKAMURA, 2012). Detalhes de sintaxe e utilização dessa primeira versão podem ser encontrados em uma grande variedade de fontes como, por exemplo, na Internet (W3C, 2004b), trabalhos acadêmicos (NAKAMURA, 2012) e livros impressos (ANTONIOU; HARMELEN F., 2008).

A segunda versão da OWL é denominada OWL 2 e teve a sua recomendação aceita pela W3C em dezembro de 2012 (W3C, 2012a). Assim como a primeira versão da OWL, esta segunda versão também é subdividida em três perfis (*profiles*) que são sublinguagens destinadas a diferentes tipos de aplicações (W3C, 2012a) (CHEN; WU; CUDRE-MAUROUX, 2012). Cada um dos três perfis da OWL 2 (OWL 2 EL, OWL 2 QL e OWL 2 RL) representa uma restrição sintática da especificação estrutural da OWL 2, isso significa que, um subconjunto de elementos pode ser utilizado para proporcionar uma certa expressividade na ontologia de acordo com a

² Informações de acordo com o site <http://protege.stanford.edu/aboutus/aboutus.html>

³ <http://www.darpa.mil/>

⁴ <http://www.ebay.com/>

⁵ <http://www.nci.nih.gov/>

⁶ <http://www.nist.gov/index.html>

restrição de cada perfil que é mais restritiva que a OWL DL. Dessa forma, cada perfil da OWL 2 oferece um nível de expressividade em contrapartida a certos benefícios de implementação e capacidade computacional, ou seja, há um *trade-off* entre esses aspectos para cada perfil da OWL 2 (W3C, 2012a). Portanto, as características e objetivos das aplicações devem ser considerados na escolha dos perfis adequados. Por exemplo, a OWL 2 EL é interessante para aplicações em que a ontologia contém uma grande quantidade de propriedades e/ou classes. A OWL 2 QL é destinada às aplicações que utilizam grandes volumes de instância de dados e exigem respostas eficientes nas consultas que envolvem raciocínio ou inferências (*reasoning*). Por fim, a OWL 2 RL visa as aplicações que necessitam de um raciocínio baseado em regras de forma escalável (CHEN; WU; CUDRE-MAUROUX, 2012).

A segunda versão da OWL propõe preencher algumas lacunas reveladas na utilização prática de sua primeira versão com relação aos aspectos de sintaxe e expressividade. Resumidamente, as novas funcionalidades da OWL 2 oferecem melhorias em dois pontos (HIRANKITTI; XUAN, 2010):

1. **Sintaxe facilitada:** A OWL 2 provê novos construtores que facilitam a expressão de padrões comuns. Por exemplo, na OWL é preciso vários axiomas (*disjointWith*) indicando que uma determinada classe é disjunta de outras classes emparelhadas. Contudo, na OWL 2 é preciso apenas um axioma (*DisjointClasses*) para a mesma tarefa.
2. **Novos construtores para propriedades:** Diferentemente dos construtores da OWL que foram definidos baseados na DL SHOIN⁷ (HORROCKS; PATEL-SCHNEIDER, 2004), os construtores da OWL 2 foram definidos na DL SROIQ⁷ (HORROCKS; KUTZ; SATTLER, 2006) e, por este motivo, incluem novos construtores que expressam restrições adicionais sobre as propriedades, novas características às propriedades, incompatibilidade de propriedades, cadeias de propriedades e criação de chaves. Por exemplo, novos construtores foram adicionados a OWL 2 para a declaração de propriedades de objetos reflexivas (*ReflexiveObjectProperty*), irreflexivas (*IrreflexiveObjectProperty*) ou assimétricas (*AsymmetricObjectProperty*). A criação de chave (*key*) é outro recurso interessante provido pela OWL 2 que permite a criação de restrições de propriedades de dados, esse recurso é uma característica fundamental na tecnologia de banco de dados. A OWL 2 provê axiomas de chaves (*Hashkey*) que afirma que cada instância de uma classe é identificada exclusivamente por uma propriedade (de dado objeto) ou por um conjunto de propriedades, ou seja, se duas instâncias de uma classe têm os mesmos valores para as propriedades de chaves então esses dois indivíduos são o mesmo (W3C, 2012b).

⁷ Há diversas variedades de lógicas de descrição (DL - *Description Logic*) que, por convenção, recebem suas nomenclaturas de acordo com os símbolos dos conceitos e funções que implementam. O apêndice 1 do livro de Baader et. al (2003) introduz de uma forma compacta, porém com vários detalhes, a terminologia da lógica de descrição (BAADER et al., 2003). Entretanto, vale ressaltar novamente que este trabalho de doutorado não envolve a pesquisa nesta área, que é mais voltada para a parte de inteligência artificial e permite que ontologias sejam processadas, consultadas e inferidas. Contudo, a utilização de recursos providos por esses mecanismos serão explorados neste projeto.

Uma vasta documentação sobre a criação e características da OWL 2 e comparações com a primeira versão da OWL pode ser encontrada na literatura (GRAU *et al.*, 2008), (HIRANKITTI; XUAN, 2010) e uma lista completa dos novos recursos da OWL 2 é fornecida na especificação da W3C (W3C, 2012b).

Segundo, Antoniou e Harmelen, 2008, a camada de ontologia é instanciada com duas alternativas: a OWL e uma linguagem baseada em Regras (*Rules*) (ANTONIOU; HARMELEN F., 2008). As assertivas na linguagem OWL e regras podem ser utilizadas pelos agentes computacionais para racionalizar informações por meio de inferências. Esses agentes, também denominados na literatura como *reasoners*, são capazes de processar (“racionalizar”) e extrair informações de ontologias respondendo a questões que necessariamente não estão explícitas. O Pellet (SIRIN *et al.*, 2007), o FaCT++ (TSARKOV; HORROCKS, 2006) e o Hermit (OXFORD, 2015) são três exemplos de *reasoners* capazes de trabalhar com a linguagem OWL. A linguagem *Semantic Web Rule Language* (SWRL) (W3C, 2004) é utilizada para a criação de regras baseadas em OWL e permite que regras sejam expressas em termos de conceitos OWL provendo uma capacidade de raciocínio dedutivo mais poderoso do que quando utilizando apenas a linguagem OWL (PRACTICE, 2016).

Além da linguagem OWL, as regras e racionalizadores, outro elemento interessante é a utilização da linguagem de consulta *Simple Protocol and RDF Query Language* (SPARQL). Essa linguagem é utilizada para consultar informações armazenadas em arquivos RDFs (PRUD’HOMMEAUX; SEABORNE, 2008). Ela opera como algumas semelhanças a uma consulta *Structured Query Language* (SQL) sendo utilizada para realizar consultas relacionando diferentes bases de conhecimentos (ontologias) distribuídas. Há outros elementos, não são menos importantes, que podem ser envolvidos na criação de uma ontologia e fazem parte da Web Semântica como, por exemplo, a utilização de assinatura digital e criptografia para a garantia da segurança das informações (vide Figura 3).

2.4 Considerações Finais

O emprego de Web Semântica e ontologias vem sendo pesquisado nos últimos anos e muitos trabalhos foram desenvolvidos para diversos propósitos na computação (MAXIMILIEN; SINGH, 2004), (PAPAIOANNOU *et al.*, 2006), (TRAN, 2008), (AKLOUF; REZIG, 2009), (ZHOU; YANG; HUGILL, 2010), (NAKAMURA *et al.*, 2011), (GARCÍA; RUIZ; RUIZ-CORTÉS, 2012), (CHANG *et al.*, 2012), (LATRÉ *et al.*, 2013), etc. Conforme apresentado neste capítulo, diversas são as vantagens proporcionadas pela sua utilização, principalmente a capacidade de organizar dados e informações sobre esses dados (metadados) de uma forma organizada e formal (ontologia), mesmo que essas informações estejam distribuídas (*Uniform Resource Identifier* (URI)) e, conseguindo ainda, recuperar informações capazes de responder a diversas questões, inclusive por meio de inferências. Várias ferramentas, *frameworks*, *Application*

Programming Interfaces (APIs) e abordagens de utilização estão disponíveis para a realização de pesquisas, embora algumas ainda estejam em fase de amadurecimento. Contudo, os benefícios comentados anteriormente justificam a sua utilização.

O próximo capítulo aborda conceitos sobre a Computação Autônoma que podem fazer uso da Web Semântica para automatizar processos auxiliando no gerenciamento dos elementos de uma arquitetura que visa a provisão de QoS. Isso é possível pelo fato da Web Semântica permitir a representação do conhecimento de forma distribuída, por meio da linguagem OWL. Acrescenta-se ainda a possibilidade de consultar essas informações distribuídas, uma vez que a ontologia pode ficar disponível por meio de uma URI. Finalmente, recursos da Web Semântica podem garantir a tomada de decisões através da existência de agentes computacionais capazes de racionalizar as informações e, dessa forma, conciliar com as necessidades de um gerenciador autônomo presente em conceitos da Computação Autônoma.

COMPUTAÇÃO AUTÔNOMA

3.1 Considerações Iniciais

O termo "Computação Autônoma (CA)" é alvo de pesquisas desde sua proposta pela IBM em março de 2001. Na época Paul Horn, vice-presidente e diretor de pesquisa da IBM, apresentou esse tema e sua importância na Academia Nacional de Engenharia em Harvard (*Academy of Engineering at Harvard University*). Um mês depois, o vice-presidente de estratégia e tecnologia da IBM introduziu um novo projeto de computação autônoma nomeado "eLiza". Essas primeiras iniciativas deram início a uma série de discussões e pesquisas sobre a computação autônoma como solução para o rápido crescimento da complexidade de operação, gerenciamento e integração de sistemas computacionais (GANEK; CORBI, 2003).

Segundo (Dobson et. al, 2010) a preocupação da indústria de TI em 2001 poderia ser comparada ao que ocorreu com a indústria telefônica dos Estados Unidos na década de 20. Naquela época, a rápida expansão e infiltração do telefone no cotidiano das pessoas culminaram em uma série de preocupações, tais como a falta de operadores treinados para trabalhar manualmente nos quadros de distribuições. Os analistas estimavam que nos anos 80, a metade da população dos Estados Unidos teria que se tornarem operadores de telefone para atender a demanda. A solução para esse problema partiu da AT&T/Bell System, que implementou um protocolo de comutação automatizado e outras inovações tecnológicas para evitar a crise (DOBSON et al., 2010).

O problema da falta de profissionais preparados na área de TI ainda não foi completamente resolvido. Algumas estatísticas do trabalho nos Estados Unidos (*Bureau of Labor Statistics*) sugerem que há cerca de 260.000 trabalhadores na área de TI nos Estados Unidos, contudo esse número está em constante e leve declínio (DOBSON et al., 2010). Além disso, esta estatística relata apenas dados nos Estados Unidos e não leva em consideração que a demanda existente por esse tipo de profissional está sendo recrutada em outros países como Índia e Brasil.

Ou seja, muitas empresas de TI americanas estão utilizando mão de obra externa (seja em suas filiais ou por meio de empresas terceirizadas) para suprir a demanda por profissionais de TI.

Em 2001 havia a demanda por profissionais na indústria de TI e por volta de uma década depois esse problema ainda permanece. Nesse ponto, surge a dúvida de como a indústria de TI ainda não entrou em colapso. Uma das respostas pode estar no investimento de países em desenvolvimento como Brasil e Índia, esta última em especial, na criação de cursos e no desenvolvimento humano de profissionais.

Outra resposta para essa dúvida sugere que a utilização da Web 2.0 e a computação em nuvem (*Cloud Computing*) reduz a necessidade de gerenciamento por unidade de serviço (DOBSON *et al.*, 2010). De fato, a Web 2.0 engloba recursos que facilitam a interoperabilidade e o reuso entre sistemas como, por exemplo, os *Web Services*. A virtualização empregada na Computação em Nuvem é vantajosa para a configuração, reconfiguração, gerenciamento e manutenção de máquinas virtuais. Além disso, a Computação em Nuvem propõe um novo paradigma/modelo de negócio baseado na prestação de serviço sob demanda. Nesse modelo, empresas não precisam mais ter a preocupação com a área de TI (infraestrutura, processamento de dados, manutenção), pois toda ela estaria na “Nuvem” sendo gerenciada por uma empresa prestadora de serviço especializada.

Contudo, a solução de computação em nuvem não elimina a necessidade por profissionais de TI. Ela apenas reduz a quantidade e a centraliza nas grandes empresas prestadoras. A utilização dos recursos da Web 2.0 facilita, mas também não elimina a necessidade de gerenciamento. Portanto, uma alternativa utilizada para combater o problema de gerenciamento e escassez de profissionais na área de TI é por meio da pesquisa e utilização de computação autônoma.

3.2 Embasamento Teórico

A Computação Autônoma é um paradigma de computação cujo conceito está inspirado em sistemas biológicos que visam lidar com a administração de sistemas complexos oferecendo possibilidades de um autogerenciamento (*self-management*) minimizando a necessidade de intervenção humana (KHALID *et al.*, 2009).

A ideia da Computação Autônoma estabelece uma analogia ao sistema nervoso autônomo humano. Nesse sistema há o controle do batimento cardíaco, da temperatura corporal, do nível de açúcar, da pressão arterial, etc., de forma automática (SHEN *et al.*, 2009). Da mesma forma um sistema computacional autônomo deve ser capaz de gerenciar seus recursos, ou seja, ele deve ser capaz de se monitorar e tomar decisões que garantam o seu correto funcionamento. Baseada nessa ideia foram propostas quatro propriedades principais que permitem a autonomia de sistemas e a utilização de algumas delas garantem alguma forma de comportamento autônomo. Essas propriedades são as seguintes (KHALID *et al.*, 2009) (GANEK; CORBI, 2003):

- **Autoconfiguração (*Self-Configuration*)**: é a capacidade do sistema se configurar automaticamente de acordo com as políticas pré-definidas em alto nível e também a facilidade de se adaptar às mudanças causadas por configurações automáticas.
- **Auto-otimização (*Self-Optimization*)**: é a capacidade do sistema se automonitorar continuamente e controlar recursos para melhorar o desempenho e eficiência.
- **Autocura (*Self-Healing*)**: é a capacidade de o sistema detectar, diagnosticar e reparar falhas automaticamente.
- **Autoproteção (*Self-Protection*)**: é a capacidade de o sistema ser proativo em identificar e proteger a si mesmo de ataque maliciosos ou de falhas em cascata que não são corrigidas pelas medidas tomadas na autocura (*self-healing*).

A Figura 4 exibe estas propriedades apontadas pela IBM como necessárias para que um sistema computacional complexo alcance o autogerenciamento (*Self-Management*) (ZHAO; GAO; DUAN, 2009).

Figura 4 – Propriedades da Computação Autônoma – (Autogerenciamento) - Adaptado de (JACOB *et al.*, 2004).

Em um sistema autônomo há a interação entre Elementos Autônomos (EA) - *Autonomic Element* (AE) (KEPHART; CHESS, 2003). A arquitetura de elemento autônomo possui duas partes: Gerenciador Autônomo e Recurso Gerenciável (SHEN *et al.*, 2009). O Gerenciador Autônomo (GA) - *Autonomic Manager* (AM) é responsável pelo controle de recursos gerenciáveis, ele opera de acordo com um processo que estabelece um "laço de controle" (*control loop*) que gira em torno de quatro atividades: "Monitorar, Analisar, Planejar e Executar" (MAPE). Esse controle conta com o suporte do conhecimento sobre o ambiente computacional e de políticas de gerenciamento (CHENG; LEON-GARCIA; FOSTER, 2008). O conhecimento do atual estado do ambiente computacional é importante, pois, dependendo das políticas informadas pelos administradores do sistema, alguma decisão deve ser tomada a fim de garantir o comportamento desejado.

O Recurso Gerenciável (RG) - *Managed Resource* (MR) está dividido em duas categorias: *Hardware* e *Software* (KEPHART; CHESS, 2003). Na parte de *Hardware* um RG pode ser um

computador, uma impressora, um roteador, etc. Já na parte de *Software* o RG pode ser um *Web Service*, um serviço de diretório, um sistema operacional, uma aplicação, uma máquina virtual, etc. Um elemento em particular que pode ser gerenciado por meio de recursos de *Software* e *Hardware* é a Energia. Algumas funções e chamadas ao sistema especiais podem acessar os recursos ACPI⁸ disponíveis nas máquinas mais recentes para, por exemplo, diminuir o *clock* dos processadores quando estes estiverem ociosos com o intuito de economizar energia. Essa preocupação com o desperdício de recursos naturais propõe um novo paradigma de computação, a computação verde (*Green Computing*).

A comunicação entre o Gerenciador Autônomo e o Recurso Gerenciável ocorre por meio de uma interface denominada Interface de Gerenciamento (*Manageability Interface*). Essa interface é composta por um elemento **Sensor** e um elemento **Efector**⁹. O elemento Sensor é responsável por coletar informações do RG, atuando de forma parecida com um sensor passando informações para a parte de monitoração do GA. Por outro lado, o elemento Efetor tem a função de executar as operações enviadas pela parte de execução do GA no RG. A Figura 5 exibe uma referência para a arquitetura de um elemento autônomo.

Figura 5 – Arquitetura para um Elemento Autônomo (Adaptado de (KEPHART; CHESSE, 2003) e (CHENG; LEON-GARCIA; FOSTER, 2008)).

3.3 Conceitos Pertinentes ao Projeto

A teoria da computação autônoma orbita próxima a outras grandes áreas de pesquisa como a área de sistemas computacionais bio-inspirados e a área de sistemas adaptativos. Para

⁸ **Advanced Configuration and Power Interface (ACPI)** é um padrão de interface desenvolvido por empresas (Intel, HP, Microsoft, Toshiba, etc.) para permitir a configuração e gerenciamento de energia do computador.

⁹ O nome Efetor (em inglês: *Effector*) faz uma analogia na biologia ao órgão terminal de um nervo o qual distribui impulsos ativadores para a contração muscular ou para a secreção glandular.

cumprir com o seu objetivo a computação autônoma propõe que sistema computacional se adapte e, para isso, siga um comportamento semelhante ao sistema nervoso autônomo humano (bio-inspirado).

Contudo, apesar da sua relação com essas grandes áreas, os conceitos envolvidos na computação autônoma envolvem uma alta complexidade de implementação. A teoria baseada no sistema nervoso autônomo humano facilita o seu entendimento, porém tornar um sistema completamente autônomo não é uma tarefa trivial, exigindo planejamentos, etapas de desenvolvimento e configurações que demandam tempo.

Devido às restrições de indisponibilidade de recursos humanos suficientes e cronograma limitado, fica inviável construir um sistema autônomo completo para os propósitos deste projeto de doutorado. Embora a criação de uma arquitetura totalmente autogerenciável não seja o foco deste projeto, pretende-se criar mecanismos baseados em dois conceitos da computação que permitam facilitar algumas dessas tarefas.

Dessa forma, torna-se relevante detalhar e relacionar os conceitos de Autoconfiguração e Auto-otimização com o contexto deste projeto.

3.3.1 Autoconfiguração (Self-Configuration)

A autoconfiguração permite uma integração automática entre elementos autônomos na qual um elemento autônomo é capaz de detectar outro existente ou mesmo um novo elemento. Além disso, a configuração segue políticas de alto nível que são regulamentadas pelo Gerenciador Autônomo (HASSAN; AL-JUMEILY; HUSSAIN, 2009). Mecanismos de autoconfiguração podem estar presentes nos mais diversos e complexos sistemas computacionais. Por exemplo, um sistema operacional pode ser capaz de se autoconfigurar para economizar energia, para isso ele segue políticas de gerenciamento de energia. Tais políticas se comunicam com *hardware* via ACPI e geralmente adotam escalas de economia de energia, diminuindo a frequência do processador ou o brilho do monitor.

A autoconfiguração deve ter pelo menos um propósito que, dentro do escopo desse projeto, pode ser regido por políticas estáticas ou dinâmicas. Como exemplo de políticas estáticas, pode-se considerar a hibernação ou desligamento de um recurso ocioso (menos de 10% de utilização) para economia de recursos ou então aumentar a sua capacidade quando este recurso se tornar o gargalo do sistema (90% de utilização). As políticas estáticas podem ser definidas em tempo de projeto, durante a configuração dos mecanismos pelos administradores responsáveis em conjunto com os clientes/usuários do sistema.

As políticas dinâmicas não determinam um valor fixo para uma eventual autoconfiguração, um estímulo exige uma possível autoconfiguração informando o que precisa ser reconfigurado. Esse estímulo geralmente parte de outros mecanismos que também compõe o sistema autônomo, exigindo uma nova configuração para aumentar o desempenho (Auto-otimização), se

proteger contra ataques (Autoproteção) ou corrigir um problema (Autocura).

3.3.2 Auto-otimização (Self-Optimization)

A auto-otimização permite detectar comportamentos ideais, adaptando o sistema voluntariamente para a melhor configuração e é uma proposta muito eficaz na satisfação das necessidades do usuário (HASSAN; AL-JUMEILY; HUSSAIN, 2009). Mecanismos de auto-otimização buscam satisfazer determinadas necessidades do sistema de forma eficiente. Políticas e técnicas de otimização¹⁰ fazem parte desses mecanismos. As políticas indicam o que deve ser otimizado, quando essa otimização deve ocorrer e, adicionalmente, podem incluir restrições às técnicas de otimização que têm como objetivo auxiliar na resolução do problema.

Com relação a este projeto de doutorado, os mecanismos de auto-otimização devem a princípio maximizar o desempenho e minimizar os custos. Para isso, as políticas envolvidas devem estar relacionadas aos critérios de QoS e/ou economia financeira.

3.4 Considerações Finais

Há propostas que especificam um tipo particular de interação entre os diversos elementos autônomos a fim de garantir um autogerenciamento. Por exemplo, elementos de computação autônoma podem ser inspirados na biologia, como em colônias de formigas ou enxames de abelhas para coordenar o comportamento de robôs (DOBSON *et al.*, 2010). Dessa forma, nota-se que a computação autônoma abre um leque de oportunidades para a exploração e pesquisa envolvendo conceitos da mesma área (computação bio-inspirada) ou de diferentes áreas (robótica, inteligência artificial, sistemas distribuídos, etc.).

Outra proposta é a utilização da computação autônoma no processo educacional, mais especificamente na criação de ferramentas para projetos de aprendizagem (*Learning Design*), como é o caso de um *framework* baseado em princípios da Web Semântica e Computação Autônoma. Esse *framework* emprega mais especificamente a propriedade *autoconfiguração* para gerenciar e explorar a semântica dos conceitos utilizados na criação de um projeto de aprendizagem (CHARLTON; MAGOULAS, 2010). Portanto, a computação autônoma em sua proposta original visa facilitar e automatizar o gerenciamento de sistemas complexos.

Um exemplo de sistema complexo é uma arquitetura que tem como objetivo prover qualidade de serviço em arquiteturas virtualizadas na nuvem. Diversos são os desafios para garantir a autoconfiguração e a auto-otimização nesse tipo de arquitetura. Por exemplo, há diversos elementos que precisam ser listados e agentes autônomos devem ser associados a eles. Além disso, é preciso encontrar uma forma de representar o conhecimento e tomar decisões com

¹⁰ Para este trabalho entende-se *técnica de otimização* como sendo tanto modelos matemáticos formais quanto algoritmos de otimização.

base nesse conhecimento, isso pode ser feito com os recursos da Web Semântica, assunto visto no capítulo anterior.

Neste capítulo foram abordados os conceitos gerais da Computação Autônoma, sendo importante relembrar que para a criação de um sistema completamente autônomo é preciso a integração dos quatro conceitos envolvidos (Autoconfiguração, Auto-otimização, Autocura e Autoproteção). Contudo, o emprego e integração de todos os conceitos não é uma tarefa trivial, sendo uma tarefa dispendiosa e inviável no cronograma deste projeto de doutorado. Portanto, alguns desses conceitos (Autoconfiguração e Auto-otimização) foram elencados e fazem parte dos mecanismos a serem desenvolvidos com o intuito de garantir um gerenciamento relevante aos profissionais da área de TI. Apesar deste projeto não preencher as lacunas na parte de segurança (Autoproteção) e tolerância a falhas (Autocura), em um projeto maior, que envolva outros trabalhos acadêmicos (iniciações científicas, mestrados e doutorados), esses tópicos poderão ser cobertos. Dessa forma, será possível a criação de mecanismos que, trabalhando em conjunto, consigam tornar uma arquitetura autogerenciável.

No próximo capítulo são abordados os conceitos teóricos e estruturais da Computação em Nuvem, que também compõem o embasamento teórico para a criação de mecanismos autoconfiguráveis e auto-otimizáveis para arquiteturas virtualizadas que visam a provisão de QoS.

COMPUTAÇÃO EM NUVEM

4.1 Considerações Iniciais

Neste capítulo o tema "Computação em Nuvem" é abordado em diversos aspectos que abrangem desde uma discussão sobre a sua definição até modelos e as características desse modelo de computação. Um enfoque maior é dado aos tópicos relacionados ao presente projeto com o objetivo de conduzir a um melhor entendimento do trabalho.

4.2 Embasamento Teórico

O surgimento do termo "Computação em Nuvem" ("*Cloud Computing*") é incerto. Anteriormente ao surgimento deste termo, a Internet e toda a sua complexidade já era representada pelos administradores de redes e sistemas computacionais como sendo uma "Nuvem" ("*Cloud*"), pois as informações poderiam tomar diversos caminhos distintos, por exemplo, podendo trafegar por diversos roteadores globalmente distribuídos.

Acredita-se que o termo completo foi introduzido publicamente pela primeira vez por Eric Schmidt, quando ainda era *Chief Executive Officer* (CEO) da Google, em uma palestra na *Search Engine Strategies Conferences* em 2006 (QIAN *et al.*, 2009). O termo pode ter sido influenciado por outro modelo computacional, já existente na época, a computação em grade ("*Grid Computing*"). Ambos os termos já foram discutidos por diversos pesquisadores (EVOY; SCHULZE, 2008) (VOAS; ZHANG, 2009) (FOSTER *et al.*, 2008).

Alguns autores despertaram um senso crítico para o termo, que rapidamente se disseminou, instigando a reflexão sobre o surgimento de algo extraordinariamente novo na área de computação ou algo que já existia e recebeu um novo rótulo. Esse é o caso do artigo de Voas e Zhang, 2009, cujo título traduzido para o português faz a seguinte pergunta: "Computação em Nuvem: Um novo vinho ou apenas uma nova garrafa?" (VOAS; ZHANG, 2009). Nesse

Figura 6 – Mudança de paradigma da computação. Extraído de (VOAS; ZHANG, 2009).

artigo, os autores traçam uma perspectiva histórica encaixando seis fases (Figura 6) de algumas das evoluções da área da computação dentro do contexto de computação em nuvem buscando identificar se realmente há algo de novo ou tratava-se apenas de um próximo modismo.

Na primeira fase, os autores relembram a época dos *mainframes* da década de 60, quando os recursos dessas "poderosas" máquinas eram acessados por meio de terminais simples compostos basicamente de teclado e monitor. A segunda fase relata o surgimento do *Personal Computer* (PC) com capacidade computacional suficiente para satisfazer os usuários nos trabalhos cotidianos, descartando o uso dos *mainframes*. Na terceira fase são citadas as redes de computadores que surgiram e permitiram que múltiplos computadores se conectassem e compartilhassem recursos. A quarta fase destaca a expansão das redes de computadores de em escala global com o advento da Internet e a possibilidade dos usuários acessarem aplicações e recursos remotos. Na quinta fase os autores introduzem a computação em grade que permite de forma facilitada o compartilhamento de capacidade computacional e armazenamento de recursos de forma transparente para os usuários. Na sexta e última fase, surge a computação em nuvem que explora todos os recursos disponíveis na Internet de um modo simples e escalável.

A comparação da Computação em Nuvem com *mainframes* e terminais "burros" é claramente uma visão simplista; os autores desse artigo expõem essa diferença explicando que enquanto a capacidade de um *mainframe* é finita, a computação em nuvem representa todos os recursos possíveis na Internet, o que sugere uma capacidade infinita (VOAS; ZHANG, 2009). Contudo, os autores não chegam a uma conclusão no artigo, recomendando a leitura de outros trabalhos e estimulando o senso crítico sobre o termo que surgia e incentivando a contribuição de novas sugestões, ideias e trabalhos sobre o assunto.

Na área de computação distribuída, a computação em grade pode ser considerada como a fase antecessora à da computação em nuvem, existindo algumas semelhanças entre ambos

os modelos computacionais. Foster et. al, 2008 comparam e contrastam esses dois modelos computacionais concluindo que ambos possuem pontos em comum no que diz respeito às suas visões, arquiteturas e tecnologias, mas diferem em outros aspectos como modelo de programação, modelo de negócios, modelo de dados, aplicações, etc. (FOSTER *et al.*, 2008). Ainda nesse mesmo artigo, os autores definem a Computação em Nuvem como sendo: “*Um paradigma de computação distribuída em larga escala que é conduzida por economias de escala, no qual um conjunto abstrato, virtualizado, dinamicamente escalável e gerenciável de processamento, armazenamento, plataformas e serviços são entregues sob demanda para clientes externos através da Internet* (FOSTER *et al.*, 2008)”.

Resumidamente, podem-se citar algumas das principais semelhanças e diferenças entre os modelos. Por exemplo, ambos os modelos estão calcados no conceito de Computação Utilitária (*Utility Computing*), que conceitualmente ganhou popularidade como um modo eficiente e escalável de oferecer recursos de computação (QIAN *et al.*, 2007). Dessa forma, recursos computacionais podem ser fornecidos dinamicamente e sob demanda. Para um melhor entendimento, pode-se fazer uma analogia ao sistema de fornecimento de energia elétrica ou de abastecimento de água, no qual se tem a impressão de serem recursos ilimitados e sempre disponíveis. Neste ponto pode-se destacar uma diferença entre os dois tipos, na computação em nuvem há um forte apelo econômico na prestação deste tipo de serviço, enquanto na computação em grade esse fato nem sempre é uma realidade, especialmente em grades com objetivos científicos, onde recursos (geralmente ociosos) são cedidos de forma voluntária, caracterizando o que alguns pesquisadores denominam como "Computação Voluntária" (*"Volunteer Computing"*) (CUNSOLO *et al.*, 2009).

Em ambos os modelos existe a preocupação com o desperdício de recursos ociosos que podem ser utilizados por outros clientes da grade ou nuvem. A diferença está na visão que cada modelo faz sobre o recurso ocioso, enquanto para a computação em grade um computador ocioso representa um possível candidato que irá ceder os seus recursos, para a computação em nuvem esse computador pode ser visto de duas formas, como um recurso em uso caso alguém esteja pagando por ele, ou caso contrário como um desperdício de recurso. Ou seja, para a grade recursos ociosos são sempre bem vindos desde que haja a intenção e oportunidade de utilizá-los, enquanto para a nuvem recursos ociosos podem ser tanto necessários quanto inconvenientes. Independente do modelo, quando há o interesse de engenheiros e desenvolvedores em tentar otimizar a eficiência energética dos diversos processos de computação, sejam eles através de intervenção no *hardware* ou na arquitetura de *software*, surge o conceito de "Computação Verde"(do inglês: *"Green Computing"*) (KATSAROS *et al.*, 2012).

Em resumo, na computação em grade os recursos são geralmente fornecidos de forma colaborativa ou voluntária, explorados em sua maioria pela comunidade científica como, por exemplo, para pesquisas em computação de alto desempenho (*High Performance Computing* (HPC)). Embora, haja propostas de utilização da computação em nuvem como sendo computação voluntária (CUNSOLO *et al.*, 2009) ou relacionadas com HPC (VECCHIOLA; PANDEY;

BUYA, 2009) (EGWUTUOHA *et al.*, 2012) (GUPTA *et al.*, 2013), na computação em nuvem os recursos podem ser tanto cedidos gratuitamente quanto taxados pelos provedores da nuvem. Empresas comerciais como Amazon, HP, IBM, Microsoft e Google investem em constantes pesquisas e fazem diversos investimentos nesse tipo de computação, especialmente em *marketing* para atrair cada vez mais clientes.

Analisando de forma crítica, esse modelo de computação pode ser uma estratégia para essas grandes empresas (re)aproveitarem os seus grandes potenciais de computação que estão concentrados em seus *data centers*, tirando vantagens dos momentos de ociosidade de seus equipamentos ou se beneficiando de toda infraestrutura e profissionais já alocados. Por outro lado, também há outras vantagens para os clientes desses provedores, que podem ser outras empresas, cientistas, desenvolvedores e usuários finais. Essas vantagens, modelos de implementação, tipos de serviços prestados e algumas das principais características da computação em nuvem serão abordados nas próximas seções.

Antes de finalizar o embasamento teórico é interessante analisar a conceituação e visão de algumas empresas, acadêmicos e instituições sobre a computação em nuvem. Para a IBM, as características essenciais da computação em nuvem¹¹ são os autoserviços (*self-service*) sob demanda (*on demand*), o amplo acesso pela rede, o agrupamento e compartilhamento (*pooling*) de recurso, a rápida elasticidade e o serviço mensurado ou serviço medido (*measured service*) (BEHRENDT *et al.*, 2011).

Ainda na visão da IBM essas características podem ser encontradas em requisições e soluções com *Service Oriented Architecture* (SOA) em várias empresas, porém, embora tais características sejam opcionais para SOA, para a computação em nuvem são obrigatórias. De fato, um dos principais alicerces da computação em nuvem está na prestação de serviços sob demanda e a arquitetura orientada a serviços torna-se uma aliada nesta tarefa.

Contudo, a IBM chama a atenção sugerindo que uma única solução em SOA não reúne todas essas características ao mesmo tempo, exceto no caso de uma organização muito madura e alavancada em SOA. Outro aspecto destacado é que as soluções SOA geralmente não são compartilhadas entre organizações, por exemplo, a reutilização é empregada geralmente dentro de uma mesma organização ou indústria, e não entre organizações e comunidades inteiras como pode ser o caso da computação em nuvem (BEHRENDT *et al.*, 2011). Esse exemplo remete para um termo denominado "*multitenancy*" ("múltiplos inquilinos") que permite o aproveitamento de uma mesma arquitetura e/ou sistema por vários clientes independentemente de serem da mesma organização. A empresa Salesforce.com foi uma das responsáveis por tornar esse termo popular. Porém, lembrando a primeira fase citada por Voas e Zhang (2009), os antigos *mainframes* proviam uma tarefa semelhante; há um longo tempo a IBM presta serviço de computação (processamento, armazenamento, etc.) para outras empresas, organizações e governos.

¹¹ Essas cinco características essenciais são as mesmas definidas pelo *National Institute of Standards and Technology* (NIST) e serão detalhas ainda neste capítulo.

Entretanto há grandes diferenças entre a era dos *mainframes* com o modelo proposto pela computação em nuvem. Por exemplo, atualmente praticamente não existem mais terminais "burros", embora haja uma tendência dos usuários finais utilizarem cada vez mais computadores de recursos limitados, como *smartphones*, *tablets* e *netbooks* (denominados "*thin clients*"). Outra diferença está na comunicação entre os clientes e provedores que é fortemente associada à Internet. Finalmente, pesquisas recentes sobre um provável aquecimento global são utilizados como argumentos para esse tipo de organização de recursos buscando uma forma otimizada de utilização, possibilitando a economia de energia e consequentemente a diminuição de emissão de CO₂ na atmosfera.

A Oracle está em consenso com a IBM quanto às características essenciais da computação em nuvem. Para a Oracle a computação em nuvem emerge como uma das mais importantes estratégias de computação para o mundo empresarial. A combinação de novas tecnologias e processos proporcionou uma revolução na maneira que a computação é desenvolvida e entregue aos usuários finais, prometendo uma capacidade de entregar aplicativos a um custo menor (ANBARASU *et al.*, 2011). IBM, Oracle, HP, entre outras grandes empresas estão ativamente envolvidas no desenvolvimento e na criação de normas fundamentais para o setor em torno da computação em nuvem. Essas empresas e muitas outras participam do grupo de trabalho aberto em computação em nuvem (*The Open Group Cloud Computing WorkGroup*¹²) que busca criar um entendimento comum e formalizar padrões para aspectos relacionados a esse modelo computacional.

Resumidamente, o conceito de computação em nuvem para Armbrust *et. al.* (2010) envolve tanto as aplicações fornecidas como serviços por meio da Internet quanto ao *hardware* e ao *software* presentes em grandes centros de dados que fornecem esses serviços. Para Buyya *et. al.* (2009) a computação em nuvem pode ser considerada uma extensão de outros paradigmas, como os da computação em grade e utilitária, acrescentando a capacidade de que as aplicações de negócios podem ser expostas como serviços sofisticados acessados por meio de uma rede (Internet) (BUYYA *et al.*, 2009).

O Instituto Nacional de Padrões e Tecnologia dos Estados Unidos (NIST - *National Institute of Standards and Technology*) define a computação em nuvem da seguinte forma (MELL; GRANCE, 2012):

“A computação em nuvem é um modelo que permite o acesso à rede de forma ubíqua, conveniente e sob demanda, a um conjunto compartilhado de recursos computacionais configuráveis (tais como, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou interação com o provedor de serviços. Este modelo de nuvem é composto de cinco características essenciais, três modelos de serviço e quatro modelos de implementação (MELL; GRANCE, 2012).”

¹² <http://www.opengroup.org/getinvolved/workgroups/cloudcomputing>

Diversos pesquisadores citam essa definição de computação em nuvem em seus artigos (ZHANG; CHENG; BOUTABA, 2010), (JR., 2011), (RIMAL; CHOI, 2012), (XU, 2012), (DUKARIC; JURIC, 2013), etc. Além desses pesquisadores, diversas empresas, incluindo IBM e Oracle, compactuam com essa definição incluindo inclusive às mesmas cinco características essenciais que já foram citadas anteriormente. Detalhes dos modelos de implementação, tipos (ou modelos) de serviços e características da computação em nuvem são discutidos a seguir.

4.3 Modelos de Computação em Nuvem

De acordo com Patidar et. al. (2012) geralmente as implementações de computação em nuvem são classificadas de acordo com o proprietário do centro de dados (*data center*) que hospeda a nuvem. Essa afirmação provavelmente foi inferida das definições do NIST sobre os quatros modelos de implementação para a computação em nuvem que são (MELL; GRANCE, 2012):

- **Nuvem Privada (*Private Cloud*):** Nesta implementação, a infraestrutura é fornecida para uso exclusivo de uma única organização composta de vários clientes (por exemplo, unidades de negócios). Além disso, essa infraestrutura pode ser mantida, gerenciada e operada pela própria organização, por um terceiro, ou por alguma combinação desses dois últimos. Acrescenta-se ainda que ela pode estar localizada dentro ou fora das instalações da organização. Rimal et al., 2009, acrescentam que os dados e processos são gerenciados dentro da organização sem restrições de largura de banda, exposições de segurança e requisitos legais (RIMAL; CHOI; LUMB, 2009).
- **Nuvem Comunitária (*Community Cloud*):** A infraestrutura de uma nuvem comunitária é fornecida para uso exclusivo de uma comunidade específica de clientes que fazem parte de organizações que possuem interesses compartilhados (por exemplo, missões, requisitos de segurança, políticas, etc.). Ela pode ser mantida, gerenciada e operada por uma ou mais organizações da comunidade, por um terceiro, ou alguma combinação entre eles. Assim como a nuvem privada, a infraestrutura pode estar localizada dentro ou fora das instalações das organizações.
- **Nuvem Pública (*Public Cloud*):** Neste modelo de nuvem, a infraestrutura é fornecida para uso aberto do público em geral. Ela pode ser mantida, gerenciada e operada por uma empresa, instituição acadêmica, organização governamental ou uma combinação entre eles. Na nuvem pública a infraestrutura está localizada nas instalações do provedor da nuvem. Patidar et. al. (2012) classificam a nuvem pública como sendo um centro de dados composto de *hardwares* e *softwares* que são executados em empresas terceiras como, por exemplo, Google e Amazon, as quais expõem seus serviços para outras empresas e clientes por meio da Internet. Além disso, esses autores acrescentam que a computação

Figura 7 – Modelos de Implementações de Nuvens baseada nas definições do NIST.

em nuvem pode atender dois tipos de clientes, os usuários finais e as companhias. E assim, englobando ambos os mercados B2C (*Business-to-Consumer*) e B2B (*Business-to-Business*) respectivamente.

- **Nuvem Híbrida (*Hybrid Cloud*):** Na nuvem híbrida, a infraestrutura da nuvem é uma composição de duas ou mais infraestruturas de nuvem distintas (privada, comunitária ou pública) que permanecem sendo entidades únicas, mas são unidas por tecnologias padronizadas ou proprietárias que permitem a portabilidade de dados e aplicações (por exemplo, *cloud bursting*¹³ para o balanceamento de carga entre nuvens).

A Figura 7 exibe as quatro implementações de nuvens propostas nas definições do NIST.

4.4 Modelos de Serviços na Computação em Nuvem

Os modelos de serviços na computação em nuvem propõem as capacidades ou as formas de como os serviços na nuvem são prestados aos seus clientes. Essa abordagem pode caracterizar uma divisão em camadas abstrata dos serviços prestados (BUYA; BROBERG; GOSCINSKI, 2011).

¹³ O termo "*bursting*" ou "*burst*" que significa "explosão" em português tem sido utilizado por empresas de computação como uma forma de expressar um ganho de capacidade. Esse ganho de desempenho pode ser temporário ou com um propósito específico e geralmente é fornecido sob demanda. Os processadores Intel possuem essa propriedade que é comercialmente denominada *Intel® Burst Technology*.

Os três modelos de serviços para a computação em nuvem definidos pelo NIST são os seguintes (MELL; GRANCE, 2012):

- **Software como um Serviço (*Software as a Service (SaaS)*):** Trata-se da capacidade do provedor em fornecer, ao cliente, aplicativos em execução em uma infraestrutura em nuvem¹⁴. As aplicações são fornecidas como um serviço e são acessíveis a partir de diversos dispositivos clientes, por meio de interfaces *thin client*, como um navegador web (por exemplo, aplicação cliente de e-mail baseado na Web), ou uma interface de programa (API). O cliente não gerencia ou controla a infraestrutura básica da nuvem, ou seja, não gerencia a rede, os servidores, os sistemas operacionais, o armazenamento, ou mesmo capacidades individuais de aplicativos, com a possível exceção de configurações limitadas de aplicativos específicos do cliente. Em outras palavras essa definição pode ser complementada ao afirmar que SaaS estabelece que as aplicações são executadas nas nuvens eliminando a necessidade de instalação e execução nos computadores dos clientes, exemplos de aplicações incluem, em nível empresarial, a *Salesforce*, *Google Apps* para aplicações pessoais, *Gmail*, *Facebook* e *Twitter* (MARSTON *et al.*, 2011).
- **Plataforma como um Serviço (*Platform as a Service (PaaS)*):** Corresponde a capacidade do provedor em fornecer ao cliente a possibilidade de implantar (*deploy*) aplicações na infraestrutura da nuvem que são criadas usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. Assim como no modelo SaaS, o consumidor não gerencia ou controla a infraestrutura de nuvem (rede, servidores, sistemas operacionais ou armazenamento), mas tem controle sobre os aplicativos implementados e eventuais permissões para ajustes de configurações no ambiente do aplicativo hospedado. Na visão da Intel, a PaaS fornece uma plataforma para os desenvolvedores que facilita a implementação e implantação de aplicativos (*Web Applications* ou SaaS) evitando o custo e a complexidade da compra e gerenciamento de *hardware* e camadas de *softwares* (INTEL-CORPORATION, 2010). Pode-se citar dois exemplos mais conhecidos de PaaS como sendo o *Google App Engine* (GAE) (CIURANA, 2009) e a *Windows Azure* da Microsoft (MICROSOFT, 2012b).
- **Infraestrutura como um Serviço (*Infrastructure as a Service (IaaS)*):** É fornecida ao cliente a capacidade de provisionamento de processamento, armazenamento, redes e outros recursos computacionais fundamentais. O cliente é capaz de implantar e executar *softwares* arbitrários, os quais podem incluir sistemas operacionais e aplicativos. Novamente, o cliente não gerencia ou controla a infraestrutura de nuvem, mas tem controle sobre sistemas operacionais, armazenamento, aplicativos implantados e um eventual e limitado controle dos componentes de rede selecionados (por exemplo, *firewalls* do *host*). Provedores de

¹⁴ Uma infraestrutura em nuvem é composta por uma coleção de *hardwares* e *softwares* que possibilitam as cinco características essenciais da computação em nuvem (MELL; GRANCE, 2012)

Figura 8 – Modelos de serviços da Computação em Nuvem - Adaptação de (BUYYA; BROBERG; GOSCINSKI, 2011) e (INTEL-CORPORATION, 2010).

IaaS normalmente oferecem uma infraestrutura virtualizada como o serviço ao invés do *hardware* real diretamente. Tais serviços podem ser fornecidos por interfaces padronizadas para serviços do tipo PaaS e SaaS, alguns exemplos de serviços IaaS são a *Amazon Web Services* (AWS) com os seus serviços de processamento *Elastic Compute Cloud* (EC2) e armazenamento *Simple Storage Service* (S3) (PATIDAR; RANE; JAIN, 2012).

A Figura 8 exibe os três modelos de serviços propostos nas definições do NIST; nela podem-se observar as principais formas de acesso ou ferramentas de gerenciamento e os respectivos conteúdos de cada modelo (BUYYA; BROBERG; GOSCINSKI, 2011). Ainda é possível identificar as prováveis classes de usuários (INTEL-CORPORATION, 2010) e alguns exemplos de serviços comerciais que pertencem a cada camada da pilha de modelos de serviços da computação em nuvem.

Esses três modelos de serviços podem ser considerados como os principais modelos da computação em nuvem, pois são lembrados por diversos autores, instituições e empresas em suas definições de modelos de serviços. Contudo, não é incomum encontrar diversos outros acrônimos para outros tipos (ou modelos) de serviços prestados na nuvem. Por exemplo, Zhou et. al., 2012, além dos três modelos do NIST, acrescentam mais três categorias de serviços: “Rede como um Serviço” (*Network as a Service* (NaaS)), “Gerenciamento de Identidade e Política como um Serviço” (*Identity and Policy Management as a Service* (IPMaaS)) e “Dados como um Serviço” (*Data as a Service* (DaaS)). Além disso, os autores classificam essas categorias em três níveis (ZHOU et al., 2010):

- **Nível de Hardware:** Engloba os serviços das categorias IaaS e NaaS que provêm armazenamento, ciclos de CPU e rede para os clientes.

- **Nível de Sistema:** Apenas os serviços da categoria PaaS pertencem a esse nível que fornece uma plataforma para execução de aplicações dos clientes.
- **Nível de Aplicação:** Esse nível é composto pelas categorias IPaaS, DaaS e SaaS que provêm aplicações específicas para clientes dedicados.

Além desses modelos ou categorias (tipos) de serviços, podem-se encontrar diversos outros como, por exemplo, *Hardware as a Service* (HaaS), *[Development, Database, Desktop] as a Service* (DaaS), *Business as a Service* (BaaS), *Framework as a Service* (FaaS), *Organization as a Service* (OaaS), etc. (RIMAL; CHOI; LUMB, 2009). Em Patidar et. al. (2012) é citada uma observação que foi feita por Larry Ellison (CEO da Oracle) em 2008, que chamava a atenção pelo fato do termo (Computação em Nuvem) ser utilizado em demasia e sendo aplicado a tudo no mundo da computação. Embora possa existir realmente um exagero devido a toda uma jogada de *marketing* ao redor do termo, qualquer recurso ou aplicação que possa ser disponibilizado sob demanda pela rede (Internet) pode fazer parte do escopo da computação em nuvem, o que sugere ao acrônimo XaaS (*Everything as a Service* (XaaS)).

4.5 Características da Computação em Nuvem

No decorrer deste capítulo diversas características da computação em nuvem foram destacadas. Nesta seção algumas dessas características serão mais bem detalhadas possibilitando um entendimento mais profundo sobre o assunto e permitindo uma reflexão sobre os benefícios e desafios deste tema.

4.5.1 Características Essenciais

O NIST adota cinco características que considera como sendo essenciais para a computação em nuvem, essas características também são aceitas por grandes empresas de TI (IBM, Oracle, etc.) e citadas por diversos pesquisadores ((DILLON; WU; CHANG, 2010), (TAKABI; JOSHI; AHN, 2010), (RIMAL; CHOI, 2012), (BOSSCHE; VANMECHELEN; BROECKHOVE, 2013), etc.). Essas cinco características são detalhadas da seguinte forma (MELL; GRANCE, 2012):

- **Autoatendimento sob demanda (*On-demand self-service*):** Um cliente pode adquirir unilateralmente recursos computacionais, tais como tempo de processamento no servidor e armazenamento na rede, conforme seja necessário e de forma automaticamente, sem a necessidade de interação humana com cada prestador de serviço. Essa característica está relacionada com outra denominada elasticidade rápida.
- **Ampla acesso à rede (*Broad network access*):** Essa característica permite que os recursos da nuvem estejam disponíveis por meio de uma rede de banda larga e sejam acessados

utilizando mecanismos padronizados que promovem o uso de plataformas clientes, heterogêneas e leves (*thin*) ou pesadas (*thick*) (por exemplo, celulares, *tablets*, *notebooks* e estações de trabalho). Realmente, a banda larga é uma característica essencial, pois os serviços são fornecidos por meio de uma rede (por exemplo, a Internet) que pode sofrer influências de um maior tráfego de dados conforme esses serviços são utilizados sob demanda.

- **Agrupamento e compartilhamento de recursos (*Resource pooling*):** Os recursos computacionais do provedor de serviços da nuvem são agrupados com o objetivo de atender vários clientes por meio de um modelo de múltiplos inquilinos (*multi-tenant*). Os diferentes recursos físicos e virtuais são atribuídos dinamicamente de acordo com a demanda do cliente. O cliente geralmente não tem o controle ou o conhecimento sobre a exata localização dos recursos que são disponibilizados, mas pode ser capaz de especificar um determinado local em um nível maior de abstração (por exemplo, o cliente pode definir em qual país, estado ou *data center* estarão os recursos). Exemplos de recursos incluem o armazenamento, processamento, memória e largura de banda de rede.
- **Elasticidade rápida (*Rapid elasticity*):** Essa característica garante que os recursos possam ser provisionados e liberados de uma forma elástica. Em alguns casos isso é feito de forma automática, permitindo escalar rapidamente, aumentando ou diminuindo a quantidade de recursos de forma compatível com a demanda. O cliente tem a impressão de que os recursos disponíveis para provisionamento parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento. Em uma próxima subseção essa característica será abordada com maiores detalhes.
- **Serviço Mensurado (*Measured service*):** Os sistemas em nuvem podem automaticamente controlar e otimizar o uso dos recursos. Isso pode ser feito por meio de recursos de medições presentes em algum nível de abstração apropriado para o tipo de serviço (por exemplo, armazenamento, processamento, largura de banda e contas de usuários ativos). O uso desses recursos pode ser monitorado, controlado e reportado, oferecendo transparência tanto para o provedor quanto para o cliente do serviço utilizado. Essa característica remete a modelos de pagamento pelo uso (*pay-for-use* ou *pay-as-you-go*) baseados na computação utilitária.

Essas características em conjunto permitem que os recursos da nuvem possam ajudar as empresas na rápida configuração de novas aplicações e beneficiando o surgimento de novas ideias originais (LIU *et al.*, 2012).

4.5.2 Virtualização

A virtualização é um item chave dentro da computação em nuvem, pois permite uma rápida (re)configuração de recursos computacionais facilitando e agilizando o gerenciamento de

arquiteturas completas.

A tecnologia de virtualização não é recente, há relatos que ela foi originalmente desenvolvida pela IBM na década de 60, para servir à plataforma de *mainframe* System/360 (GOLDBERG, 1974). Basicamente a técnica consiste em um monitor de máquina virtual (*Virtual Machine Monitor* (VMM)) que é executado diretamente sobre o *hardware*, como um sistema operacional, executando outras instâncias de sistemas operacionais. Cada uma dessas instâncias é executada de forma isolada e têm acesso a apenas um conjunto de recursos definidos em sua máquina virtual. O VMM é atualmente mais conhecido como “hipervisor” (*hypervisor*), devido à sua posição hierárquica e função de gerenciamento acima do sistema operacional.

Atualmente a técnica de virtualização é extrapolada em diversas frentes, por exemplo, a máquina virtual java (*Java Virtual Machine* (JVM)) da Oracle permite que as aplicações criadas na linguagem Java sejam executadas sobre essa máquina virtual que pode possuir uma versão para diversos sistemas operacionais (Windows, Linux, Mac Os, etc.) e, dessa forma, torna as aplicações criadas nessa linguagem em aplicações multiplataforma. Outro exemplo, é a virtualização de rede, uma rede virtual privada (*Virtual Private Network* (VPN)) pode ser criada para interligar duas unidades de negócio fisicamente distantes sem a necessidade de um custo despendido em um enlace físico dedicado. A criação de uma VPN permite que essas duas unidades de negócio se comuniquem por meio de uma infraestrutura já existente da Internet de uma forma segura.

Devido a uma ampla utilização da técnica de virtualização, torna-se necessário definir esse tema no escopo desse projeto de doutorado. Dessa forma, para este projeto, qualquer referência a virtualização está relacionada à técnica de virtualização de recursos computacionais que é empregada por provedores de serviços em nuvem, em especialmente aqueles que fornecem serviços no modelo IaaS. A tecnologia de virtualização é uma grande aliada desses provedores, pois possibilitam o aumento da utilização da infraestrutura por meio da multiplexação da execução de várias instâncias de máquinas virtuais em um único servidor físico. A virtualização também permite ajustar de forma flexível as características individuais (núcleos de processamento, memória, disco, rede, etc.) de uma determinada instância (BOSSCHE; VANMECHELEN; BROECKHOVE, 2013).

Segundo Antonopoulos e Gillam (2010) os métodos de virtualização podem ser classificados em duas categorias, “paravirtualização” (*paravirtualization*) e “virtualização completa” (*full virtualization*), que se diferenciam de acordo com a necessidade ou não do núcleo (*kernel*) do Sistema Operacional (SO) convidado (*guest*) ser modificado.

Na virtualização completa há todo um suporte de virtualização para que o ambiente de *hardware* por completo seja virtual. Dessa forma, o núcleo do sistema operacional convidado não precisa ser alterado, pois ele interage com os recursos virtuais de *hardware* da mesma forma como se estivesse interagindo com os reais. A virtualização completa pode ser interessante para a execução de sistemas operacionais de código não aberto, como o Windows da Microsoft, em

Figura 9 – Comparação entre a virtualização completa e a paravirtualização. Adaptado de (CHANTRY, 2009) e (ANTONOPOULOS; GILLAM, 2010).

que uma alteração no *kernel* exigiria o código fonte (ANTONOPOULOS; GILLAM, 2010).

A paravirtualização exige uma modificação no núcleo do sistema operacional convidado e em seus respectivos *drivers* de dispositivos com o objetivo de colaborar com o hipervisor. Como a paravirtualização não é preciso virtualizar todo o ambiente *hardware* e o SO convidado, tendo ciência que está sendo executado em uma máquina virtual, pode acessar alguns recursos reais diretamente e, assim, atingir um melhor desempenho que a virtualização completa (ANTONOPOULOS; GILLAM, 2010). Em contrapartida, os sistemas operacionais de código fechado precisam ser fornecidos pelos seus fabricantes contemplando tais características.

A Figura 9 exibe uma comparação entre as arquiteturas da virtualização completa e paravirtualização.

A Tabela 1 exibe uma comparação entre algumas das mais conhecidas ferramentas (hipervisores) de virtualização de servidores.

Tabela 1 – Ferramentas de virtualização em servidores (ANTONOPOULOS; GILLAM, 2010).

Hipervisor	Tipo de Virtualização	Destaque	Desempenho	Licença
KVM (KVM, 2013)	Virtualização Completa	Estabelece uma MV como sendo um processo regular do Linux	Próximo ao desempenho nativo do hardware	Código Aberto
Xen (XEN, 2013)	Paravirtualização	Suporta migração de MV em tempo real	Desempenho compatível ao nativo do hardware	Código Aberto
VMware (VMWARE, 2013)	Virtualização Completa	Fornecer uma família de produtos consolidados para o gerenciamento de infraestrutura virtual	Próximo ao desempenho nativo do hardware	Comercial
Microsoft Hyper-V (MICROSOFT, 2012a)	Virtualização Completa	Capaz de interceptar chamadas ao sistema feitas pelos SOs convidados	Próximo ao desempenho nativo do hardware	Comercial

A virtualização é base para a rápida elasticidade e consequentemente para a prestação de serviços sob demanda e serviços mensurados.

4.5.3 Computação Elástica

As tecnologias avançadas empregadas na virtualização permitiram que recursos computacionais fossem fornecidos, modificados e gerenciados de uma forma mais facilitada e ágil. Os recursos computacionais podem ser rapidamente alocados e liberados conforme as necessidades dos clientes da nuvem.

A escalabilidade de sistemas distribuídos é alcançada quando há um aumento significativo no número de recursos computacionais e no número de usuários e o sistema permanece eficiente (COULOURIS; DOLLIMORE; T., 2007). Dentro desse conceito, a elasticidade da computação em nuvem é escalável, porém há outros aspectos interessantes que envolvem a elasticidade. Além de escalar permitindo o aumentando da quantidade de recursos, ela também permite a diminuição de recursos e geralmente isso é feito de forma automática (HAN *et al.*, 2012) (ISLAM *et al.*, 2012b).

Segundo Caron *et al.* (2012) diversas soluções para o autodimensionamento (*auto-scaling*) têm sido propostas pela academia e fornecedores de serviços em nuvem. Todas essas soluções permitem que os usuários definam um conjunto de regras de dimensionamento relacionadas aos serviços hospedados nas nuvens. Cada uma dessas regras é composta por uma ou mais condições e por um conjunto de ações que devem ser tomadas quando essas tais condições são alcançadas. Essas condições são geralmente definidas utilizando-se um conjunto de métricas como, por exemplo, utilização de CPU, e limiares, quando um limiar é ultrapassado então a condição é alcançada (CARON *et al.*, 2012). Tais limiares e métricas geralmente estão especificados em acordos de nível de serviço (*Service Level Agreement* (SLA)) que devem ser cumpridos a fim de garantir uma QoS adequada.

Dessa forma, percebe-se que o autodimensionamento depende de um mecanismo ou sistema de monitoramento. O monitoramento pode ser feito em dois diferentes níveis, sendo cada nível destinado ao beneficiário das informações de monitoramento (CARON *et al.*, 2012):

- **Baixo nível, monitoramento de plataforma:** Este nível de monitoramento é interessante para o provedor da plataforma. Sua finalidade é obter informações que refletem a situação da infraestrutura física da plataforma em nuvem como um todo. Em provedores comerciais esse nível de monitoramento é usualmente mantido como confidencial.
- **Alto nível, monitoramento de informação:** Este nível de monitoramento tipicamente interessa mais ao cliente da nuvem. As informações desse monitoramento são focadas na situação da plataforma virtualizada de cada cliente.

O autodimensionamento tem uma relação importante com o balanceamento de carga (*load balancing*) na computação elástica. Ambos devem trabalhar em conjunto com o objetivo de obter um uso mais eficiente da plataforma e poupar gastos. A forma mais comum de dimensionamento é a horizontal, que implica na alocação e liberação de máquinas virtuais, a outra forma

é o dimensionamento vertical que aumenta ou diminui os recursos computacionais de uma ou mais instâncias de máquinas virtuais. Essa segunda opção de dimensionamento geralmente não é considerada devido a impossibilidades de mudanças em tempo real dos recursos computacionais virtuais (CPU, memória, etc.) em sistemas operacionais de propósito geral (CARON *et al.*, 2012). Além disso, algumas aplicações podem exigir alterações em suas configurações para aproveitar o aumento de recursos de forma eficaz, por exemplo, aumentar o número de *threads* de um servidor Web Apache ou configurar a quantidade de memória para uma aplicação em Java (por exemplo, configurando o *stack size*).

Portanto, a elasticidade rápida é uma das principais características da computação em nuvem que envolve diversas outras características e se relaciona com os princípios e aspectos computacionais como, por exemplo, virtualização, autodimensionamento, monitoramento, balanceamento de carga, SLAs, QoS, etc. O gerenciamento e execução da elasticidade de recursos computacionais podem ser associados à soluções baseadas nos conceitos da computação autônoma em busca de uma elasticidade automática e mais eficiente.

4.5.4 Computação Autônoma

Buyya *et al.* (2011) consideram que o mundo de TI está passando por uma fase de transição na qual a capacidade computacional está deixando de ser gerada internamente nas empresas e instituições e passando a ser fornecida como recursos oriundos de uma computação utilitária que são entregues por meio da Internet. Isso foi possível devido ao avanço de várias tecnologias, especialmente em *hardware* (virtualização, *chips multi-core*), as tecnologias da Internet (*Web Services*, arquiteturas orientadas a serviços, Web 2.0), de computação distribuída (*clusters*, *grids*) e gerenciamento de sistemas (computação autônoma, automação de *data centers*). A convergência dos vários avanços nessas áreas conduziu ao advento da Computação em Nuvem (Figura 10) (BUYYA; BROBERG; GOSCINSKI, 2011).

Dentro desse contexto, a utilização de recursos ou princípios da computação autônoma são utilizados com a intenção de reduzir a intervenção humana (ANGHEL *et al.*, 2010) (SOLOMON *et al.*, 2010) (ETCHEVERS *et al.*, 2011) (GHANBARI *et al.*, 2011) (PANDEY *et al.*, 2012). A busca pela prestação de serviços sob demanda, de forma elástica, eleva o nível de complexidade das aplicações, tanto na parte de configuração, execução e gerenciamento. Acrescenta-se ainda critérios relacionados a QoS e acordos SLA que, dentro de um modelo *pay-as-you-go*, podem estar ancorados às regras de negócios visando principalmente o retorno do investimento (*Return on Investment* (ROI)).

4.5.5 Mercado e QoS

É sabido que a computação é dita como sendo uma "atividade-meio" para diversas outras áreas ou ciências, pois fornece meios de agilizar e organizar processos de forma eficiente. Porém,

Figura 10 – A convergência de vários avanços em áreas que conduziram ao advento da Computação em Nuvem. Traduzido de (BUYYA; BROBERG; GOSCINSKI, 2011).

muitas empresas vêm seus departamentos de TI como um “mal necessário”, pois consomem altas divisas financeiras e nem sempre têm influência no produto final, porém sem esse departamento e seus profissionais todo o processo produção pode ser ineficiente e exigências externas podem não ser cumpridas (por exemplo, descontrole no estoque e/ou falhas no processo de qualidade do produto).

A evolução da computação acontece em um ritmo acelerado, muitas vezes de forma mais rápida do que os negócios das empresas. Alguns responsáveis pelos departamentos de TI enfrentam dificuldades em aprovar novos orçamentos para a compra de novos equipamentos junto os departamentos comerciais ou financeiros. Enquanto os profissionais de TI estão preocupados em manter o sistema da empresa sempre em funcionamento e com o seu desempenho aceitável em qualquer situação, os outros estão preocupados no custo/benefício e se haverá um retorno desse investimento a curto ou longo prazo. A rápida evolução dos equipamentos é praticamente linear à desvalorização dos mesmos. Além disso, o avanço nas tecnologias de desenvolvimento de *softwares* e um provável aumento no número de usuários do sistema podem acarretar em uma futura necessidade de novos equipamentos. Por esse motivo, muitos preferem comprar equipamentos com recursos aquém dos necessários, tentando prever uma possível necessidade

no futuro.

Novos equipamentos ou recursos exigem novas configurações ou alterações em configurações existentes (por exemplo, instalação de sistema operacional, configuração de rede, inclusão dos novos recursos nas regras do *firewall*, etc.). Eles também exigem espaço, pontos de energia disponíveis e geralmente resfriamento adequado. Por fim, os equipamentos são compostos de componentes eletrônicos, já os profissionais de TI são seres humanos, embora diferentes, ambos estão sujeitos a falhas. Todas essas necessidades e problemas são inerentes às empresas que dependem de alguma forma da área de TI em seus negócios.

Diversas grandes empresas fazem fortunas propondo soluções para esses problemas, geralmente fornecendo consultorias e prestação de serviços. Dentro desse contexto, a computação em nuvem surge como uma solução que beneficia tanto o prestador de serviço como o cliente.

A computação em nuvem desperta a atenção do mercado com a sua capacidade de rápida elasticidade de recursos sob demanda. A Figura 11 exibe possíveis situações em que as empresas enfrentam um dilema se devem ou não adquirir novos recursos computacionais. No item (a) observa-se que a capacidade do sistema computacional da empresa em atender os seus clientes, falha em alguns momentos de pico (destacado em vermelho), nesses momentos alguns clientes podem ficar insatisfeitos e abandonarem os serviços da empresa definitivamente. No entanto, em alguns outros momentos (destacado em verde) os atuais recursos computacionais do sistema são suficientes. No item (b) a empresa decide aumentar os seus recursos computacionais adquirindo novos equipamentos, os momentos de picos não são mais um problema, porém a quantidade de capacidade computacional ociosa foi elevada. No item (c) a empresa passa por uma fase de pouca demanda, entre o sétimo e nono mês, muitos clientes deixam de utilizar os serviços dessa empresa nessa época, não há picos de utilização que superem a capacidade do sistema e o gasto com recursos ociosos é enorme. Em vista dos problemas enfrentados no período do item (c), a empresa resolveu criar uma promoção no período do item (d), novamente há momentos de picos e a atual infraestrutura computacional não é mais capaz de atendê-los.

Embora o exemplo anterior seja fictício, não é uma situação improvável de ocorrer. Empresas que estão iniciando no mercado, denominadas atualmente como "*start-ups*", podem se beneficiar de oportunidades encontradas na computação em nuvem eliminando um trabalho e custo adicional na aquisição e configuração de equipamentos para as suas aplicações (ISLAM *et al.*, 2012a).

Contudo, alguns ambientes de Computação em Nuvem estão abertos ao acesso público, portanto a quantidade de requisições pode ser imprevisível e variável, o que dificulta a criação de estimativas e a QoS (SULEIMAN *et al.*, 2012).

Empresas provedoras de serviços em nuvem geralmente adotam modelos de negócios que estabelecem contratos de níveis de serviço (SLA) com seus usuários. Nesses contratos estão definidos os requisitos (atributos) de QoS e os seus valores (níveis) máximos e/ou mínimos em

Figura 11 – Possíveis situações de provisão de recursos, capacidade de atendimento e demanda por serviços para uma empresa fictícia. Baseada em (ARMBRUST *et al.*, 2010).

que os serviços devem atuar (WU; GARG; BUYYA, 2011). Os provedores de serviços podem estar suscetíveis à penalidades ou multas caso o contrato não seja respeitado.

Os conceitos que tangem a qualidade de serviços e os acordos de níveis de serviços foram exaustivamente estudados em diversos outros trabalhos científicos nas áreas de sistemas distribuídos e redes de computadores. Basicamente, alguns (senão todos) atributos de QoS (Tempo de Resposta, Disponibilidade, Integridade, Custo, etc.) que são utilizados para aplicações Web como, por exemplo, *Web Services* (LEE; SHIN, 2008), podem ser instanciados no contexto de serviços em nuvem, principalmente para SaaS e IaaS.

Por fim, a computação em nuvem está consolidada (MARSTON *et al.*, 2011) e tomou sua posição no mercado estabelecendo um modelo econômico de fundamental importância para a indústria de computação.

4.6 Arquitetura da Computação em Nuvem

A revisão sobre o conteúdo teórico, conceitual e características da computação em nuvem, feita no decorrer deste capítulo, desperta o interesse no entendimento de como esse complexo conjunto pode ser organizado.

Diversos autores apresentam taxonomias ou referências de arquiteturas para a computação em nuvem (LIU *et al.*, 2011) (DUKARIC; JURIC, 2013) (BEHRENDT *et al.*, 2011) (ANBARASU *et al.*, 2011) (INTEL-CORPORATION, 2010) (RIMAL; CHOI; LUMB, 2009) (RIMAL; CHOI, 2012). De uma forma geral, elas apresentam várias semelhanças, conside-

Figura 12 – Modelo de referência conceitual da computação em nuvem na visão do NIST. Traduzida de (LIU *et al.*, 2011).

rando, por exemplo, os modelos de serviços (SaaS, PaaS e IaaS), módulos ou componentes de gerenciamento e muitas vezes os participantes envolvidos (provedor, cliente, etc.).

A referência de arquitetura para computação em nuvem do NIST foi concebida após uma pesquisa que analisou os existentes modelos e referências de computação em nuvem propostos por organizações, fornecedores e agência governamentais. Essa pesquisa envolveu a ampla participação da indústria, academia, organizações de desenvolvimentos de padrões e de adeptos da computação em nuvem pública e privada (LIU *et al.*, 2011). Dessa forma, é possível considerar a referência de arquitetura do NIST com um exemplo padrão de arquitetura para computação em nuvem, diferenciando-se de outras referências em termos de nomenclaturas, detalhes técnicos, empregos de tecnologias específicas, modelo de negócios, etc.

A Figura 12 exibe o modelo de referência conceitual da computação em nuvem na visão do NIST.

Nesse modelo estão presentes os cinco principais elementos (ou atores) participantes da arquitetura que são brevemente descritos a seguir (LIU *et al.*, 2011) (BOHN *et al.*, 2011):

- **Cliente da Nuvem:** É a pessoa ou a organização que mantém uma relação de negócios com provedores de nuvem utilizando os seus serviços (SaaS, PaaS ou IaaS).

- **Provedor da Nuvem:** É a pessoa, organização ou entidade que tem a responsabilidade de tornar os serviços (SaaS, PaaS e IaaS) disponíveis para das partes interessadas. O provedor de nuvem está envolvido com atividades nas áreas de implantação de serviços, orquestração de serviços, gerenciamento de serviços em nuvem, segurança e privacidade.
- **Auditor da Nuvem:** Trata-se de um terceiro que pode conduzir avaliações dos serviços em nuvem de forma independente. A auditoria pode considerar informações de indicadores de desempenho, segurança, privacidade, etc.
- **Intermediador da Nuvem:** O intermediador da nuvem é uma entidade que gerencia o acesso, desempenho, segurança e a prestação de serviços em nuvem, intermediando as relações e negociações entre os provedores e os clientes da Nuvem. Oferece também serviços de agregação (combinando e integrando múltiplos serviços em um único serviço ou em novos serviços) e arbitragem (semelhante à agregação, porém mais dinâmica uma vez que os serviços escolhidos pelo intermediador não são fixos).
- **Transportador de Nuvem:** Um participante intermediário que tem a função de fornecer conectividade e transporte dos serviços em nuvem do provedor para o cliente. O transportador de nuvem fornece acesso aos clientes por meio da rede, meios de comunicações ou outros dispositivos de acesso. O provedor da nuvem deve firmar acordos SLAs com o transportador de nuvem com o objetivo de fornecer serviços com o nível de qualidade adequado e assim cumprir possíveis acordos SLAs estabelecidos com os clientes da nuvem.

A arquitetura da computação em nuvem é vista de uma forma abstrata na Figura 12. Cada um dos módulos ou componentes pertencentes aos participantes da arquitetura pode ser detalhado. Porém, entrar nesse nível de detalhamento não é o foco deste trabalho, que está restrito mais especificamente à parte de IaaS.

4.7 Desafios da Computação em Nuvem

No decorrer deste capítulo, diversas vantagens e benefícios proporcionados pela computação em nuvem foram destacados. Embora seja considerada uma solução consolidada (MARSTON *et al.*, 2011), alguns desafios ainda merecem ser discutidos.

4.7.1 Segurança

As questões de segurança são intrínsecas aos sistemas distribuídos, os dados trafegam pelas redes de computadores e podem estar sujeitos a problemas de privacidade, confiabilidade, autenticidade e possíveis perdas. O modelo de múltiplos inquilinos (*multi-tenancy*) fortemente presente na computação em nuvem introduz novos desafios de segurança, pois agora os usuários não compartilham apenas a rede, mas também os mesmos recursos físicos (DILLON; WU;

CHANG, 2010). A garantia de um isolamento virtual confiável feita pelos hipervisores ainda é um assunto ainda questionável.

Outro problema acentuado pelo uso da computação em nuvem é o caso das empresas migrarem todos os seus processos e dados para a nuvem. A alta dependência e acoplamento forte com os serviços prestados pelo provedor podem causar diversos problemas à empresa. Mesmo que sejam empregados mecanismos de segurança como autenticação e criptografia, ataques do tipo *Distributed Denial of Service* (DDoS) podem não ser evitados e colocam os negócios da empresas em risco (RIMAL; CHOI, 2012).

4.7.2 Escalabilidade

Rimal et. al. (2012) consideram a escalabilidade na computação em nuvem um desafio, pois dimensionar corretamente a quantidade os recursos necessários não é uma tarefa trivial, realizar esse mesmo procedimento de forma otimizada é um desafio ainda maior (RIMAL; CHOI, 2012).

4.7.3 Provisionamento de serviços automatizados

Para Zhang et al. (2010) o provisionamento de serviços automatizados em computação em nuvem ainda é um tópico em aberto que pode envolver abordagens que empregam modelos de previsões de demanda que empregados para a alocação de recursos e satisfação dos requerimentos de QoS (ZHANG; CHENG; BOUTABA, 2010).

4.7.4 Modelos de Custo e Cobrança

Dillon et. al. (2010) destacam a necessidade de modelos de custo para a computação em nuvem. Os autores argumentam que há vantagens e desvantagens com relação ao custo, por exemplo, embora as empresas possam economizar com infraestrutura, dependendo do tipo de aplicações utilizada, isso pode implicar em um custo ainda maior com a comunicação de dados.

Modelos de cobrança precisam incorporar itens como, licenças de *softwares*, utilização da rede virtual, o gerenciamento da sobrecarga do hipervisor e dos nós da rede, etc. (DILLON; WU; CHANG, 2010).

Pode-se ainda destacar outro desafio que envolve a cobrança de impostos. Em alguns países, há a cobrança de impostos pela prestação e importação de serviços. Portanto, controlar, fiscalizar e cobrar os clientes pelo uso de serviços da computação em nuvem de provedores estrangeiros também é questão a ser considerada.

4.7.5 Acordos de nível de serviço (SLA)

De acordo com Rimal et. al. (2012) os provedores não estão fornecendo SLA. Os autores argumentam que um simples mecanismo de SLA não irá atender a complexa natureza das demandas dos clientes. Muitas pesquisas na área de SLA (*SLA monitoring*, *SLA-driven*, *SLA-aware*, etc.) possuem abordagens mais elaboradas que devem ser consideradas no contexto de computação em nuvem. Cada modelo de serviço (SaaS, PaaS ou IaaS) precisará definir diferentes especificações de SLA. Esses autores acrescentam ainda importantes observações a respeito do emprego de SLAs no contexto de computação em nuvens. Uma primeira observação é que os provedores de nuvem deverão contar com gerenciadores de recursos eficientes e que estejam sempre atualizados com informações precisas sobre os recursos utilizados. Isso significa que qualquer alteração no ambiente de nuvem precisa ser informada ao gerenciador de recursos que deve realizar uma avaliação e ajuste para o que SLA seja cumprido em tempo real. Dessa forma, os gerenciadores de recursos precisam empregar modelos de decisão rápida e eficiente utilizando algoritmos de otimização para isso. Além disso, o gerenciador pode rejeitar certos pedidos de recursos quando as SLAs não podem ser cumpridas. Todas essas atividades devem ser realizadas de uma forma automática devido à promessa de “autoatendimento” da computação em nuvem. Por fim, os autores acrescentam que esses avançados mecanismos de SLA precisam incorporar constantes resultados de *feedbacks* dos clientes e personalizar determinadas características em uma plataforma de avaliação de SLA.

4.7.6 Migração de Máquinas Virtuais

A virtualização traz diversos benefícios significativos para a computação em nuvem, um deles é a capacidade de migração de uma máquina virtual com o objetivo de balancear a carga de trabalho. Essa máquina pode ser migrada tanto de uma máquina física para outra na mesma rede local quanto para outra máquina física em outro *data center* (ZHANG; CHENG; BOUTABA, 2010). O maior desafio está na detecção de focos de sobrecargas. Embora, alguns dos atuais sistemas de virtualização permitem a migração em tempo real (*live migration*), a migração não é instantânea e ágil o suficiente para atender a mudanças repentinas na carga de trabalho (ZHANG; CHENG; BOUTABA, 2010).

A migração pode ser uma opção útil para muitos casos, além do balanceamento de carga que melhora o desempenho do sistema, ela pode auxiliar no cumprimento do SLA e proporcionar economias para o provedor de serviço. Por exemplo, ao invés de utilizar dois servidores físicos para poucas máquinas virtuais, elas podem ser migradas para um único servidor físico, economizando energia. Ou ainda, as máquinas virtuais podem ser migradas para *data centers* em outros países, nos quais o custo da energia é menor.

4.7.7 Questões políticas e jurídicas

Rimal et. al. (2012) levantam as questões políticas e jurídicas relacionadas à prestação de serviços de provedores de um determinado país para clientes em outros. Até mesmo dentro de um mesmo país podem existir leis locais diferentes que não tratam de um determinado assunto da mesma forma ou estão até mesmo em conflito. Alguns países impedem o tráfego de determinados tipos de dados ou informações, como tratar esses problemas é um complexo desafio a ser resolvido.

4.7.8 Transparência e Dependência dos Provedores

Segundo Rimal et. al. (2012), a maioria das arquiteturas da computação em nuvem são proprietárias e todos os provedores tentam na medida do possível tornar seus clientes, desenvolvedores e analistas de testes dependentes de suas tecnologias. Esse problema ocorre pela ausência de uma API e formato de dados padronizados. Uma possível solução para esse problema é a utilização de padrões propostos como OpenSpec (*open specification*) e *Open Cloud API*. Além disso, os autores também afirmam que os fornecedores de serviços devem contemplar a segurança tornando seus serviços mais transparentes para os usuários.

Outro problema de padronização pode estar relacionado às imagens¹⁵ de máquinas virtuais que por padrão são incompatíveis entre os hipervisores, ou ainda há um problema de incompatibilidade de contextualização, no qual a imagem de um determinado provedor podem não funcionar em outro provedor (KEAHEY et al., 2009) (DUKARIC; JURIC, 2013). Com relação à transparência, provedores de serviços devem fornecer contratos e informações da forma mais clara possível aos seus clientes.

4.7.9 Computação Verde

Poupar custos, recursos e economizar energia preservando o ambiente por meio da diminuição da emissão de CO₂ na atmosfera são fundamentos da computação verde (RIMAL; CHOI, 2012). A ideia de lucrar economizando energia e ainda favorecer o meio ambiente (*marketing*) é um assunto que chama muito a atenção tanto da indústria quanto da academia. Porém, essa não é uma tarefa simples, pois está relacionada com outros desafios como, por exemplo, migração de máquinas virtuais, acordos SLA e provisionamento de serviços automatizados.

Em uma visão simplista a economia de energia pode ser obtida desligando os recursos ociosos. Contudo, isso tem algumas implicações, um recurso físico desligado leva um intervalo de tempo T para ser inicializado, quando necessário, e esse tempo deve estar previsto pelas políticas de provisionamento de serviços sob demanda e nos acordos SLA. Em um ambiente muito dinâmico, recursos físicos ociosos que acabaram de receber uma ordem para serem desligados podem logo em seguida receber um pedido para entrarem em atividade novamente;

¹⁵ Uma "imagem" é basicamente um arquivo que representa o disco rígido da máquina virtual.

neste pior cenário, ao invés de uma economia de energia houve um desperdício de energia e prováveis riscos de violações do SLA. Uma solução para esses problemas é colocar o recurso em uma fase de hibernação, na qual uma quantidade de energia é consideravelmente economizada e o recurso consegue ser inicializado de forma mais rápida. Porém, ainda consome um instante de tempo t (sendo $t < T$) que também precisa ser considerado.

Diversas pesquisas recentes buscam soluções para a questão energética da computação em nuvem (YANG, 2013) (HSU *et al.*, 2012) (MEZMAZ *et al.*, 2011) (BENEDICT, 2012) (FELLER; RILLING; MORIN, 2011) (KATSAROS *et al.*, 2012) (BELOGLAZOV; ABAWAJY; BUYYA, 2012a). Técnicas de monitoramento e otimização podem ser consideradas em busca de soluções para esse problema. Entretanto, tais soluções devem ser aplicadas tanto do lado dos clientes quanto dos provedores de serviços. Por exemplo, enquanto o cliente estiver com a sua máquina virtual ativa, mesmo que ociosa, o provedor de serviços comercial não irá desligá-la, pois ele está recebendo por aquela utilização, no máximo ele poderá migrá-la e muito dificilmente ele irá diminuir os seus recursos.

4.8 Considerações Finais

As vantagens proporcionadas pela adoção da computação em nuvem, mais especificamente pelo uso da virtualização, permitem que os objetivos deste projeto sejam alcançados. Outro ponto motivador para este projeto em relação à computação em nuvem é dado pelo fato deles se complementarem (Figura 10). Em outras palavras, as características da computação em nuvem facilitam e agilizam a autoconfiguração dos mecanismos propostos; em contrapartida, esses mecanismos permitem que a computação em nuvem seja empregada de forma mais eficiente, superando desafios e procurando atender às necessidades dos administradores e clientes em uma forma mais automática.

No próximo Capítulo é apresentada uma coleção de trabalhos relacionados que propõem soluções em comum a este projeto de doutorado.

TRABALHOS RELACIONADOS

5.1 Considerações Iniciais

Neste capítulo alguns trabalhos relacionados são apresentados com o objetivo de demonstrar a abordagem adotada por pesquisadores para problemas que podem ser solucionados por meio de mecanismos de autoconfiguração e auto-otimização. Alguns exemplos desses problemas podem envolver basicamente, mas não se limitam, aos seguintes tópicos:

- **Gerenciamento de recursos ou serviços:** Neste caso, o gerenciamento ou controle envolve recursos e serviços tanto computacionais (*hardware* e *software*) quanto de outros tipos relacionados de alguma forma à computação (usuários, acordos, energia, etc.).
- **Descoberta de recursos ou serviços:** Além do gerenciamento de recursos, a descoberta ou seleção de recursos ou serviços adequados constituem tópicos relacionados. É possível acrescentar outras implicações a este tópico, pois, além da descoberta, a decisão de qual recurso ou serviço utilizar entre vários pode ser automática baseada em questões relacionadas à SLA, QoS e econômicas. Dessa forma, esse tópico pode se expandir para áreas que empregam: tomada de decisão, personalização de informações ou de contexto, políticas financeiras, etc.
- **Otimização de recursos ou serviços:** Os recursos ou serviços podem ser otimizados de alguma forma. Por exemplo, técnicas de otimização são utilizadas para minimizar o custo tendo o desempenho do sistema como restrição. Na visão do cliente*¹⁶ da nuvem, isso significa identificar e alocar o menor número de máquinas virtuais (cobrança por hora/máquina) garantindo o desempenho esperado (ou estabelecido em SLA) para seus usuários*¹⁶.

¹⁶ * A partir deste ponto em diante, os termos cliente e usuário terão contextos diferentes neste trabalho. Dessa forma, o “cliente” é o indivíduo ou empresa que utiliza ou contrata os serviços em nuvem de um provedor para

Há várias propostas das mais diversas áreas que abordam soluções aos tópicos destacados. Contudo, o número de propostas se torna limitado quando se consideram soluções que envolvam todos eles em conjunto, em uma mesma arquitetura. Acrescenta-se ainda que considerar tais tópicos dentro do contexto da computação em nuvem pode acarretar tanto em vantagens quanto em desvantagens. Conforme observado no capítulo anterior, a computação em nuvem é capaz de proporcionar facilidades e agilidade no gerenciamento de recursos, principalmente, por conta da virtualização.

Contudo, alguns pesquisadores (ZHANG; CHENG; BOUTABA, 2010) (DILLON; WU; CHANG, 2010) (RIMAL; CHOI, 2012) consideram a existência de vários desafios ainda à serem solucionados na computação em nuvem. Conforme comentado em capítulos anteriores, a computação autônoma é uma opção que possibilita atacar tais desafios dentro dos tópicos destacados. Embora alguns autores considerem que a área da computação autônoma ajudou de alguma forma no surgimento da computação em nuvem, sendo parte de sua composição (BUYYA; BROBERG; GOSCINSKI, 2011). Em outros trabalhos são apresentados desafios em aberto para arquiteturas de computação em nuvem autônomas (BUYYA; CALHEIROS; LI, 2012). Tais desafios estão relacionados com: (1) a prestação de QoS e o cumprimento de acordos SLAs, (2) eficiência energética e (3) questões de segurança.

No trabalho de Abdelmaboud et. al. (2015) (ABDELMABOUD *et al.*, 2015) foi realizado um mapeamento sistemático com o intuito de fazer um levantamento sobre as atuais abordagens em QoS na computação em nuvem, o objetivo do estudo foi identificar onde deve ser colocada mais ênfase nas pesquisas atuais e nortear as futuras pesquisas. Nesse trabalho, 67 artigos foram selecionados na recente literatura como estudos primários que foram classificados em relação ao foco da citada pesquisa, ou seja, diversos trabalhos que tratam sobre o tema QoS na computação em nuvem. Quase metade dos artigos selecionados por esse trabalho (48%) envolviam a pesquisa na área de IaaS. Os autores destacam a importância do estudo em abordagens para prover QoS na computação em nuvem nos últimos anos e concluem o artigo afirmando que existem desafios em abertos e lacunas a serem exploradas em futuras pesquisas. Mais especificamente na parte de IaaS, os autores destacam que os requisitos de QoS são de grande importância para os clientes, uma vez que eles pagam para acessar os serviços da nuvem (ABDELMABOUD *et al.*, 2015). Além disso, os autores destacam outro aspecto, que vai de encontro com o que é proposto neste trabalho de doutorado, quando afirmam que os provedores de serviços em nuvem tem pouco ou nenhum suporte para o gerenciamento de recursos baseado em negócios.

Citando Buyya et. al. (2009) (BUYYA *et al.*, 2009) os autores (ABDELMABOUD *et al.*, 2015) ainda destacam que a negociação de QoS dinâmica se faz necessária entre os provedores e os consumidores para que os SLAs suportem a alocação de recursos automática a fim de satisfazer várias requisições. Ainda para esses autores, três desafios devem ser observados

prestar serviços para seus “usuários” finais. Por exemplo, uma empresa que disponibiliza uma aplicação Web para seus usuários utilizando serviços PaaS e/ou IaaS.

pelos pesquisadores: 1) Gerenciamento de recursos baseados no SLA, 2) Monitoramento para gerenciamento de recursos e desempenho e 3) Escalabilidade de recursos. No primeiro desafio os clientes não tem autoridade e controle sobre a gestão dos recursos e, assim, eles não têm certeza se os SLAs serão mantidos. O segundo desafio está relacionado a necessidade de um monitoramento eficiente para o gerenciamento de recursos e desempenho evitando adversidades que impactam a QoS. O terceiro desafio está na dificuldade em alcançar uma escalabilidade automática confiável que consiga escalar aumentando e diminuindo recursos com uma eficiência de custo sem a violação do SLA.

Uma vez identificada a necessidade de mais pesquisas nesta área, este trabalho de doutorado propõe a pesquisa e utilização da computação autônoma e recursos da Web Semântica na criação de mecanismos para o gerenciamento de arquitetura virtualizadas que visam a provisão de QoS. Embora os seus conceitos da computação autônoma, baseados no sistema nervoso autônomo, não sejam complicados de serem entendidos, o desenvolvimento e o emprego dos mesmos podem ser tornar uma tarefa complexa dependendo das teorias e tecnologias adotadas. Por exemplo, para a realização do “laço de controle” (*control loop*) MAPE-K (*Monitor, Analyse, Plan, Execute e Knowledge*) alguns *frameworks* utilizam técnicas de IA e a teoria do controle (KHALID *et al.*, 2009). Diversas técnicas de *feedbacks* (CAPRARESCU; PETCU, 2009), *data cluster* (KHALID *et al.*, 2009), otimização (BUYA; CALHEIROS; LI, 2012), aprendizado de máquina, uso de regras e políticas (STATHIS, 2010) e Web Semântica (ANGHEL *et al.*, 2010) também são possibilidades que podem ser consideradas.

Dessa forma, alguns trabalhos relacionados estão organizados e divididos em três seções neste capítulo. Na seção 5.2 estão os trabalhos focados em Web Semântica/Ontologia para a Computação em Nuvem ou a Computação Autônoma. Na seção 5.3 estão os trabalhos sobre computação autônoma envolvidos com a Computação em Nuvem ou com Web Semântica/Ontologia. Finalmente, na seção 5.4 estão listados os trabalhos relacionados que envolvem as três áreas; Web Semântica, Computação Autônoma e Computação em Nuvem.

5.2 Web Semântica

Em Loutas et. al. (2010) é proposta uma referência de arquitetura para interoperabilidade entre nuvens por meio de semântica. Essa referência, denominada *Reference Architecture for Semantically Interoperable Clouds - RASIC* relaciona os conceitos de SOA e Web semântica com o objetivo da criação de uma arquitetura em camadas, capaz de prover interoperabilidade entre os serviços de provedores. Os conflitos de interoperabilidade semântica ocorrem na maioria dos casos, especialmente em nuvens híbridas (LOUTAS *et al.*, 2010). Na camada semântica os autores recomendam aos provedores que utilizem os modelos de serviços e recursos propostos e que sejam formalmente expressos por ontologias construídas em linguagens padronizadas como OWL ou RDF. Dessa forma, o mecanismo semântico de interoperabilidade poderá atuar

em tempo de execução. Os autores incluem ainda componentes para anotações (*Annotations*) em serviços e recursos. Como cada provedor pode criar a sua ontologia, possíveis conflitos são resolvidos por técnicas de raciocínio responsáveis por garantir a correspondência semântica entre conceitos equivalentes.

Analisando esse trabalho percebe-se que, com o foco na interoperabilidade entre nuvens, os autores propõe uma padronização que permite a *descoberta de recursos ou serviços* em vários provedores e consequentemente diminuem a *dependência dos provedores* (desafio), dando opção ao cliente, ou usuário, de selecionar um determinado serviço naquele provedor que oferecer o menor custo ou a melhor QoS. Finalmente, essa interoperabilidade permite criação de nuvens híbridas e comunitárias de forma mais direta e opcionalmente provedores públicos e comerciais podem se associar mais facilmente, compartilhando seus recursos e serviços para seus clientes em comum, com o objetivo de garantir o SLA. Outras propostas semelhantes para a interoperabilidade entre provedores baseada em semântica são apresentadas em (EJARQUE *et al.*, 2011), (LOUTAS; KAMATERI; TARABANIS, 2011) e (NELSON; UMA, 2012).

Ainda relacionado ao tema de *descoberta de recursos ou serviços*, o trabalho de Di Modica et. al. (2012) propõe um modelo semântico que auxilia os clientes e provedores na caracterização de suas demandas ou ofertas. Duas ontologias (*Resource features' ontology* e *Application requirements' ontology*) foram criadas em OWL para representar cada domínio e semanticamente caracterizar os recursos oferecidos pelos provedores e os requisitos dos clientes (MODICA; TOMARCHIO, 2012). Para relacionar o domínio (ontologia) dos requisitos de aplicação com o domínio das características de recursos utilizou-se um processo denominado “mapeamento” (*mapping*) por meio de regras definidas em SWRL. Ao contrário do trabalho citado anteriormente, neste fica claro o processo de implementação e resultados dos testes efetuados são apresentados.

Em Zhang et. al. (2012) um sistema baseado em ontologia em OWL foi criado para a *descoberta de serviços* em nuvens IaaS. A ontologia CoCoOn (*Cloud Computing Ontology*) define o modelo da camada IaaS e descreve os seus serviços. A descoberta pode ser baseada na funcionalidade e em atributos de QoS do serviço IaaS (processamento, armazenamento e rede) (ZHANG *et al.*, 2012).

No trabalho de Dastjerdi et. al. (2012) uma ontologia criada em WSMO (WSMO, 2013) que contém informações de QoS foi utilizada por um sistema de monitoramento de contratos SLAs entre nuvens. Os autores discutem questões de violação de contrato SLA e apresentam uma avaliação de desempenho. Nessa avaliação eles analisam o “tempo de implantação” (*deployment time*) que é a soma dos tempos de descoberta e classificação dos serviços na monitoração, os autores afirmam que esse resultado do tempo é razoável (aproximadamente 10 segundos) para o que consideram um grande número de serviços no repositório (120) serviços (DASTJERDI; TABATABAEI; BUYYA, 2012).

Haase et. al. (2010) apresentam mais especificamente um componente (*The Intelligence*

Edition) de um conjunto de produtos denominado "eCloudManager". Este conjunto, desenvolvido na linguagem Java, atua no gerenciamento empresarial de ambientes em nuvem. O componente faz uso de tecnologias semânticas para integrar recursos disponíveis em um repositório de dados semântico, analisando esses dados e interagindo de forma colaborativa. Dessa forma, as principais contribuições são: a integração de dados, a documentação e anotação colaborativa e o acesso e análise inteligente de informações (HAASE *et al.*, 2010). Para alcançar essas contribuições foi criada uma ontologia OWL 2 que representa o domínio empresarial de gerenciamento da nuvem. A linguagem SPARQL é utilizada para a realização de consultas na ontologia e existe uma interface para a apresentação dos recursos. Os autores comentam sobre resultados impressionantes como, por exemplo, utilizando o "eCloudManager" eles conseguiram o provisionamento de recursos para uma equipe de 1500 desenvolvedores em minutos o que geralmente levaria semanas. Embora isso seja interessante, muitas das informações são passadas de forma abstrata no artigo, provavelmente pelo fato da proposta ser um produto.

No trabalho de Xu *et al.* (2013) são propostos um modelo e uma arquitetura para gerenciamento de serviços em nuvem (MaaS - *Management as a Service*). O foco desse trabalho está na implementação otimizada de diferentes serviços de gerenciamento de rede. Para isso uma ontologia criada em OWL representa o domínio de gerenciamento de rede (XU *et al.*, 2013). Em Jeong e Hong (2011) é proposto um método de gerenciamento para controlar recursos na computação em nuvem. Nesse trabalho os autores também fazem uso de ontologias, porém ela é associada a um modelo baseado em agentes (JEONG; HONG, 2011). Wei e Junpeng (2010) apresentam uma ontologia com o mesmo propósito (WEI; JUNPENG, 2010). Youseff (2008) apresenta uma ontologia conceitual sobre todo o contexto de computação em nuvem (YOUSEFF; BUTRICO; SILVA, 2008).

Em outros trabalhos (HU *et al.*, 2009), (BERNABE *et al.*, 2012), ontologias OWL e regras são utilizadas para realizar controle de acesso baseado em semântica. Em Zhou *et al.* (2010a) é proposta uma abordagem baseada em ontologia para a reengenharia de *softwares* empresariais, permitindo identificar potenciais serviços que podem ser migrados para a nuvem (ZHOU; YANG; HUGILL, 2010). Em outro trabalho, duas ontologias são apresentadas e comparadas durante a busca por serviços em nuvem (KANG; SIM, 2011a).

O trabalho de Kalou *et al.* (2010) não está dentro do contexto da computação em nuvem, mas aborda o uso de Web Semântica (ontologias e regras) para a personalização de usuários. Essa abordagem pode ser estendida para personalizar os serviços de clientes e usuários da computação em nuvem (KALOU *et al.*, 2010).

No trabalho de Kang e Sim (2011b) é apresentado um protocolo de descoberta de serviço em nuvem baseado em agentes, os autores utilizam uma ontologia para representar a nuvem e utilizaram a capacidade semântica de "raciocinar" sobre essa ontologia para avaliar e descobrir serviços na nuvem (KANG; SIM, 2011b). Ainda nesse trabalho foi proposta uma arquitetura na qual os clientes podem descobrir e combinar os recursos existentes para alcançar as suas

exigências. Embora os autores destaquem a eficiência de sua abordagem, ela somente visa a descoberta e utilização de serviços, não há menção sobre o gerenciamento de serviços em relação a QoS ou a garantia de SLA. Identificar as configurações ou serviços adequados somente é interessante se a QoS seja garantida sem a violação de SLA no decorrer da utilização dos recursos da nuvem.

A proposta de Yongsiriwit, Assy e Gaaloul (2016) consiste na proposta de uma estrutura semântica para modelos configuráveis BPaaS (*Business Process as a Service*). Esses autores utilizaram as tecnologias de Web Semântica e técnicas de mineração de dados para criar uma representação abstrata de base ontológica de modelos BPaaS configuráveis e, assim, extrair diretrizes de configuração de repositórios de processos existentes. Dessa forma, os autores combinaram o uso de ontologias OWL e BPMO (*Business process modeling ontology*) com recursos da Web Semântica (SWRL, SPARQL) e técnicas de mineração de dados para criar um *framework* para o gerenciamento compartilhado de conhecimentos de uma base BPaaS destinada a modelos de processos baseados em nuvem. Os autores destacam que esse *framework* semântico pode ajudar as empresas na redução de custos no desenvolvimento e manutenção de processos, no uso da infraestrutura de TI e proporcionar um menor consumo de recursos com maior economia de energia (YONGSIRIWIT; ASSY; GAALOUL, 2016).

Os artigos citados nesta seção não abordaram as três áreas relacionadas com este projeto de doutorado, limitaram-se às áreas de web semântica e computação em nuvem. Algumas propostas solucionam apenas um ou dois tópicos dos três destacados no início do capítulo, algumas não estão dentro do contexto de IaaS e poucas apresentaram resultados de desempenho.

5.3 Computação Autônoma

Em Buyya et. al. (2012) são discutidos desafios para computação em nuvem autônoma que focam principalmente na QoS, eficiência energética e questões de segurança. Além dessa discussão, os autores apresentam uma arquitetura para o gerenciamento autônomo da nuvem destinada às aplicações SaaS. A proposta é o desenvolvimento de um sistema autônomo de gerenciamento e algoritmos para a provisão dinâmica de recursos baseados nos requisitos de QoS dos usuários, maximizando a eficiência e minimizando o custo. Além disso, também abordam a criação de mecanismos de segurança, principalmente contra ataques DDoS, permitindo que os recursos alocados sejam apenas para requisições legítimas de usuários. O foco do sistema autônomo proposto pelos autores está na parte de PaaS da arquitetura.

Ainda segundo esses autores, essa camada coordena os recursos da nuvem de acordo com os requerimentos da camada SaaS (QoS). Dessa forma, a camada PaaS escalona aplicações e recursos satisfazendo a QoS exigida pelos clientes e também não torna o provisionamento tão custoso para o provedor PaaS. O gerenciador autônomo é composto por escalonadores de aplicação e eficiência energética que utilizam algoritmos de escalonamento para aplicações

multiobjetivo e são baseados em heurísticas relacionadas aos atributos de QoS e técnicas de otimização para problemas multiobjetivo com conflitos (SLA). O gerenciador ainda possui um componente de algoritmos de provisionamento de recursos dinâmico composto por algoritmos de predição e outro componente de segurança e detecção de ataques que utiliza técnicas para tentar evitar ataques DDoS no contexto de sistemas em nuvem (BUYYA; CALHEIROS; LI, 2012). Esse trabalho possui algumas semelhanças com este projeto de doutorado, porém difere em vários pontos, por exemplo, os autores não consideram o uso de web semântica e ontologia, os conceitos sobre computação autônoma estão ausentes ou não foram comentados, não há informações sobre o laço de controle (MAPE-K), e a abordagem está relacionada com a camada PaaS enquanto o foco principal deste trabalho de doutorado está na camada IaaS.

No trabalho de Tchana et. al. (2013) são propostas três estratégias de implementação para o gerenciamento de recursos virtualizados no modelo IaaS. A primeira estratégia é implementada apenas no nível de virtualização, a segunda apenas no nível de aplicação e a terceira em ambos. Dessa forma, a proposta dos autores é o gerenciamento autônomo de recursos em dois níveis considerando uma nuvem privada (TCHANA et al., 2013). Para automatizar a execução dessas estratégias os autores utilizaram um sistema de gerenciamento autônomo denominado *TUNe* (BROTO et al., 2008).

Em Solomon et. al. (2010) foi proposta uma abordagem de como projetar e desenvolver sistemas de gerenciamento autônomos. Os autores descrevem as etapas de desenvolvimento e citam um exemplo da criação de uma estratégia de *self-optimization* para a computação em nuvem (SOLOMON et al., 2010).

Emeakaroha et. al. (2012) apresentam um detector autônomo de violação de acordos SLA para infraestruturas em nuvem denominado DeSVi (*Detecting SLA Violation infrastructure*). A arquitetura proposta pelos autores utiliza o *framework* LoM2His (*Low Level Metrics to High Level SLAs*) (EMEAKAROHA et al., 2010) que detecta violações de SLA em tempo de execução e estende o projeto FoSII (*Foundations of Self-governing Infrastructures*) (BRANDIC, 2009). O projeto FoSII considera os princípios da computação autônoma para o desenvolvimento de modelos e conceitos de autogerenciamento de SLA em nuvem (EMEAKAROHA et al., 2012). Embora este projeto de doutorado também adote os princípios da computação autônoma, há diversas diferenças entre esses trabalhos. Por exemplo, este projeto não se limita apenas ao gerenciamento de SLA, outros fatores como custo, economia e opções personalizadas dos clientes podem ser consideradas. Além disso, os mecanismos utilizados para atender ao conceito da computação autônoma são diferentes. Por exemplo, o projeto FoSII considera a utilização de bases CBR (*Case-Based Reasoning*) para o gerenciamento do conhecimento e resolver problemas com base em experiências passadas (EMEAKAROHA et al., 2012).

Neste projeto de doutorado, a proposta é tentar inovar, no contexto de computação em nuvem, empregando recursos de web semântica, utilizando uma ontologia para representar o conhecimento e regras e racionalizadores (*reasoners*) para a tomada de decisão.

Em Kertesz et. al. (2012) também é proposta uma arquitetura autogerenciável (*self-manageable*) para serviços de virtualização interoperáveis baseados em SLA. Novamente princípios da computação autônoma são utilizados em conjunto com extensões *SLA-aware* para alcançar uma operação confiável e eficiente do serviço em ambientes distribuídos (KERTESZ; KECSKEMETI; BRANDIC, 2012).

No trabalho de Yeo et. al. (2010) é proposto um mecanismo de preços autônomo para serviços da computação utilitária. Esse mecanismo ajusta de forma automática os preços conforme é necessário permitindo o aumento da receita. Segundo os autores, configurando os preços dos recursos de acordo com os requisitos do cliente, pode ser vantajoso para a computação utilitária, pois clientes diferentes possuem necessidades específicas a serem cumpridas e estão dispostos a desembolsar preços variados para alcançá-las (YEO *et al.*, 2010). Manipular preços pode ser muito interessante, pois embora exista a impressão de que os recursos são infinitos, para o cliente o orçamento pode ser bem limitado. Por exemplo, para esses clientes alguns provedores comerciais de nuvem pública disponibilizam um serviço de alerta (*Billing Alerts*) que avisa quando o consumo atingiu certo valor estipulado pelo cliente. Se o comportamento desse cliente é finalizar sua instância e aguarda o próximo mês, o provedor pode promover uma promoção para esse usuário e diminuir o valor da hora máquina até o final desse mês. Diversas outras políticas podem ser sugeridas, porém mecanismos precisam dar suporte ao funcionamento das mesmas.

Em outros trabalhos relacionados às computações em nuvem e a autônoma, foram propostos, um protocolo de roteamento para o gerenciamento de colaboração de serviço entre provedores de nuvem (ITANI *et al.*, 2012), a criação de uma mecanismo autônomo de planejamento de capacidade para sistema de computação em nuvem (AMORETTI; ZANICHELLI; CONTE, 2013) e uma abordagem de gerenciamento de energia (*Energy-Aware online* para aplicativos de HPC em plataforma de computação consolidadas (RODERO *et al.*, 2010).

Maurer et. al. (2013) apresenta uma configuração de recursos adaptativa para o gerenciamento de infraestrutura em nuvem. Nesse novo trabalho, os autores novamente utilizaram o projeto FoSII, porém desta vez, além do CBR, eles adotam um método baseado em regras (fora do contexto da web semântica ou ontologia) para investigar violações de SLA. Segundo os autores, no contexto de técnicas que lidam com a reconfiguração de máquinas virtuais em infraestruturas em nuvem, os resultados provaram que o método baseado em regras é melhor do que o CBR tanto no respeito ao SLA quanto em desempenho (MAURER; BRANDIC; SAKELLARIOU, 2013).

No trabalho de Mohamed et. al. (2016) (MOHAMED; BELAÏD; TATA, 2016) é apresentado um modelo que abrange todas as funções definidas do ciclo MAPE-K para o provisionamento de recursos em nuvem autônomo baseado no *Open Cloud Computing Interface* (OCCI). Assim, os autores descrevem quais são os recursos necessários para tornar autônomo um determinado recurso na Nuvem, independentemente do nível de serviço (IaaS, PaaS ou SaaS). Nessa

proposta, alertas são gerados quando a QoS não é ou não será respeitada, com base nesses alertas o sistema de planejamento poderá executar alguma reconfiguração. As estratégias utilizadas para tal planejamento são extraídas do SLA baseadas em regras do tipo: "evento-condição-ação". Os autores propõem essa infra-estrutura autônoma para gerenciar processos de negócios baseados em Serviços (*Service-based Business Processes* - SBPs) de forma elástica.

Em Mehrotra et. al. (2016) (MEHROTRA *et al.*, 2016), os autores afirmam que a alocação eficiente de recursos computacionais para serviços é um desafio ainda predominante no ambiente de computação em nuvem. Assim, eles apresentam a proposta de um corretor de nuvens (*Cloud Broker*) para o gerenciamento de desempenho em um ambiente de computação em nuvem federada. Para isso, há um controle distribuído que fornece características de computação autônoma que possui o seu laço de controle (*loop control*) baseado em *feedbacks*. Além, do desempenho a proposta dos autores também visa a manutenção dos acordos de níveis de serviços (SLAs), para isso é realizada a alocação dinâmica de recursos. Nesse cenário, a abordagem também visa maximizar o lucro, tanto para o corretor de nuvem quanto para os prestadores de serviços ao calcular a atribuição de recursos adequada.

Nesta seção foram listados alguns artigos relacionados a este projeto de doutorado que estão interseccionados nas áreas de computação autônoma e computação em nuvem. Alguns desses trabalhos recentes possuem semelhanças com este projeto devido à motivação dos pesquisadores em tentar propor soluções para as questões consideradas como desafios em ambas as áreas. Na próxima seção, alguns trabalhos que relacionam as três áreas são discutidos.

5.4 Computação Autônoma, Computação em Nuvem e Web Semântica

Anghel et. al. (2010) propõem um algoritmo para eficiência energética em centro de serviços. Embora os autores não tenham mencionado o termo "computação em nuvem" e nem considerado o uso de máquinas virtuais, esse trabalho pode ser aplicado nos *data centers* dos provedores de nuvens. O algoritmo baseado em contexto (*context aware*) proposto pelos autores propõe soluções para a eficiência energética em tempo de execução por meio da auto-adaptação que é proporcionada pela implementação das quatro fases do laço de controle MAPE.

As informações são coletadas do contexto seguindo a metodologia GAMES (*Green Active Management of Energy in IT Service Centers*) e, em seguida, são representadas utilizando um modelo que os autores denominam como RAP (*Resources, Actions Policies*). As políticas existentes no RAP são um conjunto de condições ou regras descritas em XML que na fase de *Análise* são convertidas em regras SWRL. Essa conversão é feita, pois o modelo RAP define duas representações equivalentes de informações de contexto, sendo uma representação baseada em conjunto e outra representação baseada em ontologia. A primeira representação é utilizada para avaliar as mudanças de energia e desempenho do centro de serviço e a segunda

é utilizada para determinar se os indicadores GPI (*Green Performance Indicator*) e KPI (*Key Performance Indicator*) estão sendo atendidos em tempo real. Para a avaliação de uma política os autores utilizaram o racionalizador Pellet. Outro ponto interessante desse trabalho é que os autores monitoraram tanto os recursos computacionais quanto o ambiente do centro de serviços, considerando, por exemplo, a temperatura do ambiente, umidade, ar condicionados, controle de iluminação, etc. (ANGHEL *et al.*, 2010).

Em Nicolicin-Georgescu *et al.* (2010) é apresentada uma abordagem para gerenciar alocação de *cache* em *data warehouse* pelo uso de sistemas autônomos que são auxiliados por tecnologias da web semântica. Embora neste artigo não haja relação direta com a computação em nuvem, trata-se de mais um caso onde a web semântica, ontologia (OWL) e regras servem de base para a criação do laço de controle proposto no gerenciamento autônomo. Um ponto interessante deste trabalho foi a adoção dos laços MAPE-K em hierarquia, se baseado no modelo da arquitetura DSS (*Decision Support System*) que é utilizado no gerenciamento em *data warehouses*. O laço MAPE-K proposto pelos autores onde existe um laço de controle para cada nível da hierarquia (Servidor Físico e Base OLAP (*On-line Analytical Processing*)) (NICOLICIN-GEORGESCU *et al.*, 2010).

Em Famaey *et al.* (2011) é apresentado um *framework* hierárquico eficiente e escalável para a disseminação de contexto em sistemas computacionais e de comunicação em larga escala. Elementos Autônomos (EA) são estruturados de forma hierárquica, aqueles no topo da hierarquia possuem uma visão geral do ambiente e executam tarefas de gerenciamento em um nível mais alto, os EAs subjacentes realizam tarefas de configurações de baixo nível, pois possuem uma visão detalhada de um subconjunto do ambiente (FAMAHEY *et al.*, 2011). Ainda nesse trabalho, os autores consideram modelos semânticos e *reasoners* são utilizados para facilitar o correto entendimento e interpretação de informações, metas e ações. O *framework* é considerado como sendo uma arquitetura de gerenciamento autônoma responsável pela disseminação de uma enorme quantidade de informações do contexto entre os EAs. A semântica é empregada para filtrar de forma eficiente esse contexto, baseado no seu significado e metas de gerenciamento do EA. Dessa forma, quando o ambiente gerenciado sofre alterações, os filtros de contextos são automaticamente e dinamicamente adaptados pelo *framework*.

Ainda segundo os autores, esse processo dinâmico garante que o contexto correto seja entregue no local correto, no tempo estipulado. Dessa forma, a sobrecarga relacionada ao contexto é reduzida e informações desnecessárias não são distribuídas entre os AEs (FAMAHEY *et al.*, 2011). Apesar desse trabalho apresentar uma solução para o gerenciamento em federações de nuvens (*Federated Clouds*), muitos dos desafios propostos para as computações em nuvem não foram abordados como, por exemplo, gerenciamento eficiente de energia, políticas orçamentárias (custo x benefício ou ROI), acordos SLA, etc. A contribuição pode ser resumida em um mecanismo inteligente, eficiente e escalável de disseminação de informações entre elementos distribuídos que possivelmente podem ser utilizadas para atacar os desafios propostos.

Em um outro trabalho (LATRÉ *et al.*, 2013) os autores do trabalho anterior (FAMAEY *et al.*, 2011) ainda continuam utilizando recursos da web semântica e conceitos da computação autônoma para a disseminação de contexto. Contudo, em um contexto que não aborda computação em nuvem diretamente, o foco desse artigo está em redes colaborativas autônomas. Uma contribuição interessante desse artigo é a comparação entre diferentes configurações do *reasoner* e utilização ou não de modelos de sumarização (resumo) de informações. Quando a quantidade de informações de contexto é volumosa, realizar um processo de sumarização contribui em desempenho para o processamento semântico realizado pelos *reasoner*. Por fim, os autores afirmam que há um *trade-off* durante a escolha dos melhores conjuntos de configuração entre a expressividade e desempenho.

No trabalho de Dautov *et. al.* (2013) é apresentada a proposta de fornecer autogerenciamento para as plataformas de aplicações da computação em nuvem (PaaS). Os autores apresentam uma nova visão onde as plataformas em nuvem são vistas como as redes de fontes de dados distribuídas (rede de sensores). A proposta dos autores é utilizar os conceitos de computação autônoma (MAPE-K) e de redes de sensores semânticos SSW (*Semantic Sensor Web*) no contexto da computação em nuvem (DAUTOV *et al.*, 2013). Dessa forma, os autores propõem um *framework* composto por elementos gerenciáveis e gerenciador autônomo. O gerenciador autônomo recebe as informações dos elementos gerenciáveis que passam por um mecanismo de "triplicação" (*triplication engine*) que é responsável por consumir e tornar homogêneos os dados gerados pelos componentes de plataforma, aplicações implantadas, serviços externos, etc. Assim, esse mecanismo recebe dados brutos que são transformados em triplas RDF, que podem ser associadas a ontologias OWL e serem então utilizadas para consultas. Após o processo de transformação, esses dados em triplas RDF são enviados para um componente de consulta contínua baseada na linguagem SPARQL denominado "*continuous SPARQL query engine*" que executa as consultas nas triplas. Em seguida os resultados das consultas passam por máquinas de inferência ou racionalização que utilizam informações da ontologia OWL e regras SWRL para determinarem se executam ou não uma ação de adaptação. Esse trabalho não apresenta resultados de testes ou uma avaliação de desempenho e, da mesma forma que citado anteriormente, não abordam diretamente os desafios da computação em nuvem propostos na literatura.

Em Jeffery *et. al.* (2015) é apresentada uma solução baseada em um corretor (*Broker*) para o gerenciamento de serviço em nuvem auto-adaptável e preocupado com o cumprimento e regulamentação da legislação. Para isso, os autores propõem um gerenciador autônomo que integra o laço de controle MAPE-K com o *framework* LEGEX (*Legislation Execution*) utilizado para o cumprimento e verificação de normas jurídicas (leis). Assim, os autores buscam atingir dois objetivos principais: 1) Detectar de ante-mão se um serviço de nuvem viola as leis, regulamentos, políticas, contratos ou acordos de níveis de serviços (SLA) e 2) Avisar os prestadores de serviços e o corretor (*broker*) da nuvem no caso de legislações que devem introduzir novos requisitos de serviços, novas privacidades, políticas de segurança, etc. Para isso, eles utilizam um mecanismo que extrai conhecimento jurídico criado com ferramentas NLP (*Natural Lan-*

guage Processing - Processamento de Linguagem Natural) e ontologias jurídicas voltadas para minimizar a modelagem de fontes legais e também para melhorar a consulta por Web Semântica em documentos legais (JEFFERY *et al.*, 2015). Embora relacionar a computação autônoma, computação em nuvem e Web Semântica esse artigo não relata resultados de desempenho e eficiência da alocação de recursos, apesar de envolver um assunto interessante e complexo voltado para a legislação, esse artigo ainda não propõe soluções diretas para os desafios da computação em nuvem propostos na literatura.

5.5 Considerações Finais

Este capítulo apresentou uma revisão de alguns dos trabalhos relacionados que podem ser encontrados na literatura. Os artigos selecionados para compor este capítulo estão relacionados com pelo menos duas das três áreas em que este trabalho está imerso conceitualmente (Web Semântica, Computação Autônoma e Computação em Nuvem - IaaS).

Na pesquisa por trabalhos relacionados foram considerados artigos publicados em revistas e conferências nos últimos anos. Além da busca por artigos que se encaixassem em pelos duas das três áreas relacionadas, procurou-se focar naqueles que estão associados com pelo menos um dos três tópicos citados no início deste capítulo, pois o objetivo da pesquisa é a criação de mecanismos autônomos que operam com recursos da Web Semântica na busca por garantir QoS em arquiteturas virtualizadas na computação em nuvem.

Dessa forma, as diversas propostas de gerenciamento e otimização, muitas delas não autônomas ou não automáticas, que constituem soluções válidas para a computação em nuvem não foram selecionadas. Da mesma forma ficaram de fora as soluções comerciais para arquiteturas ou plataformas, por não revelarem detalhes de conceitos, implementações e tecnologias utilizadas. A Tabela 2 exibe um resumo de alguns dos trabalhos relacionados com as suas principais características relacionadas a este projeto de doutorado.

Tabela 2 – Lista Resumida de Trabalhos Relacionados

Trabalhos Relacionados								
Trabalho	MAPE- K		Web Semântica		VM/RM	Gerenciamento		
	Conceitos	Conhecimento	Ontologia	Regras		Econômico	QoS/SLA	Otimização
Emekaro et. al. (2012)	Sim	Bases CBR	Não	N/A	Sim	N/A	Sim	N/A
Anghel et. al. (2010)	Sim	Ontologia	Sim	SWRL	Sim	N/A	Sim	N/A
Buyya et. al. (2012)	Sim	N/A	Não	N/A	Sim	N/A	Sim	Sim
Famaey et. al. (2011)	Sim	Ontologia	Sim	SWRL/Jena	Sim	N/A	Sim	N/A
Latré et. al. (2013)	Sim	Ontologia	Sim	SWRL/Jena	Sim	N/A	Sim	N/A
Dautov et. al. (2013)	Sim	Ontologia	Sim	SWRL	N/A	N/A	Sim	N/A
Jeffery et. al. (2015)	Sim	Ontologia	Sim	NLP	N/A	N/A	Sim	N/A
Projeto	Sim	Ontologia	Sim	SWRL	Sim	Sim	Sim	Sim

O próximo capítulo aborda a proposta sistematizada para este projeto de doutorado.

MECANISMOS DE AUTOCONFIGURAÇÃO E AUTO-OTIMIZAÇÃO

6.1 Considerações Iniciais

Em um ambiente em nuvem, uma arquitetura virtualizada pode ser considerada como sendo a infraestrutura que é provisionada diretamente pelo o provedor da nuvem. As ações nessa arquitetura é regida pelas regras previamente estabelecidas entre os clientes e os provedores de serviços em nuvem. Neste contexto, os mecanismos propostos de AutoConfiguração e Auto-Otimização podem ser acoplados nessa arquitetura como sendo novos módulos, definidos a seguir:

- **Módulo de AutoConfiguração (MAC):** Esse mecanismo de autoconfiguração é utilizado para reconfigurar a infraestrutura virtualizada do cliente visando a garantia dos requisitos de QoS estabelecidos no SLA ou para atender alguma personalização do cliente. Dessa forma, esse mecanismo é responsável pela elasticidade (escalabilidade) da infraestrutura.
- **Módulo de Auto-Otimização (MAO):** Esse mecanismo de auto-otimização é acionado quando uma solução automática e otimizada deve ser considerada na reconfiguração da infraestrutura, também considerando requisitos de QoS estabelecidos no SLA.

A Figura 13 exibe um modelo de alto nível com as abstrações dos dois principais mecanismos. Nesse modelo abstrato os elementos são descritos da seguinte forma:

- **Cliente:** Esse elemento representa os clientes do provedor de serviços em nuvem que contratam a infraestrutura como um serviço (IaaS) dos provedores sob demanda e pagam por sua utilização (por hora). Esses clientes assinaram um contrato de nível de serviço

Figura 13 – Visão Abstrata da utilização dos mecanismos em uma arquitetura virtualizada.

(SLA) com o provedor e possuem expectativas de qualidade de serviço (QoS), economia (custos) e opcionalmente algumas personalizações na alocação de seus recursos virtuais.

- **Provedor IaaS:** Esse elemento representa o provedor de serviço em nuvem. Ele fornece serviços de infraestrutura (IaaS) para seus clientes. O provedor deve cumprir com o SLA estabelecido e, para isso, faz uso dos mecanismos propostos em sua arquitetura. Assim, o provedor também deve escolher as configurações/regras de autoconfiguração e auto-otimização, bem como parametrizar tais mecanismos para o gerenciamento autônomo da sua infraestrutura.

Dessa forma, neste capítulo são apresentados os mecanismos de AutoConfiguração e Auto-Otimização oriundos do estudo e desenvolvimento proposto neste projeto de doutorado. Tais mecanismos e seus componentes, bem como as suas funcionalidades são apresentados de forma objetiva buscando manter o foco nas principais contribuições do trabalho. Portanto, na Seção 6.2 é descrito o mecanismo de AutoConfiguração e suas características internas. Na Seção 6.3 é abordado o mecanismo de Auto-Otimização destacando alguns dos seus algoritmos e as suas funcionalidades. Na Seção 6.4 são apresentados algumas ferramentas e os principais componentes desenvolvidos e utilizados em comum pelos dois mecanismos deste projeto. Na Seção 6.5 é apresentada a organização e o funcionamento dos mecanismos propostos juntamente com os componentes anteriormente citados. Por fim, na Seção 6.6 são apresentadas algumas considerações finais deste capítulo.

6.2 Módulo de AutoConfiguração - MAC

Os mecanismos de autoconfiguração são responsáveis pela reconfiguração automática da infraestrutura virtual da arquitetura, em outras palavras, eles devem gerenciar as Máquinas Virtuais (MVs) dos clientes para que se adaptem automaticamente a demanda. A Computação

Elástica pode ser dimensionada tanto horizontalmente (variando o número de máquinas virtuais) quanto verticalmente (variando a quantidade de recursos físicos das MVs, por exemplo, quantidades de CPUs, Memória, Disco, etc.) (CARON *et al.*, 2012). A autoconfiguração remete ao termo "computação elástica" ou computação sob demanda no contexto da computação em nuvem. Para realizar a autoconfiguração é preciso ter conhecimento da atual infraestrutura (monitorar), tomar uma decisão (analisar e planejar) se uma nova configuração deve ser feita. Para isso deve-se basear em algum parâmetro como, por exemplo; acompanhar a demanda, previsão ou políticas pré-estabelecidas. Por fim, uma possível reconfiguração é realizada. Este laço ou ciclo (monitorar, analisar, planejar e executar) deve ser repetido continuamente, sendo conhecido na computação autônoma como "laço de controle" (*loop control*) (KHALID *et al.*, 2009).

6.2.1 Adaptador Elástico

Algumas das ideias e hipóteses elencadas para esse mecanismo foram desenvolvidas e avaliadas no trabalho de mestrado de Ariel da Silva Dias (Processo Fapesp N^o: 2012/10777-0), que associado com este projeto de doutorado, concebeu os mecanismos de autoconfiguração que foram criados utilizando agentes computacionais. Tais agentes foram desenvolvidos com o auxílio do framework JADE¹⁷ e as suas funções são de monitorar a arquitetura e executar a reconfiguração da mesma.

A principal proposta para o mecanismo de AutoConfiguração era a criação de um adaptador elástico responsável pelo gerenciamento auto-adaptativo de recursos da infraestrutura. Assim, para o correto funcionamento deste adaptador foram desenvolvidos alguns outros elementos (DIAS, 2015):

- **Controle de admissão:** trata-se de um módulo que recebe as requisições dos clientes antes de serem processadas. Este módulo analisa se o cliente possui ou não Máquinas Virtuais disponíveis para atender a requisição. Além disso, ele também realiza um controle financeiro, como por exemplo, verificar se o cliente ainda possui créditos (horas/máquina) disponíveis para executar a requisição. Caso o usuário disponha de créditos a requisição é aceita e executada, caso contrário a requisição é negada e um relatório será encaminhado para o administrador da infraestrutura.
- **Agente monitor:** agentes monitores foram desenvolvidos com o objetivo de coletar informações (utilização de CPU, memória, disco, etc.) das Máquinas Virtuais (MV's) constantemente.
- **Agente atuador:** agentes atuadores são responsáveis por alterar configurações (CPU, Memória, Disco) nas Máquinas Virtuais a fim de melhorar o desempenho ou economizar recursos.

¹⁷ <<http://jade.tilab.com/>>

A Figura 14 exibe o adaptador elástico, o controle de admissão e os agentes (monitor e atuador) desenvolvidos em conjunto com o projeto de mestrado (DIAS, 2015).

Figura 14 – Adaptador Elástico e outros elementos para o gerenciamento automático de recursos computacionais virtualizados (DIAS, 2015).

Ainda em conjunto com o projeto de mestrado, a tomada de decisão por uma nova reconfiguração foi dada de acordo com duas abordagens.

6.2.1.1 AutoConfiguração baseada em Monitoração

Na primeira, reconfiguração da infraestrutura, ou seja das máquinas virtuais (MVs), ocorre de acordo com a demanda. Dessa forma, a reconfiguração é guiada pela monitoração dos recursos computacionais das MVs, que podem ser aumentados ou diminuídos gradativamente seguindo uma porcentagem estipulada pelo administrador. Por exemplo, se a demanda aumenta e o agente monitor repassa informações indicando que as MVs do cliente estão sobrecarregadas, o Adaptador Elástico deverá aumentar os recursos computacionais virtuais (CPU, Memória e Disco) desse cliente. Para auxiliar nessa tarefa, a proposta foi a criação de *shell scripts* que executam uma ferramenta de gerenciamento de máquinas virtual denominada *Virsh*¹⁸.

6.2.1.2 AutoConfiguração baseada em Previsão

Na segunda abordagem, a reconfiguração ocorre baseada na previsão da carga que é estipulada para o dia corrente. Para realizar a previsão foi desenvolvido um módulo com duas políticas de previsão de carga baseadas nas técnicas da Média Móvel Exponencial (MME) e no Modelo de Holt-Winter (MHW). A MME é uma técnica que elimina grandes oscilações de valores e, assim, permite filtrar apenas o movimento principal através da utilização da média móvel simples (CORRAR; THEÓPHILO, 2004). Além disso, a MME foi escolhida, pois ela

¹⁸ <<http://libvirt.org/virshcmdref.html>>

dá mais ênfase (peso) aos valores mais recentes e menor aos valores mais antigos que serão esquecidos mais rapidamente (DIAS, 2015). O MHW é destinado ao tratamento de demandas como séries temporais que apresentam tendências e sazonalidade (CORRAR; THEÓPHILO, 2004).

Para o cálculo da Média Móvel Exponencial (MME), Dias (2015) apresentou a seguinte equação (DIAS, 2015):

$$MME_x = ME_{(x-1)} + K_x \{Fech_{(x)} - ME_{(x-1)}\} \quad (6.1)$$

Onde:

MME_x é a média móvel exponencial do dia x ;

$ME_{(x-1)}$ é a média móvel exponencial do dia $x-1$;

K é uma constante onde $K = 2/(N + 1)$, sendo N o número de dias desejados para o cálculo.

$Fech$ é o fechamento atual ou valor atual;

Para a utilização do modelo de Holt-Winter (MHW) foram utilizadas as equações de Makridakis (1998) (DIAS *et al.*, 2014)(MAKRIDAKIS, 1998):

- **Nível:** elemento estacionário (sem tendência e sazonalidade), mas pode possuir alguma variação aleatória;

$$L_t = \alpha \frac{Y_t}{S_t - s} + (1 - \alpha)(L_{t-1} + b_{t-1}) \quad (6.2)$$

- **Tendência:** elemento adicionado ao nível.

$$b_t = \beta(L_t - L_{t-1}) + (1 - \beta)b_{t-1} \quad (6.3)$$

- **Sazonalidade:** elemento adicionado aos elementos de nível e tendência.

$$S_t = \gamma \frac{Y_t}{L_t} + (1 - \gamma)S_{t-s} \quad (6.4)$$

- **Previsão:** elemento resultado do modelo.

$$F_{t+m} = (L_t + b_t m)S_{t-s+m} \quad (6.5)$$

Onde:

L_t : é o elemento nível para o período t ;

b_t : é o elemento tendência para o período t ;

S_t : é elemento sazonalidade para o período t ;

F_{t+m} : é o valor da previsão para o período m ;

Y_t : é o valor observado no período t ;

s : é o intervalo de tempo da sazonalidade;

m : é o número do período previsto;

α : constante para ajustar o *Nível*;

β : constante para ajustar a *Tendência*;

γ : constante para ajustar a *Sazonalidade*.

Desde que: $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$ e $0 \leq \gamma \leq 1$.

Dessa forma, duas políticas diferentes de previsão foram desenvolvidas para tentar prever a demanda dos clientes para o dia seguinte e assim, tentar antecipar a reconfiguração da infraestrutura do provedor para que tente atender adequadamente as demandas.

6.2.1.3 Retorno do Investimento (ROI - Return on Investment)

Além disso, durante todo esse processo, independentemente da abordagem que foi escolhida, é calculado o retorno do investimento ou *Return on Investment* (ROI) que informa ao cliente se o seu investimento inicial (o "aluguel" pago pela infraestrutura virtualizada no início do acordo) foi recuperado (utilizado) adequadamente (DIAS, 2015). É importante ressaltar que para a proposta do ROI torna-se necessário que o cliente faça uma reserva das instâncias (máquinas virtuais) durante um certo período.

No final de cada período é gerado um relatório sobre o quanto de recursos computacionais contratados foi gasto (investido) e o quanto ainda pode ser utilizado pelo cliente. Dessa forma, o cliente pode acompanhar se os recursos contratados (investimento) realmente estão sendo utilizados (retorno). As equações para esses cálculos são as seguintes: (baseadas em (BRAGA, 1989), (GITMAN, 2006), (BRIGHAM; HOUSTON, 1999) (DIAS *et al.*, 2014)):

$$vg = \sum_{pdc}^{cv - (r * mbu)} \quad (6.6)$$

$$rest = \sum_{pdc}^{cv - (\frac{cv}{pdc}) - vg} \quad (6.7)$$

$$rent = rest / cv \quad (6.8)$$

Sendo, cv o valor contratado, r é a tarifa (*rate*) aplicada, mbu é a quantidade calculada de MBs utilizados no período (dia) e pdc é a quantidade de períodos (dias) de duração do contrato. A Equação 6.6 retorna o valor gasto (investido) no período contratado. A Equação 6.7 resulta no quanto ainda pode ser utilizado do valor contratado. Finalmente, a Equação 6.8 resulta na utilização em porcentagem do valor contratado.

Esse trabalho colaborativo concebeu ao mecanismo de Autoconfiguração a possibilidade de gerar resultados que comprovam que a reconfiguração por monitoração é capaz de acompanhar e reconfigurar a infraestrutura de forma mais eficiente que a abordagem de previsão utilizada. Alguns dos resultados desta proposta são apresentados no próximo capítulo. Esse trabalho em conjunto resultou no artigo de título "*Providing IaaS resources automatically through prediction and monitoring approaches*" (DIAS *et al.*, 2014) publicado na conferência: "Nineteenth IEEE Symposium On Computers And Communications (ISCC 2014)".

6.2.2 Ferramenta de Administração de Máquinas Virtuais (FaMaV)

A proposta mencionada anteriormente para a reconfiguração de arquiteturas virtualizadas faz uso de *shell scripts* que executam a ferramenta *Virsh*, isso apresenta algumas desvantagens. Por exemplo, não havia um tratamento de falhas no caso de um problema na comunicação ou execução de uma reconfiguração na máquina virtual, pois identificar e se recuperar de erros durante a execução de um *shell script* não é uma tarefa trivial, pois implicam em meios intermediários onde prever e tratar a ocorrência de falhas tornam-se desafios complicados de serem solucionados.

Além disso, a ferramenta *Virsh* é baseada em linhas de comando (CLI - *Command Line Interface*) o que torna dispendiosa a escrita de *scripts* para a execução de um mesmo "comando" para várias máquinas virtuais. Por exemplo, para ligar dez máquinas virtuais era preciso executar um *script* dez vezes ou criar um novo *script* com um laço de repetição.

Dessa forma, foi proposto e desenvolvido um *framework* visando solucionar tais problemas. Esse *framework* foi nomeado como *Ferramenta de Administração de Máquinas Virtuais* (FAMaV) e trata-se de um conjunto de classes e métodos desenvolvidos na linguagem Java para o gerenciamento facilitado de máquinas virtuais. A escolha pela linguagem Java é justificada devido aos agentes computacionais, o adaptador elástico e o módulo de inferência (Seção 6.4.2) também terem sido desenvolvidos nesta linguagem.

Para a implementação do FAMaV foi proposto um projeto de Iniciação Científica (IC), executado pelo aluno Yuri Toledo Neves (Bolsista Santander), que desenvolveu uma *Application Programming Interface* (API) em Java para o gerenciamento de MVs e também uma ferramenta *Command Line Interface* (CLI) semelhante ao *Virsh*. Portanto, criou-se uma camada de *software* sobre a biblioteca *libvirt* que foi utilizada como base inicial. Contudo a *libvirt* foi desenvolvida em C e a proposta da FAMaV era que também fosse escrita em Java, assim o projeto de IC

também fez uso da (*Java API bindings + Java Native Access (JNA)*¹⁹) que fornece a interação entre a linguagem Java e a *libvirt* desenvolvida em C.

Dessa forma, no projeto de IC criou-se uma API que pode ser utilizada como alternativa a execução de *scripts* e do *Virsh* via *shell*. Assim, por meio da programação com Java, erros e exceções poderiam ser identificados e recuperados programaticamente (utilizando blocos de *try-catch*). A Figura 15 exibe o *framework* FAMaV em uma pilha de camadas abstratas que visa exemplificar melhor a sua aplicação.

Figura 15 – *Framework* FAMaV no contexto de um arquitetura em nuvem. (NEVES *et al.*, 2014).

A Tabela 3 (NEVES *et al.*, 2014) exibe algumas das funções desenvolvidas na FAMaV comparadas as funções equivalentes da *Virsh*. Contudo, o FAMaV possui a vantagem de ser capaz de manipular várias MVs ao mesmo tempo, ou seja, ele é capaz de trabalhar com listas (*List*). Em comum nas duas ferramentas existem funções que alteram a quantidade de memória, migram MVs e retornam informações sobre recursos.

Os resultados de comparação entre o *Virsh* e a nova ferramenta FAMaV (Ferramenta de Administração de Máquinas Virtuais) são apresentados no próximo capítulo. Esses resultados também resultaram na publicação do artigo intitulado: "FAMaV: Análise Comparativa entre Ferramentas de Gerenciamento de Máquinas Virtuais"(NEVES *et al.*, 2014) no "XV Simpósio em Sistemas Computacionais de Alto Desempenho - WSCAD 2014".

O mecanismo de AutoConfiguração pode ser utilizado isoladamente fazendo uso tanto a abordagem de monitoramento quanto da previsão. Porém, a parte de decisão pode ser executada pelo módulo de inferência (Seção 6.4.2) que dispõe de uma base de conhecimento representada por uma ontologia. Nessa base informações de QoS (*Quality of Service*) que compõem o SLA, políticas que envolvem a parte financeira (custo), configurações de otimização de recursos e regras personalizadas podem ser especificadas visando uma decisão mais complexa e elaborada.

¹⁹ <<https://libvirt.org/java.html>>

Tabela 3 – Comparação entre as funções do FAMaV e da Virsh (NEVES *et al.*, 2014).

<i>FAMaV</i>	<i>Virsh</i>	Descrição da Função
-poweron	start	Inicia um domínio através do seu nome.
-poweronlist	–	Inicia uma lista de domínios através de seus nomes.
-poweroff	shutdown	Desliga um domínio através do seu nome.
-poweroffid	shutdown	Desliga um domínio através de seu ID.
-powerofflist	–	Desliga uma lista de domínios através de seus nomes.
-hibernate	suspend	Hiberna um domínio através de seu nome.
-hibernateid	suspend	Hiberna um domínio através de seu ID.
-hibernatelist	–	Hiberna uma lista de domínios através de seus nomes.
-resume	resume	Retoma um domínio através de seu nome.
-restart	reboot	Reinicia o domínio através de seu nome.
-restartlist	–	Reinicia uma lista de domínios através de seus nomes.
-changememory	setmem	Muda a memória de um domínio através de seu nome.
-changememoryid	setmem	Muda a memória de um domínio através de seu ID.
-memoryalloc	dommemstat	Quantidade alocada de memória para o domínio.
-infostatus	dominfo	Informações sobre o domínio.
-maxmemory	memtune	Informa a memória máxima do domínio.
-usecpu	vcpuinfo	Informação de uso da CPU do domínio.
-nodeinfo	nodeinfo	Informações sobre o nó da máquina real.
-diskallocation	–	Informação sobre alocação do disco para o domínio.
-maxdisk	–	Máxima alocação possível de disco.
-listinactives	list –inactive	Lista os domínios inativos.
-listactives	list	Lista os domínio ativos.
-listall	list –all	Lista todos os domínios.
-listnetworks	net-list	Lista as redes.
-migrate	migrate –live	Migra o domínio.

6.3 Módulo de Auto-Otimização - MAO

Prover uma infraestrutura adequada dinamicamente aos seus clientes é um desafio para os provedores de serviços. Cada cliente exige capacidade de recursos e em escalas diferentes visando garantir que as suas aplicações hospedadas na infraestrutura atendam a demanda imposta por seus usuários. Por exemplo, a aplicação de um determinado cliente pode ser direcionada à CPU (*CPU-Bound*) enquanto de outro cliente pode ser uma aplicação que exija mais memória (*Memory-Bound*). Algumas aplicações ou serviços podem ter uma demanda de acessos simultâneos extremamente alta, porém sazonais, enquanto outra aplicação pode ter poucos acessos aleatórios porém constantes.

Dimensionar e prover recursos adequadamente é o desafio dos provedores em Nuvem, que geralmente, disponibilizam recursos virtualizados como, por exemplo, CPU, Memória e Disco, agrupados ou baseados em Instâncias ou Máquinas Virtuais (VMs). Algumas configurações de VMs são pré-definidas pelo provedor em Nuvem (a Amazon trabalha com diferentes classes de VMs²⁰) e o próprio cliente deve determinar a melhor quantidade e capacidade (tipo ou classe) das máquinas virtuais para que o seu sistema funcione adequadamente. Delegar essa tarefa (dimensionamento) para os clientes pode não ser interessante, pois alguns clientes podem não ter experiência sobre como realizar um planejamento de capacidade ou mesmo identificar e analisar cargas de trabalho. Além disso, uma vez que os serviços ou aplicações estão disponibilizados na Web, o fluxo de requisições é imprevisível.

²⁰ <https://aws.amazon.com/ec2/instance-types>

Nesse contexto, a Computação em Nuvem proporciona uma grande vantagem para os provedores, pois eles podem utilizar e fornecer serviços que permitem a elasticidade da capacidade computacional (aumentar ou diminuir recursos virtuais), por meio de configurações de parâmetros e políticas simples de escalonamento fornecendo uma escalabilidade dinâmica aos clientes. Assim, sob uma alta demanda por serviços, os recursos computacionais (CPU, Memória, Disco ou mesmo novas instâncias de MVs) podem ser adicionados automaticamente para atender o aumento de requisições. Por outro lado, em um momento de baixa demanda, recursos computacionais excedentes podem ser desalocados automaticamente para que o cliente tenha economia, pois geralmente ele paga pela utilização dos recursos por hora.

Provedores como *Amazon Elastic Compute Cloud* (EC2) e *Microsoft Azure* utilizam uma metodologia de provisionamento de recursos na qual os clientes são responsáveis por estimar com precisão a quantidade de recursos necessários e selecionar a instância a ser contratada (HUU; MONTAGNAT, 2010).

Porém, uma grande desvantagem é que tais sistemas de escalonamento, fornecidos pelos provedores, consideram basicamente a demanda ou a taxa de utilização dos recursos para realizar o acréscimo ou decréscimo de recursos. Por exemplo, a proposta da Amazon, um dos maiores provedores de serviços em nuvem, para a reconfiguração automática da infraestrutura de seus clientes é baseada em monitoramento, por meio de alertas (*CloudWatch Alarms*) e políticas (*Scaling Policies*). O escalonamento da *Amazon Web Services* (AWS) pode ser feito baseado em métricas que são geralmente baseadas no consumo de CPU (AMAZON, 2015b) ou em políticas que são basicamente divididas em políticas manuais (*Manual Scaling*), políticas dinâmicas (*Dynamic Scaling*) e políticas agendadas (*Scheduled Scaling*) (AMAZON, 2015c).

As políticas dinâmicas são as mais interessantes para este trabalho, a Amazon permite ao usuário configurar o tipo de escalonamento de recursos baseado no ajuste do número de instâncias que devem ser alocadas ou desalocadas que seguem uma quantidade ou uma porcentagem definida pelo usuário (AMAZON, 2015a). Assim, ainda fica a cargo do cliente tentar prever qual seria a melhor configuração para um escalonamento eficiente da sua infraestrutura hospedada na nuvem do provedor. Contudo, nenhuma configuração eficiente é feita automaticamente para que seja garantida a QoS da infraestrutura. Além disso, geralmente a alteração ocorre no número de instâncias respeitando as classes definidas por eles, onde eles adicionam ou removem VMs da mesma classe (escalabilidade horizontal) ou eles modificam VMs de classes diferentes, por exemplo, um cliente contrata uma VM do tipo *Small* e por alguma razão uma nova VM do tipo *Large* é ligada. Dessa forma, é mais fácil para os provedores como a Amazon definir instâncias com configurações fixas ao invés de criar uma instância com configurações específicas para cada cliente. Geralmente os provedores também deixam de fora, a alteração dinâmica e em tempo de execução das configurações dos recursos virtuais dessas máquinas (escalabilidade vertical).

Dessa forma, um dos desafios é quantificar o volume de recursos (quantidade de instâncias de máquinas virtuais) que devem ser iniciados ou desligados. Portanto, realizar tal reconfiguração

da infraestrutura dinamicamente, não apenas com o objetivo de atender a demanda, mas realizar tal procedimento de forma eficiente e almejando outros objetivos como redução dos custos e a garantia da qualidade de serviço definida no SLA estabelecido entre provedor e clientes são tarefas não triviais e que necessitam de maiores estudos. Outro desafio destacado é acertar o limiar de quando uma reconfiguração da infraestrutura deve ser realizada, também trata-se de um tópico de grande interesse tanto no âmbito acadêmico quanto industrial.

Há diversos problemas complexos dentro do contexto de computação em nuvem que são abordados por soluções que utilizam otimização. Por exemplo, os problemas de alocação de máquinas virtuais em máquinas reais (MASDARI; NABAVI; AHMADI, 2016), economia de energia (BELOGLAZOV; ABAWAJY; BUYYA, 2012b), escalonamento e balanceamento de carga das aplicações em máquinas virtuais (L.D.; KRISHNA, 2013) e utilização de recursos visando minimizar o custo e ainda garantir o desempenho (BYUN *et al.*, 2011), a QoS e o cumprimento do SLA. Todos esses problemas ainda são desafios que podem ser explorados por pesquisadores e exigem soluções ótimas ou sub-ótimas a serem alcançadas por técnicas conceituadas dentro da área de otimização, desde que sejam adequadamente mapeadas para solucionar tais problemas práticos que surgiram com o paradigma de computação nas “nuvens”.

Dessa forma, a proposta do mecanismo de auto-otimização visa solucionar o problema de alocação de recursos de forma adequada, para isso foi utilizando uma heurística e uma meta-heurística para alcançar resultados que garantam o SLA, o auto-gerenciamento e a QoS para os clientes de um provedor em nuvem. Assim, foram desenvolvidos dois algoritmos, sendo um determinístico e outro um micro-Algoritmo Genético (μ AG) para o módulo MAO. Cada algoritmo recebe como entrada a configuração da infraestrutura atual, como está o comportamento da infraestrutura atual (valores de QoS atuais, que podem estar armazenados em banco de dados ou na ontologia), qual seria o SLA/QoS adequado e as possíveis restrições impostas pelo cliente e provedor (esses dois últimos conjuntos de dados também podem ser recuperados da ontologia). Com base nessas informações os algoritmos buscam determinar a melhor configuração de infraestrutura que pode ser passada para o Adaptador Elástico para modificar a atual configuração da arquitetura por uma mais otimizada.

É relevante ressaltar que esse mecanismo de auto-otimização somente deverá ser utilizado quando o cliente e provedor concordarem em configurar e executar otimizações, essa configuração está prevista na opção do usuário/provedor na ontologia (Seção 6.4.2).

6.3.1 Metodologia para o cumprimento dos SLAs

Para o melhor entendimento do funcionamento dos algoritmos de otimização é preciso inicialmente explicar a metodologia que foi utilizada para solucionar o problema da identificação e do cumprimento dos SLAs respeitando os parâmetros/atributos de QoS. O principal objetivo dos algoritmos é que a nova proposta de infraestrutura atenda ao SLA. Por esse motivo, a ideia apresentada neste trabalho é que o SLA seja tratado, a princípio, como sendo o fornecimento da

infraestrutura como um serviço (IaaS) com qualidade e para isso deve satisfazer as metas em quatro atributos de QoS: Capacidade, Tempo de Resposta, Disponibilidade e Custo.

Dessa forma, a metodologia proposta é um processo semelhante à composição de serviços baseada em QoS empregada em SOA, onde a QoS da infraestrutura total do cliente é o resultado da composição dos atributos de QoS de todas as MVs (“serviços”) desta mesma infraestrutura. Contudo, esta proposta difere da composição de serviços baseada em QoS tradicional, pois nela é possível escolher uma quantidade variável de MVs candidatas para compor a infraestrutura e atender o SLA definido.

Para solucionar este problema de otimização, serão consideradas as propostas por Ko, Kim e Kwon (2008) que foram adaptadas para solucionar as questões de agregação de atributos e normalização da QoS respectivamente. Equações de agregações são necessárias para calcular o valor total de cada atributo de QoS de todas as MVs da infraestrutura.

A Equação 6.9 de agregação foi elaborada para a capacidade (NAKAMURA *et al.*, 2017).

- **Capacidade:** é o número de máquinas virtuais contratadas pelo cliente. A princípio há três tipos de Virtual Machines (VMs), sendo elas, *Small*, *Medium* e *Large*, tais instâncias foram baseadas nas configurações das instâncias *m3.medium*, *m3.large* e *m3.xlarge* da Amazon EC2²¹. Uma infraestrutura de um *datacenter* pode ter inúmeras VMs com diversas configurações diferentes. Para obter a capacidade foi elaborada a equação (6.9), na qual n corresponde ao número de máquinas virtuais (NAKAMURA *et al.*, 2017):

$$\sum_{i=1}^{i=n} Capacidade(VM_i) \quad (6.9)$$

As três equações de agregação seguintes foram adaptadas para este projeto de doutorado (KO; KIM; KWON, 2008) (NAKAMURA *et al.*, 2017):

- **Tempo de Resposta Médio:** é o tempo de resposta médio da aplicação que é obtido pela execução da mesma na infraestrutura contratada. O tempo de resposta pode ser obtido pela equação (6.10) (KO; KIM; KWON, 2008), na qual n indica o número de máquinas virtuais:

$$\frac{\sum_{i=1}^{i=n} TempodeRespostaMdio(VM_i)}{n} \quad (6.10)$$

- **Disponibilidade:** representa a disponibilidade da infraestrutura em presente e pronta para a execução da aplicação. Assim, o cálculo da disponibilidade pode ter duas abordagens; (1) É feito o cálculo da razão entre o número de máquinas e a probabilidade de erro/falta, ou

²¹ <https://aws.amazon.com/pt/ec2/instance-types/>

seja, 1 sobre o número de VMs disponíveis, nesse caso se há quatro VMs, a probabilidade de ocorrer erro ou falta seria de 1/4 (25%) e a disponibilidade então seria de 75%; ou (2) utiliza-se contadores para armazenarem a quantidade de requisições atendidas com sucesso e que falharam, calcula-se então o percentual de requisições atendidas com êxito, essa porcentagem será o percentual de disponibilidade da infraestrutura. Pode-se calcular a disponibilidade primeiramente pela abordagem (1) e após um período de monitoramento pode-se alternar para a abordagem (2). Na segunda abordagem, a disponibilidade pode ser calculada pela equação (6.11) (KO; KIM; KWON, 2008), na qual n corresponde ao número de máquinas virtuais:

$$\prod_{i=1}^n Disponibilidade(VM_i) \quad (6.11)$$

- **Custo por hora (Custo/h):** é o custo por hora correspondente a um valor monetário que deve estar estipulado no SLA, referente a quanto o cliente pagará por hora pela utilização da VM. O valor desse custo monetário por hora é resultante da equação (6.12) (KO; KIM; KWON, 2008), na qual n corresponde ao número de máquinas virtuais:

$$\sum_{i=1}^n Custo(VM_i) \quad (6.12)$$

Para a execução dos algoritmos é inicialmente preciso estipular um SLA para cada cliente, tendo como atributos de QoS: a Capacidade (C^c), o Tempo de Resposta (T^c), a Disponibilidade (A^c) e o Custo por hora (C/h^c) da sua infraestrutura. Para obter os valores de QoS da infraestrutura como um todo, são utilizadas as equações de agregação mencionadas anteriormente. Inicialmente, os valores de tempo de resposta, a disponibilidade e o custo/h podem ser obtidos por simulações no CloudSim²². No caso dos valores retornados pela simulação da arquitetura do cliente não forem compatíveis com os descritos no SLA, então é aplicada uma meta-heurística para encontrar uma solução (capacidade (C^*), tempo (T^*), disponibilidade (A^*) de custo/h (C/h^*)) que melhor se adeque às exigências do cliente. Os algoritmos podem ficar em um laço contínuo e os próximos valores de QoS podem ser dados reais coletados pelo agente monitor (tempo de resposta e disponibilidade) e armazenados na ontologia (Capacidade, Custo por hora) (NAKAMURA *et al.*, 2017).

6.3.2 Algoritmos de Otimização

O desenvolvimento dos algoritmos está sendo executado pelo aluno de Mestrado Leonildo José de Melo de Azevedo (Processo Fapesp Nº: 2015/11623-4). Neste outro trabalho colaborativo, os dois algoritmos propostos foram implementados: um algoritmo determinístico (vide Algoritmo 1) e um μ AG (vide Algoritmo 2). Esses algoritmos otimizam o número de MVs contratadas pelo

²² <http://www.cloudbus.org/cloudsim/>

cliente e seu resultado é uma proposta de nova configuração para a infraestrutura, definida por (s, m, l) , que são respectivamente os números de cada tipo (*Small*, *Medium* e *Large*) de MVs que irão satisfazer as necessidades do cliente. Por exemplo, se um algoritmo retornar $(2, 3, 4)$, então a nova configuração que irá satisfazer o SLA deverá ser composta por duas instâncias de MVs *Small*, três instâncias de MVs *Medium* e quatro instâncias de MVs *Large* (NAKAMURA *et al.*, 2017).

Contudo, se um SLA for mal estipulado será complicado satisfazer todos os requerimentos sem conflitos. Por exemplo, no caso da Capacidade (C^c) da infraestrutura for superdimensionada poderá entrar em conflito com o Custo (C/h^c) que o cliente deseja pagar, ou subdimensionada poderá entrar em conflito com o Tempo de Resposta (T^c) desejado. Dessa forma, nesse caso a Capacidade (C^*) é um atributo que está relacionado com os demais, mas que, nestes casos de conflito, o seu valor poderia ou deveria ser alterado dinamicamente, baseado a resposta dos algoritmos que irão tentar definir o melhor arranjo de MVs (s^*, m^*, l^*) (NAKAMURA *et al.*, 2017).

Como dito anteriormente, a solução (s, m, l) (nova configuração) é resultado da execução dos algoritmos. Uma vez aplicada a Capacidade (C^*), o Tempo de Resposta (T^*), a Disponibilidade (A^*) e o Custo/hora (C/h^*) devem ser novamente mensurados. Se seus valores não forem compatíveis com aqueles estipulados no SLA ($C^c, T^c, A^c, C/h^c$), outra representação de solução (s', m', l') deve ser avaliada. Esta compatibilidade é estimada pela Equação (6.13) (AZEVEDO, 2016) (NAKAMURA *et al.*, 2017).

$$f(P[i]) = \left| \frac{C^c - C^*}{C^*} \right| + \left| \frac{T^c - T^*}{T^*} \right| + \left| \frac{A^c - A^*}{A^*} \right| + \left| \frac{C/h^c - C/h^*}{C/h^*} \right| \quad (6.13)$$

O cálculo estatístico da Distância de Manhattan (BLACK, 2006) é empregado para estimar quão próxima a solução está aos valores estipulados no SLA. Como há quatro objetivos com escalas de valores diferentes, então é necessária uma normalização utilizando um cálculo de divisão para redução do *Gap*. A cada execução é armazenada a melhor solução e no método determinístico são geradas todas as possibilidades.

O espaço de busca é ajustado pelo cliente que define o número de máquinas, permitindo uma variação de 50% para cima ou para baixo do número MVs desejadas. Por exemplo, se o cliente contratou 50 MVs, o intervalo possível para explorar estará entre 25 e 75 MVs. Neste caso, a representação de soluções como $(s, m, l) = (1, 20, 4)$ ou $(s, m, l) = (0, 0, 75)$ são permitidas desde que $25 \leq (s + m + l) \leq 75$. Assim, todas as combinações possíveis de MVs dentro dessa faixa de 50% são avaliadas com o objetivo de obter o melhor valor para a Equação (6.13) (NAKAMURA *et al.*, 2017).

O pseudo-código em Algoritmo 1 descreve o algoritmo determinístico, no qual todas as combinações possíveis para (s, m, l) são avaliadas exaustivamente e a melhor é retornada.

O pseudo-código em Algoritmo 2 exhibe a proposta do μ AG. Este método é uma versão

Algoritmo 1: Algoritmo Determinístico

Input: SLA: $C^c, T^c, A^c, C/h^c$

```

1 //2. Varra todo o espaço de pesquisa
2 repeat
3   //Gerar uma capacidade de avaliação
4   Gerar( $s, m, l$ )
5   Avaliar( $s, m, l$ )
6 until Até que todas as possibilidades ( $s, m, l$ ) dentro do intervalo tenham sido testadas;
7 //Retornar uma configuração melhor encontrada para satisfazer o SLA do cliente
8 return  $SLA^* : C^*, T^*, A^*, C/h^*$ 

```

de Algoritmo Genético (GA) que opera com uma população de menor porte e emprega um critério de convergência que permite a reinicialização (EIBEN; SMITH, 2003).

Algoritmo 2: Algoritmo μ AG

Input: SLA: $C^c, T^c, A^c, C/h^c$

```

1 //1. Gerar uma população com 5 indivíduos
2 InicializarPopulação(P)
3 //Avaliar o fitness de cada indivíduo na população
4 Avaliar(P)
5 repeat
6   Selection(P)
7   //Seleção Crossover(P)
8   //Cruzamento Mutation(P)
9   //Mutaço Evaluate(P)
10  //Avaliar if O melhor indivduo no  atualizado aps 10 iteraes then
11    Reinitialize(P)
12 until At que o limite de tempo tenha sido atingido;
13 //retornar o melhor indivduo
14 return  $SLA^* : C^*, T^*, A^*, C/h^*$ 

```

No algoritmo μ AG a populao foi ajustada at que se estabeleceu em apenas 5 indivduos, cada um dos quais representa uma possvel configurao MV (s, m, l). A inicializao gera de forma aleatria os indivduos (s, m, l) tais como $\min \leq (s + m + l) \leq \max$, onde o intervalo possvel definido por $[\min, \max]$. Um torneio aplicado para selecionar entre dois indivduos qual ir para participar do *crossover*. O BIX- α crossover (EIBEN; SMITH, 2007) aplicado para gerar novos indivduos. No entanto, se ocorrer $(s + m + l) \leq \min$ ou $\max \leq (s + m + l)$ para o novo indivduo, a quantidade necessria adicionada ou deduzida da ltima posio (l). Se no for suficiente, a quantidade tambm pode ser reduzida da posio m .

Alguns testes computacionais realizados com o μ AG conduziram para a incluso de um operador de mutao com uma taxa de 20%. Esse valor de 20% foi escolhido depois da realizao de testes realizados para o refinamento (*tuning*) do algoritmo - vide Anexo A). O operador de mutao reinicializa completamente todos os valores de um indivduo. O critrio de convergncia

adotado foi baseado no pressuposto de que o melhor indivíduo não foi atualizado por mais de 10 iterações. Quando o critério de convergência é satisfeito, a população é reinicializada novamente e apenas o melhor indivíduo é mantido. O limite de tempo é definido como critério de parada (AZEVEDO, 2016).

Os resultados das avaliações dos algoritmos são apresentados no próximo Capítulo demonstrando que o provisionamento de recursos pode ser realizado dinamicamente baseado nas exigências feitas pelos clientes em seus SLAs e na demanda de serviços.

6.4 Componentes Comuns aos Módulos

6.4.1 Ferramenta de balanceamento de carga

Uma ferramenta para o controle do balanceamento de carga é essencial para o correto funcionamento dos serviços prestados por um provedor de serviço em nuvem. A divisão e o balanceamento da carga de trabalho entre as diversas instâncias de MVs dos clientes é uma tarefa do provedor de serviço e não faz parte do escopo deste projeto.

Em Centurion (2015) foram discutidas algumas características dos tipos de carga de trabalho em nuvem, em especial como as rajadas (*burst*) de requisições afetam o desempenho dos serviços em nuvem (CENTURION, 2015). Contudo, para auxiliar os mecanismos desenvolvidos foi utilizado neste projeto o HAProxy²³ uma ferramenta livre destinada ao balanceamento de carga em aplicativos baseados em TCP e HTTP.

O HAProxy disponibiliza nove algoritmos de balanceamento de carga, porém por uma questão de tornar a análise dos resultados mais simples o algoritmo *Round-Robin*, que escolhe um servidor diferente por vez, foi escolhido por ser o mais utilizado e destinado as conexões curtas.

6.4.2 Módulo de Inferência (ModInf)

O Módulo de Inferência (ModInf) é composto pela ontologia, por regras e o sistema de decisão. Ele pode ser utilizado pelos MAC e MAO para obter dados dos clientes, provedores, dos SLAs, da arquitetura virtualizada do provedor e para decidir o que deve ser feito com a infraestrutura do cliente.

6.4.2.1 A Ontologia IaasOnt

Para o desenvolvimento da ontologia houve uma pesquisa sobre ontologias propostas para computação em nuvem e mais especificamente para IaaS. O trabalho de Dukaric e Juric (2013) propõe uma taxonomia e arquitetura para IaaS, parte dessa taxonomia foi considerada

²³ <http://www.haproxy.org/>

inicialmente na criação da ontologia (DUKARIC; JURIC, 2013). Em Zhang et. al. (2012) um sistema baseado em ontologia OWL foi criado para a descoberta de serviços em nuvens IaaS (ZHANG *et al.*, 2012).

A ontologia CoCoOn (Cloud Computing Ontology) (ZHANG *et al.*, 2012) define o modelo da camada IaaS e descreve os seus serviços. A descoberta pode ser baseada na funcionalidade e em atributos de QoS de serviços IaaS (processamento, armazenamento e rede). A ontologia CoCoOn reúne informações de provedores de serviços (Amazon, Microsoft Azure, GoGrid, etc.) para que um mecanismo de descoberta de serviços denominado *Cloud Recommender* consiga estimar custos, estabelecer relações e possíveis comparações na escolha de serviços de diversos provedores (ZHANG *et al.*, 2012). A CoCoOn (<http://w3c.org.au/cocoon.owl>) foi estudada, porém pouco influenciou na criação da ontologia deste projeto, que já estava iniciada e foi denominada IaaSOnt (*IaaS Ontology*). Os objetivos dessas ontologias são bem diferentes, enquanto a CoCoOn é utilizada para a seleção de serviço em múltiplos provedores, a IaaSOnt é utilizada para o gerenciamento da infraestrutura.

A metodologia para o desenvolvimento da ontologia considera os procedimentos propostos em duas metodologias: *Methontology* (FERNÁNDEZ; GÓMEZ-PÉREZ; JURISTO, 1997) e Metodologia de Noy e McGuinness (NOY; MCGUINNESS, 2001). É possível notar algumas semelhanças nas etapas de desenvolvimento das duas metodologias, conforme exhibe a Tabela 4:

Tabela 4 – Etapas das Metodologias utilizadas como base para a criação da IaaSOnt (NOY; MCGUINNESS, 2001) (FERNÁNDEZ; GÓMEZ-PÉREZ; JURISTO, 1997)

Etapa	Noy e McGuinness	Methontology
1	Determinar o domínio e escopo da ontologia	Aquisição de Conhecimento
2	Considerar o reuso de ontologias existentes	Especificação
3	Enumerar os termos importantes na ontologia	Conceituação
4	Definir as classes e a hierarquia de classe	Integração
5	Definir as propriedades das classes	Implementação
6	Definir as facetas para as propriedades	Avaliação
7	Criar Instâncias	Documentação

Algumas etapas são equivalentes, por exemplo, a etapa 1 da metodologia de Noy e McGuinness sintetiza as fases 1 e 2 da *Methontology* que abordam a aquisição do conhecimento, na qual o ontologista deve determinar o domínio e escopo da ontologia. A etapa 2 é correspondente à fase 4 da *Methontology* na qual deve-se analisar a possibilidade de reutilização e integração de ontologias já existentes. As etapas restantes podem se comparadas às fases 3 e 5 da *Methontology* que abordam basicamente a parte de implementação da ontologia. Apesar da Metodologia de Noy e McGuinness não ter fases específicas para as etapas 6 e 7 da *Methontology*, no decorrer do documento que explica a metodologia, as atividades de avaliação e documentação são comentadas. Por outro lado, o documento da metodologia de Noy e McGuinness é rico em sugestões e exemplos práticos, guiando o desenvolvedor na criação de sua primeira ontologia (NAKAMURA, 2012).

As documentações que foram geradas pela utilização das metodologias neste projeto de doutorado estão disponíveis no Anexo B desta tese.

Para a criação da ontologia IaaSOnt foi utilizado um editor de arquivos de ontologias denominado Protégé²⁴ que trata-se de ferramenta visual criada pelo Centro de Pesquisas para Informática Biomédica da Universidade de Stanford (GENNARI *et al.*, 2002).

Na IaaSOnt foram criadas todas as classes, subclasses, propriedades e relacionamentos necessários para o atendimento do escopo deste projeto. A Figura 16 exibe algumas das classes que compõe a IaaSOnt, sendo que algumas dessas classes merecem uma explicação mais detalhada:

- **MachineCapacity**: Esta classe define a capacidade computacional das máquinas (em especial as virtuais). As suas subclasses (*ExtraLarge*, *Large*, *Medium* e *Small*) serão utilizadas como configurações fixas, sendo semelhante aos tipos de MVs com configurações pré-definidas que alguns provedores oferecem (também conhecido como *flavors*). A classe **VirtualMachine** (subclasse de **Machine**) deve possuir um relacionamento com uma subclasse de **MachineCapacity** que por sua vez deve ter relacionamentos com subclasses da classe **Resources**.
- **Resources**: Esta classe representa os recursos computacionais e cada subclasse define um recurso computacional em particular (Computação, Memória, Rede, etc). Tais subclasses podem incluir outras subclasses, por exemplo, a subclasse **Compute** (Computação) possui atualmente duas subclasses: CPU e GPU. Cada uma das subclasses de **Resources** deve possuir um relacionamento do tipo *Datatype* que indica a quantidade de cada recurso (até este momento considera-se esta informação como intuitiva, por exemplo para a subclasse **Memory** há um relacionamento com o tipo de dado *Integer* que indicará a quantidade de memória em Megabytes. Se houver necessidade de representar tal informação de forma mais explícita, pode-se utilizar a ideia utilizada na CoCoOn que possui as classes *UnitOfMeasurementValue*).
- **Management**: Especifica os tipos de gerenciamento que podem ser fornecidos pelo provedor (**Provider**) de serviços ao seu cliente (**Client**). Basicamente, esta classe serve para o provedor definir qual tipo de gerenciamento deve ser utilizado para um determinado cliente, tornando este gerenciamento mais personalizado. Cada tipo de gerenciamento (subclasse de **Management**) também deve estar relacionado há uma subclasse de política (**Policy**).
- **Policy**: Dependendo do tipo de gerenciamento há diversas opções de políticas que podem ser escolhidas (Todas as ideias de políticas propostas por este projeto de doutorado são

²⁴ <<http://protege.stanford.edu/>>

Figura 16 – Hierarquia de Classes da IaaSOnt.

apresentadas mais adiante). As políticas definem instruções de como os mecanismos devem se comportar (qual decisão tomar).

- **QualityOfService e QoSAttribute:** A classe **QualityOfService** cria um **link** entre a política de SLA e a classe de **QoSAttribute**. Esta última possui subclasses (**Availability**, **Capacity**, **Cost** e **ResponseTime**) que definem os atributos de QoS selecionados para o gerenciamento na IaaSOnt. Segundo Yuanjun et. al. (2013), as propriedades não-funcionais

de serviços em nuvem, consideram não mais do que o custo (*Cost*), tempo (*Time*) e fatores de confiabilidade (*Reliability*), pois é difícil descrever vários serviços em nuvem utilizando uma variada classificação de atributos de uma forma unificada (LAILI *et al.*, 2013). Diferentemente desses autores, foi decidido alterar o atributo confiabilidade para disponibilidade, o motivo para isso é que a tarefa para se determinar o quanto um serviço é confiável depende de diversos fatores que fogem do escopo deste trabalho, pois podem envolver outros conceitos como: segurança e tolerância a falhas (ou faltas). Além disso, dentro do contexto de IaaS, onde o serviço fornecido é a infraestrutura (no escopo deste projeto: a infraestrutura virtual é representada por máquinas virtuais). Portanto, assume-se que as imagens de disco, arquivos de configurações e *hypervisors* estão suficientemente seguros e desempenham suas funções adequadamente, cabendo o gerenciamento garantir, quando necessário, que a máquina virtual do cliente esteja disponível. Outro atributo de QoS na IaaS, não citado pelos autores anteriores, é a Capacidade (**Capacity**), trata-se de uma classe que possui uma propriedade (*dataproperty*) que informa um *index* de desempenho (valor decimal) representado a capacidade computacional.

A classe **QualityOfService** também está relacionada com a subclasse **VirtualMachine** que representa uma máquina virtual, ou seja, cada MV possui a sua QoS, o que faz sentido uma vez que a MV é fornecida ao cliente como serviço dentro do contexto de IaaS. Assim, cada MV possui atributos que indicam a sua disponibilidade, o seu tempo de resposta, o seu custo por hora de utilização e também o valor do *index* de sua capacidade. Neste ponto, é importante não confundir o Atributo de QoS **Capacity** com a classe **MachineCapacity**. A **MachineCapacity** define qual é o tipo (configuração) da máquina, já a **Capacity** é um valor decimal (*index*) que expressa a sua capacidade computacional, portanto embora estejam relacionados, conforme exibe um exemplo na Tabela 5, tais classes têm diferentes significados. É importante salientar que esses valores não são fixos e podem ser configurados conforme decisão do provedor.

Tabela 5 – Relação Atributo **MachineCapacity** e **Capacity**

MachineCapacity	QoS Capacity (hasValue)	Configuração (Exemplo)
<i>Small</i>	0,25	1 CPU, 512MB RAM, 8G Disk, Linux
<i>Medium</i>	0,50	1 CPU, 1GB RAM, 8G Disk, Linux
<i>Large</i>	0,75	2 CPU 1GB RAM, 8G Disk, Linux
<i>XLarge</i>	1,00	2 CPU 2GB RAM, 10G Disk, Linux

Dessa forma, algoritmos podem realizar o gerenciamento de SLA, por meio da verificação da QoS acordada (aquela relacionada e especificada pelo cliente na política de SLA) em comparação com a combinação das QoS da MVs. Mas, para que isso seja possível, considera-se a infraestrutura como sendo uma composição de vários serviços (ou máquinas virtuais), conforme explanado na seção 6.3.1.

Outras classes que merecem destaque são os gerenciamentos (**Management**) e as políticas (**Policy**) (Figuras 17 e 18 respectivamente). Neste projeto de doutorado foram idealizados quatros tipos possíveis de propostas para a gestão de Energia, Financeira, Recursos e SLA/QoS.

Figura 17 – Tipos de Gerenciamento propostos neste projeto (**Management**).

Figura 18 – Tipos de Políticas propostos neste projeto (**Policy**).

Os clientes e consequentemente os provedores devem escolher e fornecer os tipos de gerenciamentos respectivamente. Dessa forma, essas classes simplesmente informam qual(is) gerenciamento(s) o cliente requer para a sua infraestrutura. Cada gerenciamento (**Management**) está relacionado a uma política (**Policy**) de gerenciamento. Na proposta deste trabalho foram criadas as seguintes políticas:

- **SLAPolicy**: Trata-se da política de gerenciamento de SLA/QoS que busca garantir o cumprimento do SLA.
- **ScalingPolicy**: Esta é uma política que define como os mecanismos devem escalar a infraestrutura. De forma Horizontal (variando o número de VMs) ou Vertical (variando os recursos).
- **OptimizationPolicy**: Este é um tipo de política que pode ser utilizado por qualquer um dos quatros tipos de gerenciamento. Ela consiste na utilização de algoritmos de otimização na tentativa de otimizar algum tipo de gerenciamento. Essa política envolve os mecanismos de auto-otimização que são mencionados na seção 6.3.1.

- **PowerPolicy:** Esta política é de maior interesse dos provedores de serviços que buscam diminuir os gastos com o consumo de energia e ao mesmo tempo destacar-se por apresentar uma Computação Verde (*Green Computing*). Como o foco no gerenciamento energético eficiente não é a proposta deste trabalho, a ideia para esse tema pode ser tratada apenas de um problema de otimização para a alocação de MVs em máquinas reais (MRs) que poderá ser resolvido com a criação de um novo algoritmo de otimização para essa finalidade.
- **FinancialPolicy:** São políticas (suas subclasses) de gerenciamento financeiro/econômico deste projeto e fornece ao cliente a possibilidade de escolher entre vários tipos de controles financeiros.

Ainda dentro do contexto da política **FinancialPolicy** foi possível propor diversas outras políticas (subclasses) para o controle financeiro. Segundo Yuan et. al. (2011), há basicamente três métodos de cobrança por instâncias de máquinas virtuais; pacotes de cobranças por ano, mês e hora (YUAN et al., 2011). Uma grande parte dos provedores de serviços fornece ao menos um desses tipos de cobranças. Por exemplo, a SoftLayer (adquirida pela IBM) realiza a cobrança de servidores virtualizados por hora ou mensalmente (SOFTLAYER, 2016) e a Amazon também realiza a cobrança de suas instâncias de máquinas virtuais por hora (*on-demand*). No entanto, podem haver outras variações e combinações de cobranças, por exemplo, a própria Amazon também oferece a cobrança por instâncias reservadas, na qual o usuário paga um valor inicial pelo período de 1 ou 3 anos pela instância reservada e recebe um desconto pela hora utilizada dessa instância. Isso funciona como se fosse um tipo de desconto pela reserva, mas somente valerá a pena se o usuário utilizar uma quantidade de horas suficientes que compensem o valor inicial, pois provavelmente esse valor não será devolvido, mesmo que o usuário não utilize a instância reservada. Além disso, a Amazon também fornece descontos para aqueles que gastam grandes quantias de dinheiro contratando grandes quantidades de instâncias reservadas. Um último exemplo de cobrança, também feita pela Amazon, é a cobrança por instâncias *spot*. Nesse caso a Amazon permite que o usuário negocie e ofereça um valor pela hora de utilização de uma instância *spot*. Esse tipo de instância faz parte da capacidade não usada do Amazon EC2 e o seu preço (preço *spot*) é estipulado pela própria Amazon, mas o seu valor oscila periodicamente de acordo com a relação entre oferta e demanda. Quando o valor ofertado pelo usuário exceder o preço *spot* as instâncias do usuário serão executadas até que ele as finalize ou até o preço *spot* exceder o preço ofertado (o que acontecer primeiro) (AMAZON-EC2, 2015a) (AMAZON-EC2, 2015b).

Com base nesses exemplos, nota-se que há diversas formas do cliente ser cobrado, a escolha pela forma de cobrança adequada é um assunto delicado, pois dependendo do tamanho da infraestrutura contratada, uma má escolha poderá acarretar em enormes perdas financeiras. Muitos fatores podem ser considerados como, por exemplo, os valores dos preços, os descontos oferecidos e as necessidades dos clientes e suas aplicações. Conscientes dessas necessidades, diversas pesquisas propõem modelos de custos, modelos de capacidade e modelos de previsões de

gastos. Contudo, independente da forma como o cliente é cobrado/taxado, surge a necessidade da criação de um gerenciamento financeiro, especialmente para clientes de pequeno e médio porte que desejam utilizar os serviços do provedor, mas não podem comprometer o seu orçamento. Da mesma forma, isso também pode ser do interesse do provedor, pois ele busca vender o seu serviço, mas não deseja que seu cliente extrapole e não consiga cumprir com o pagamento. Dentro deste contexto, as políticas (subclasses) de gerenciamento financeiro (**FinancialPolicy**) foram idealizadas visando solucionar tais problemas. Há três grupos de políticas financeiras:

1. **ROIPolicy**: Essa política é baseada no conceito do retorno do investimento (ROI). Resumidamente, ela é útil no caso do cliente que deseja realizar um investimento inicial, contratando recursos virtualizados por um período determinado, e que gostaria de verificar se esse investimento inicial está sendo utilizado de forma consciente no decorrer deste período.
2. **DefinedCostPolicy**: Engloba duas outras políticas de custo definido. Ao iniciar um período que seja no mínimo mensal, o cliente informa o valor inicial que pretende gastar com IaaS no período. Além disso, ele deve optar por duas funções (sub políticas): A primeira é denominada **LimitedDailyPolicy**, na qual o cliente deve informar um valor fixo de utilização para cada dia do período (valor diário). Por exemplo, se o cliente informar o valor inicial 1200 Unidades monetárias (UMs), o valor diário sendo 120 (UMs), ele sempre iniciará sua infraestrutura com 10 MVs se o valor da hora/máquina for de 1(UM), então a sua infraestrutura funcionará nas primeiras 12 horas de cada dia, durante 10 dias e finalmente será ela interrompida caso ele não aumente o valor inicial.

A segunda é denominada **ScheduledPolicy**, nesta o cliente deve dividir o valor inicial durante os dias do período. Por exemplo, se o cliente informar um valor inicial de 240(UMs), o valor da hora/máquina for de 1(UM), ele possuir apenas uma MV, então o cliente terá 10 dias de utilização, ele pode optar pelos os primeiros dez dias do mês ou pelos os últimos dez dias ou ainda dividir esse valor inicial em pequenas quantidades iguais e deixar a sua MV ligada somente 8 horas por dia durante um mês. A **LimitedDailyPolicy** e a **ScheduledPolicy** oferecem vantagens antagônicas, enquanto a primeira permite um controle mais simples, porém menos flexível, a segunda é justamente o contrário. A primeira é indicada para o cliente que utiliza a infraestrutura diariamente, mas quer limitar seus gastos, a segunda é para cliente que não usa a infraestrutura frequentemente, mas sabe que oportunamente irá utilizá-la, por exemplo, para a execução de algum teste ou processar uma folha de pagamento.

3. **OnDemandPolicy**: Essa política também possui duas outras políticas ou funções de gerenciamento financeiro. Nelas o cliente também define um valor inicial e ambas são baseadas no conceito *on-demand*, permitindo que o gasto siga conforme a demanda, porém há algumas diferenças sutis. A primeira sub política é a **MaximumValuePolicy**, para

explicar o seu funcionamento pode-se fazer uma analogia ao sistema de serviços pré-pagos, ou seja, os gastos com infraestrutura ocorrem sem controle (diário ou agendado), sendo livre para atender a demanda, porém há uma exceção, quando o valor do gasto ultrapassar o valor inicial estipulado pelo cliente toda a infraestrutura será interrompida até que o cliente "insira mais créditos" para continuar a utilizar o serviço. A segunda é a **AssistedCostPolicy**, essa sub política é semelhante ao sistema de *Billing Alert* da Amazon, ou seja, da mesma forma que a primeira o gasto é livre para atender a demanda, porém quando esse valor ultrapassa o valor inicial definido, uma mensagem de alerta é enviada ao cliente. Contudo, o gasto com infraestrutura não é interrompido e, dessa forma, carece de uma assistência do cliente para que o controle financeiro não extrapole o seu orçamento.

Algumas políticas e como seriam os seus funcionamentos já estão conceitualmente definidos. Porém, algumas delas (**OptimizationPolicy** e **PowerPolicy**) ainda precisam ser melhor definidas para ficarem de acordo com as necessidades dos mecanismos de auto-otimização. A proposta também é que as subclasses de **FinancialPolicy** sejam associadas com algoritmos (**Algorithm**) que visam melhorar o controle financeiro para cada cliente. Por exemplo, a sua subclasse **ROIPolicy** pode empregar o algoritmo de cálculo do ROI apresentado na seção 6.2.1.3. Além disso, apesar das políticas já estarem representadas na ontologia é preciso a criação das regras (*rules*) para o processo de inferência da ontologia, a criação de algoritmos e uma forma para a inclusão de informações (parâmetros) que poderão ser inseridas diretamente na ontologia pelo Protégé ou por meio de uma interface Web a ser desenvolvimento que permita a interação do cliente e do administrador do provedor em nuvem com o módulo ModInf.

Na ontologia IaaSOnt há outras classes (Figura 16) que são auto-explicativas para os estudiosos e profissionais da área de computação em nuvem. Contudo, maiores detalhes sobre todas as classes dessa ontologia estão disponíveis no Anexo B.

6.4.2.2 Regras e Consultas para a Ontologia IaaSOnt

A concepção da ontologia em OWL com suas 109 classes, 48 propriedades de objetos e 66 propriedades de dados, possibilitou a representação de um sistema que buscar gerenciar recursos virtualizados de forma automática. Além da representação, a IaaSOnt também pode armazenar informações por meio da criação de indivíduos, que seriam entidades (objetos/instanciações) de uma determinada classe, tornando-se assim em uma base de conhecimento (*Knowledge Base* (KB)). Tais informações podem ser extraídas na forma de relacionamentos, consultas ou regras.

Uma propriedade de objeto (*ObjectProperty*) dentro de uma ontologia OWL é utilizada para criar uma relação entre duas classes. Por exemplo, propriedade de objeto *hasVirtualMachine* relaciona a classe *Client* com a classe *VirtualMachine*, indicando que pode haver uma relação de posse do cliente sobre uma máquina virtual. De forma semelhante, uma propriedade de dados estabelece uma relação entre indivíduos com algum tipo de dado. Por exemplo, pode-se criar

um individuo da classe *Client* e fazer a relação com a propriedade de dado (*DataProperty*) denominada *hasEmail* que aceita valores do tipo *string* e no valor dessa relação é informado o e-mail do cliente. A Figura 19 exibe algumas das propriedades de objetos e dados da IaaSOnt.

Figura 19 – Algumas propriedades de objetos (em azul) e dados (em verde) da IaaSOnt.

Tais relações garantem a navegação e também a busca semântica de informações. Outra proposta deste projeto de doutorado foi a criação de consultas e regras capazes de auxiliar os mecanismos propostos na tomada de decisão. Ou seja, consultas e regras capazes de tomar decisões baseadas em certos critérios e parâmetros que devem ser informados pelos clientes e administradores das arquiteturas virtualizadas (SLA/QoS, Políticas, etc.).

Dessa forma, foi utilizada uma API em Java denominada OWLAPI²⁵ para a navegação semântica entre classes, propriedades e indivíduos. Essa API é composta por um conjunto de interfaces que são usadas para carregar, consultar e manipular ontologias, sendo capaz de realizar processos de raciocínio (ZHAO; ZHANG; ZHAO, 2012).

Para a criação de regras foi utilizado o *plug-in* SWRLTab no Protégé e para a implementação uma API denominada SWRLAPI²⁶. As regras SWRL oferecem algumas funcionalidades incorporadas (*Built-Ins*²⁷) que podem ser utilizadas para realizar comparações, operações matemáticas, trabalhar com *strings*, manipular datas e listas, etc.

Dessa forma, torna-se possível a criação de diversas regras que associadas aos conceitos da ontologia podem dar novos significados, buscar, extrair ou incluir informações, etc. Um exemplo simples e bastante utilizado para explicar a criação e feito de uma regra é a famosa

²⁵ <<https://github.com/owlcs/owlapi/>>

²⁶ <<https://github.com/protegeproject/swrlapi-project/>>

²⁷ <<http://www.daml.org/2004/04/swrl/builtins.html>>

regra que identifica a existência de uma pessoa que seja um tio. Dentro de uma ontologia que contenha pessoas e alguns relações entre objetos que representam a relação de parentesco entre elas podemos criar a seguinte regra SWRL (*hasUncle*):

- **Exemplo de regra SWRL:** A regra específica que se uma pessoa x tem como pai a pessoa y E essa pessoa y tem um irmão z **ENTÃO** afirmasse que a pessoa x tem uma relação onde ela tem z como sendo seu tio (*hasUncle*):

```
hasParent(?x, ?y) ^ hasBrother(?y, ?z)
-> hasUncle(?x, ?z))
```

As consultas em ontologia OWL podem ser feitas de diversas formas, no trabalho de mestrado de Nakamura (2012) foi abordada a proposta de utilização da linguagem SPARQL²⁸ para a consulta de informações na ontologia (NAKAMURA, 2012). Por outro lado, outra forma, abordada neste projeto, foi a proposta da utilização de consultas com SQWRL que são baseadas e semelhantes a linguagem SWRL. Outra vantagem é que, diferentemente da SPARQL, a SQWRL possui uma base semântica forte e permite ao usuário criar diversas operações de consultas sobre ontologias OWL (O'CONNOR; DAS, 2009). A seguir são exemplificadas duas regras para o melhor entendimento sobre SQWRL:

- **Exemplo de consulta SQWRL:** A consulta abaixo retorna/seleciona as pessoas ($?p$) e suas idades ($?age$), cuja a idade da pessoa seja maior que 17 (anos):

```
Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17) ->
query:select(?p, ?age)
```

- **Exemplo de consulta SQWRL inferindo uma regra SWRL:** A regra/consulta abaixo classifica as pessoas que tenham idade maior que 17 (anos) como sendo da classe adulta (*Adult*):

```
Person(?p) ^ hasAge(?p, ?age) ^ swrlb:greaterThan(?age, 17) -> Adult(?p)
```

As consultas SQWRL dispõe de diversas outras funções interessantes como ordenar o resultado de uma seleção (ex: *query:orderBy(?age)*), calcular uma média (ex: *query:avg(?age)*), retornar a quantidade de resultados (*query:count*), etc.

Ainda dentro deste contexto, elementos de *software* conhecidos como racionalizadores ou motores de raciocínio (*reasoners*) são utilizados para inferir informações de consequências

²⁸ <<https://en.wikipedia.org/wiki/SPARQL>>

lógicas baseado em um conjunto fatos ou axiomas afirmados na ontologia ou regras. O *reasoner* Pellet²⁹ é um exemplo dessa máquina de inferência, ele é de código aberto sob uma licença de utilização liberal. Ele é capaz de avaliar ontologias desenvolvidas com a linguagem OWL, verificando se essas ontologias estão de acordo com os serviços de inferência definidos pela lógica de descrição (SIRIN *et al.*, 2007). O Pellet também é capaz de lidar com regras na especificação/linguagem *SWRL*.

Outro *reasoner* investigado neste projeto, foi o Hermit³⁰, ele é capaz de inferir ontologias OWL 2 e possui melhorias no desempenho do raciocínio/inferência em ontologias do mundo real (GLIMM *et al.*, 2014). Apesar de uma melhor integração com as versões mais recentes da OWLAPI, uma desvantagem apresentada pelo Hermit foi que, até a última versão utilizada (ver. 1.3.8.500), ele não suportava regras SWRL que utilizavam assertivas (afirmações) *Built-Ins*. A seguir algumas consultas/regras feitas para a IaaSOnt são apresentadas com o objetivo de exemplificar a utilização deste recurso no projeto.

- **IaaSOnt - Exemplo de consulta 1:** Encontrando um cliente pelo seu ID.

```
Client(?c) ^ hasId(?c, ?id) ^ swrlb:equal(?id, 1) -> sqwrl:select(?c)
```

- **IaaSOnt - Exemplo de consulta 2:** Encontrando o cliente pelo seu E-mail e Senha (autenticação).

```
Client(?c) ^ hasEmail(?c, ?email) ^ hasPassword(?c, ?pwd) ^  
swrlb:equal(?email, \"nakamura@icmc.usp.br\") ^  
swrlb:equal(?pwd, \"123#mudar\") -> sqwrl:select(?c)
```

- **IaaSOnt - Exemplo de consulta 3:** Encontrando os clientes com uma determinada necessidade por SLA de forma reduzida (apenas 2 atributos de QoS), ou seja, encontrando o cliente (?c), atributo de QoS 1 (Disponibilidade - *Availability*) e o atributo de QoS 2 (Capacidade - *Capacity*) cujo o(s) cliente(s) tenha(m) um gerenciamento que utiliza (tem) uma política baseada em SLA (*hasSLA*) relacionada a uma QoS, a qual tenha o atributo 1 (disponibilidade) com valor maior que 90 e o atributo 2 (capacidade) com valor maior ou igual a 0.5.

```
Client(?c) ^ hasManagement(?c, ?man) ^ hasPolicy(?man, ?policy) ^  
hasSLA(?policy, ?qos) ^ hasQoSAttribute(?qos, ?qosat1) ^  
asAvailability(?qosat1, ?availvalue) ^ hasQoSAttribute(?qos, ?qosat2) ^  
hasCapacity(?qosat2, ?capvalue) ^ swrlb:greaterThan(?availvalue, 90) ^  
swrlb:greaterThanOrEqual(?capvalue, 0.5)  
-> sqwrl:select(?c, ?qosat1, ?qosat2)
```

²⁹ <<https://github.com/stardog-union/pellet>>

³⁰ <<https://github.com/ignazio1977/hermit-reasoner>>

Dessa forma, o ModInf pode utilizar essas regras e consultas para identificar e auxiliar quando devem ser executadas novas reconfigurações da arquiteturas, com ou sem limitações, que visam atender as necessidades pré-definidas na SLA/QoS. Por exemplo, elas podem facilitar a execução das políticas de gerenciamento de recursos, financeiro, etc. Quando configuradas, essas políticas podem ser representadas tanto por relacionamentos semânticos (acessando e navegando entre classes e propriedades via OWLAPI), quanto por consultas ou regras (SWRLAPI, SQWRL, etc.). Assim, pode-se caracterizar as etapas inteligentes da computação autônoma ("laço de controle") como foi proposto, ou seja, fazendo uso dos recursos existentes na Web Semântica como o processamento de inferências e/ou execução de consultas e regras.

6.5 Organização e Funcionamento

Os mecanismos propostos foram desenvolvidos para serem utilizados em arquiteturas virtualizadas. A proposta é que, uma vez inseridos na arquitetura como módulos (MAC e MAO), os mecanismos e a arquitetura troquem informações para que a AutoConfiguração e a Auto-Otimização ocorram de forma eficiente e automática. O modelo conceitual da Figura 20 exhibe a interação entre eles e a seguir são descritas algumas etapas funcionais (NAKAMURA *et al.*, 2014):

Figura 20 – Modelo Conceitual da arquitetura virtualizada utilizando os mecanismos (NAKAMURA *et al.*, 2014).

1. O primeiro passo é a criação de um SLA entre o cliente e o administrador do provedor da nuvem. Este processo é realizado fora da arquitetura, mas os parâmetros acordados devem

- ser incluídos no passo seguinte;
2. O administrador poderá por meio do Prótégé ou um aplicativo da Web para criar uma conta de cliente, registrar os limites financeiros, SLA e/ou parâmetros de gerenciamento e otimização. O cliente poderá escolher algumas políticas personalizadas relativas a questões financeiras (por exemplo, restrições e restrições sobre o consumo de horas de máquina, limites sobre o número de VMs *on-line* ou aumentar o desempenho para datas especiais);
 3. Todas as configurações da etapa 2 também podem ser salvas em um banco de dados para facilitar a utilização por outras aplicações e transformadas em políticas e parâmetros estáticos a serem usados nas regras SWRL;
 4. Usando uma API ou um aplicativo da Web (*Dashboard*), o cliente irá criar e configurar a sua infraestrutura virtual;
 5. Os agentes computacionais monitoram constantemente a infraestrutura e coletam informações que são armazenadas na base de conhecimento (ontologia);
 6. O *loop control* será executado de tempos em tempos, seja monitorando os recursos, executando previsões, executando algoritmos de otimização e fazendo inferências com base nos dados de consultas ou regras SWRL na ontologia;
 7. O processo de inferência classifica e relaciona as informações do provedor, clientes, recursos e políticas a serem utilizadas pelos mecanismos de autoconfiguração e auto-otimização. Estes mecanismos poderão analisar os dados e decidir se uma nova reconfiguração é necessária. Eles também podem otimizar a própria reconfiguração escalando os recursos adequadamente e respeitando as políticas personalizadas;
 8. Finalmente, os mecanismos acionam os agentes computacionais para promover o auto-dimensionamento, e assim, prover uma nova infraestrutura adequada a necessidade do cliente.

6.6 Considerações Finais

Neste capítulo foram apresentados os mecanismos de AutoConfiguração e Auto-Otimização em conjunto com seus componentes.

Duas propostas para a autoconfiguração foram apresentadas na seção 6.2, uma delas foi baseada em monitoração de recursos, e outra que considerou técnicas de previsão. Além disso, também foi feito o cálculo do retorno do investimento (ROI) e implementado o *framework* FAMaV que facilitou a execução e o gerenciamento da autoconfiguração. A proposta deste projeto de doutorado em utilizar o conceito de autoconfiguração da teoria da computação autônoma no gerenciamento de recursos em arquiteturas virtualizadas resultou em um trabalho

colaborativo com um projeto de mestrado, que implementou os algoritmos de monitoração e previsão, e em um projeto de iniciação científica que colaborou na implementação do *framework* FAMaV.

Na seção 6.3 foi apresentada a metodologia desenvolvida neste projeto de doutorado para o cumprimento do SLA na qual foram considerados os atributos de QoS: Capacidade, Tempo de Resposta, Disponibilidade e Custo (por hora) e formas de agregação e normalização desses atributos de QoS com base na realização de uma composição de serviços/recursos (máquinas virtuais) para então determinar se a QoS prestada pela atual infraestrutura do cliente está realmente atendendo o SLA contratado. Para solucionar o problema do não atendimento do SLA, foram elaborados algoritmos de otimização que tinham como objetivo otimizar a quantidade de recursos utilizados visando a atender o SLA e diminuir o custo da infraestrutura virtual. Dessa forma, para validar a proposta do mecanismo de auto-otimização, dois algoritmos foram implementados em um projeto de Mestrado desenvolvido de forma colaborativa que utilizou a proposta e a metodologia desenvolvidas neste projeto de doutorado. O primeiro algoritmo implementado foi um algoritmo determinístico para servir de comparação e o segundo foi um algoritmo de otimização baseado em evolução denominado micro algoritmo genético (μ AG).

Alguns componentes desenvolvidos neste projeto de doutorado destinados a serem de uso comum pelos mecanismos mencionados foram apresentados na seção 6.4. Um destaque especial foi dado na seção 6.4.2.1, pois aborda um componente importante (Módulo de Inferência) responsável pela consulta de informações e tomada de decisão baseada em consultas e regras semânticas que utilizam como base de conhecimento a ontologia IaaSOnt desenvolvida neste projeto de doutorado. O processo da metodologia para a criação da ontologia foi explicado na seção 6.4.2.1 e detalhes do desenvolvimento da ontologia e a documentação gerada pela metodologia estão no Anexo B.

No próximo capítulo são apresentados resultados de validações de desempenho.

AVALIAÇÃO DOS MECANISMOS

7.1 Considerações Iniciais

Este capítulo tem como objetivo apresentar os resultados de testes de avaliações relacionados aos mecanismos de autoconfiguração e auto-otimização. Os experimentos de avaliação de desempenho foram realizados de forma separada e isolada, ou seja, diversos experimentos foram realizados para cada componente do projeto. Dessa forma, foi possível avaliar individualmente características dos algoritmos de autoconfiguração e auto-otimização. Todos os processos de avaliação de desempenho foram realizados com a colaboração dos projetos de mestrados e iniciação científica associados a este projeto de doutorado. A avaliação dos componentes da autoconfiguração foi realizada por elementos computacionais em uma execução real e os algoritmos que compõem o mecanismo de auto-otimização foram avaliados por meio da execução de simulações com auxílio do simulador CloudSim.

7.2 Avaliação de Desempenho dos Componentes do Mecanismo de Auto-Configuração

Nesta seção são apresentadas as avaliações de desempenho das propostas relacionadas a autoconfiguração que foram discutidas na seção 6.2. Dessa forma, cinco experimentos serão discutidos: Experimento sem nenhuma abordagem de reconfiguração (**Exp0**), Experimento baseado em monitoração (**Exp1**), Experimento baseado em previsão (**Exp2**), Experimento que considera tanto a monitoração quanto a previsão (**Exp3**) e Experimento que avaliou o *framework* FAMaV (**Exp4**).

7.2.1 Configuração dos Ambientes de Experimentos

Para os experimentos sem configuração, baseado em monitoramento, baseado em previsão e baseado em ambos, ou seja, experimentos 0, 1, 2 e 3 respectivamente, o ambiente de experimentos foi composto por um servidor no qual o *hypervisor* KVM era responsável por gerenciar quatro VMs com as configurações idênticas. A configuração computacional do servidor era composta por um processador Intel Core 2 Duo T8100 com 2.10GHz de processamento, 4GB de memória RAM, sistema operacional Linux Ubuntu 12.04 e Disco Rígido (HD) com 500GB. As máquinas virtuais, hospedadas, foram configuradas para iniciar com 1GB de RAM, um único núcleo de processamento e Disco Rígido (HD) com 6GB. Todas as VMs poderiam expandir seus recursos até o limite da configuração do servidor (DIAS *et al.*, 2014).

Além do agente monitor, alguns serviços Web (*Web services*) foram implantados na VMs para serem consumidos pelos clientes. Foram criados dois tipos de *Web services* que são responsáveis por receber as requisições do cliente e gerar uma carga de trabalho na máquina virtual para Memória e Disco. Dessa forma, os *Web services* têm função semelhante ao dos *benchmarks*, o responsável por gerar carga para a memória criando diversos vetores de *Strings* de 8 bytes cada e o de disco realiza operações de leitura, escrita, releitura, reescrita e remoção de arquivo. A requisição do cliente é composta por uma sequência de números de forma senoidal; nos experimentos foram utilizados 15 valores (um para cada período/dia): 5, 10, 15, 20, 25, 20, 15, 10, 5, 5, 10, 15, 20, 25, 20. Com esses valores os *Web services* geram uma carga multiplicando o valor do período enviado pelo cliente vezes um valor aleatório que definirá o tamanho do arquivo (disco) ou a quantidade de vetores (memória) (DIAS, 2015).

O cliente que consome os *Web services* possui baixa capacidade computacional (*thin client*), uma vez que o processamento pesado é executado na nuvem e, assim, a sua configuração não afeta os resultados do experimento.

Para o **Exp1**, baseado em monitoração, foi considerado um limiar de 90% como o máximo de utilização permitida para cada recurso antes de uma reconfiguração. Assim, o adaptador elástico deve executar a elasticidade de recursos de memória e disco quando a utilização desses recursos atingissem 90%. Este limite foi considerado seguro, pois garantia que nenhuma requisição fosse negada.

Para o **Exp2**, que emprega a previsão baseada nas equações de 6.1 até 6.5 do modelo de Holt-Winter, foram considerados: um intervalo de tempo de Sazonalidade $s = 4$, um número de períodos a serem previstos $m = 1$ (isso significa que a previsão ocorre apenas para o dia seguinte), a constante de Nível (*Level*) com $\alpha = 0,44$, a constante de Tendência (*Trend*) com $\beta = 0,44$ e a constante de sazonalidade com $\gamma = 0,2$. Tais constantes foram ajustadas durante o desenvolvimento e testes dos algoritmos (DIAS *et al.*, 2014).

O **Exp3**, emprega as abordagens apresentadas no **Exp1** e **Exp2** em conjunto. No início de cada período (dia) as MVs da infraestrutura são reconfiguradas para atender a previsão, porém

durante a execução da carga de trabalho, a monitoração está ativa e o adaptador elástico realiza reconfigurações.

No **Exp4**, que trata da avaliação de desempenho do *framework* FAMaV em relação a tradicional ferramenta de gerenciamento de MVs denominada Virsh. Para esse experimento foi utilizada uma máquina real com processador Intel(R) Core(TM) i5-4430 CPU com 3.00GHz de processamento, 16Gb de memória RAM e 1TB de disco. Esse servidor hospeda cinco máquinas virtuais com um único núcleo, 1GB de memória RAM e 20GB de disco (NEVES *et al.*, 2014).

7.2.2 Planejamento de Experimentos

Para o planejamento dos experimentos **Exp0**, **Exp1**, **Exp2** e **Exp3** foi considerada uma série temporal com 15 períodos (dias), onde o cliente realiza requisições aos *Web services* que geram cargas de trabalhos distintas.

No **Exp0**, nenhuma abordagem de reconfiguração é executada, o intuito desse experimento foi apenas gerar resultados e verificar o seu comportamento para que possam ser comparados com as propostas de monitoração e previsão. Já os experimentos **Exp1**, **Exp2** e **Exp3** consideram alguns fatores e níveis apresentados na Tabela 6 (DIAS *et al.*, 2014).

Tabela 6 – Fatores e Níveis para os experimentos **Exp1**, **Exp2** e **Exp3**

Fator	Níveis
Recurso	Memória ou Disco
Monitoração	Com ou Sem
Previsão	Com ou Sem

O primeiro fator é o tipo de recurso que será reconfigurado podendo variar entre dois níveis: Memória e Disco. Testes foram realizados com o recurso CPU, porém embora adicionar um nova *virtual CPU* (vCPU) fosse possível, o tempo para o sistema operacional da máquina virtual (*Guest*) conseguir identificar esse novo núcleo e começar a utilizá-lo iria impactar na execução dos experimentos. Além disso, a instalação e configuração dos recursos e agentes de *software* tanto no *Host* (máquina real com *hypervisor*) quanto no *Guest*, para que a nova vCPU fosse reconhecida sem a necessidade da reinicialização do *Guest* era dispendiosa e não trivial.

Os resultados coletados nos experimentos são os valores em *MegaBytes* (MB) de recursos "Disponíveis" e "Observados" durante os períodos. Dessa forma, a variável "Disponível" corresponde a um valor de recurso (Memória ou Disco) em MB ajustado pelo adaptador elástico (se o experimento utilizar a monitoração) ou um valor reconfigurado de acordo com a previsão (se o experimento utilizar a proposta de previsão). Já a variável "Observado" é a carga de trabalho imposta, ou seja, é o valor que realmente foi utilizado/consumido de cada recurso.

Para determinar se reconfiguração ficou próxima do adequado com relação a necessidade por recurso (tentando alocar a quantidade de recursos sem excessos ou faltas) foram calculadas duas variáveis de resposta para os experimentos de **Exp1** a **Exp3** sendo elas:

- Erro Quadrático Médio (EQM) (*Mean Square Error*): o cálculo deste erro indica a diferença entre os valores disponíveis (previstos/reconfigurados) e os observados, que no fim resulta na média dos erros.
- : Erro Médio Percentual (EMP) (*Average Percentage Error*): é a porcentagem da comparação entre o valor observado e o valor disponível. A previsão ou monitoramento será melhor quando esse percentual for menor.

De fato o Erro Quadrático Médio e o Erro Médio Percentual são bem parecidos, entretanto eles têm propósitos diferentes. O primeiro verifica a diferença entre uma previsão e o valor observado fornecendo uma medida de precisão, o segundo fornece uma porcentagem do erro em relação ao valor real (DIAS, 2015).

O planejamento de experimentos para o **Exp4** (FAMaV) avaliou o tempo de execução das funções de gerenciamento de MVs que foram desenvolvidas, cada função foi executada/repetida 35 vezes para que fossem calculados uma média e o intervalo de confiança no qual se considerou índice de confiança de 95% (NEVES *et al.*, 2014). As médias dos tempos de resposta para executar uma função, os valores dos limites (superiores e inferiores) e os valores dos intervalos de confiança estão em milissegundos.

Os resultados de desempenho da comparação FAMaV e Virsh foram divididos em três grupos: *Simples*, *Listas* e *Complexas*. O primeiro grupo (*Simples*) são as funções de gerenciamento consideradas simples (Ligar, Desligar, Reiniciar, Hibernar, Retomar e coletar informações das VMs), o segundo grupo (*Listas*) são as funções simples mais que gerenciam várias MVs ao mesmo tempo (no caso do Virsh foi preciso criar *scripts* para alcançar tal funcionalidade). Por fim, o último grupo (*Complexas*) engloba algumas das funções de gerenciamento mais complexas (realizar migração de uma MV, alterar a quantidade de memória, CPU, etc.).

7.2.3 Resultados e Análise

Conforme mencionado no planejamento, os resultados do **Exp0** são apresentados apenas para esclarecer como seria o comportamento da infraestrutura se não houvesse nenhum mecanismo de reconfiguração de recursos. Portanto, para esse experimento não foi utilizada nem a reconfiguração por monitoração e nem a reconfiguração por previsão. Portanto, as Figuras 21 e 22 exibem os resultados do **Exp0** para os recursos de memória e disco, respectivamente.

Analisando o resultado para a memória do **Exp0** observa-se que, como era esperado, houve momentos que esse recurso foi insuficiente e em outros ele esteve ocioso. A característica da carga de trabalho imposta (baseada em uma distribuição senoidal) que foi distribuída nos 15 períodos fica destacada no gráfico. No início do experimento havia disponível 1024MB (1GB) de memória (linha tracejada vermelha) e este valor não sofreu alteração até o fim do experimento. Contudo, a carga de trabalho observada (linha sólida preta) que foi imposta pelo cliente ao *Web*

Figura 21 – **Exp0**: Memória - Sem mecanismo de reconfiguração (Traduzido de (DIAS *et al.*, 2014)).

service iniciava baixa (vale) e iria aumentando até um certo ponto (pico) depois ela voltaria novamente para uma carga baixa e mais uma vez para carga alta (lembrando um senoide: 5, 10, 15, 20, 25, 20, 15, 10, 5, 5, 10, 15, 20, 25, 20).

Dessa forma, ainda analisando a Figura 21, na maioria do tempo de execução do experimento (períodos de 2 até 8 e de 11 até 15) a utilização de memória atingiu o limite, mas como nesse experimento não havia um mecanismo de reconfiguração, muitas requisições foram enfileiradas, prejudicando o desempenho, e outras não chegaram a serem atendidas.

Figura 22 – **Exp0**: Disco - Sem mecanismo de reconfiguração (Traduzido de (DIAS *et al.*, 2014)).

O comportamento para o recurso disco no **Exp0** (Figura 22) é semelhante. A configuração inicial da VM para disco é de 6GB e este valor disponível é mantido do início ao fim do experimento. Novamente, em duas ocasiões o valor observado que é imposto pela carga de

trabalho (senoide) não é suficiente para atender a demanda e em outros momentos há uma sobra de recursos.

Para esses casos sem um mecanismo de autoconfiguração, seria preciso que o cliente de alguma forma monitorasse quando os recursos de suas MVs chegam ao limite ou estão demasiadamente ociosos para realizar o ajuste (alterando a qualidade de recursos virtuais: modificando a quantidade de CPU, Memória, Disco ou iniciando/finalizando outras instâncias de máquinas virtuais).

Os resultados para o **Exp1**, que considera uma reconfiguração automática baseada apenas em monitoração, estão apresentados na Figuras 23 e 24.

Figura 23 – **Exp1**: Memória - Com mecanismo de reconfiguração baseado em monitoração (Traduzido de (DIAS *et al.*, 2014)).

Figura 24 – **Exp1**: Disco - Com mecanismo de reconfiguração baseado em monitoração (Traduzido de (DIAS *et al.*, 2014)).

No caso do **Exp1**, tanto para memória quanto o disco, o mecanismo foi capaz de monitorar a utilização de recursos e propor novas reconfigurações que satisfizessem os requisitos da carga de trabalho que foi imposta. Em relação à memória (Figura 23), houve uma aproximação maior entre o valor observado e o valor disponível.

O cálculo do EQM para esse experimento resultou em um valor de 125,45 para a memória e 1345,69 para o disco. O cálculo do EMP para a memória teve valor 7,6% e para o disco o valor de 26%. O recurso memória apresenta um melhor resultado, pois trata-se de um recurso de gerenciamento mais trivial, o seu dimensionamento (alteração da quantidade de memória) é feito de forma direta devido ao avançado sistema de gerenciamento e controle de memória dos *hypervisors*.

Portanto, a memória é mais "elástica" do que o disco; para alterar o valor da memória pode-se utilizar uma função da ferramenta Virsh (ou um método do *framework* FAMaV) passando por parâmetro o novo valor de memória. O sistema operacional do *Guest* reconhece a nova quantidade de memória quase que imediatamente. Contudo, é impraticável redimensionar o tamanho da unidade de disco de uma MV (*image*) em tempo de execução. Por exemplo, com o *hypervisor* KVM (utilizado nestes experimentos), durante a criação da unidade virtual de disco, o administrador da VM deve definir o tamanho dessa unidade e se esse espaço será alocado inteiramente no início ou será incrementado até atingir esse limite. Dessa forma, a solução utilizada nos experimentos foi adicionar um novo volume já existente para aumentar a quantidade de espaço de disco total da MV (o sistema operacional do *Guest* deve ter habilitado os módulos de *hotplug* no *kernel* para evitar a necessidade de reinicialização).

Essas características também refletem nos resultados e o valor disponível da memória acompanha o valor observado muito mais próximo do que o disco. Lembrando que a reconfiguração por monitoração, nestes experimentos é feita quando o recurso atinge 90% de utilização, ou seja, há aproximadamente 10% de lacuna que permitiu que a reconfiguração ocorresse antes dos limites serem alcançados.

Os resultados do **Exp2**, o qual considera uma reconfiguração baseada na previsão dada pela técnica que aplica o modelo de Holt-Winter, são exibidos nas Figuras 25 e 26.

No início de cada período do **Exp2** é aplicado uma reconfiguração baseando-se na previsão. Dessa forma, foram realizadas 15 reconfigurações no decorrer do experimento. Assim, os recursos de memória e disco tiveram suas capacidades previstas pelo contexto histórico e o mecanismo de autoconfiguração foi capaz de fornecer recursos suficientes para atender a todas as solicitações (linha preta sólida) (DIAS *et al.*, 2014).

Neste **Exp2**, o EMP para a memória foi de 17,92% e para o disco foi 31,44%. Analisando o gráfico, pode-se observar que existe uma distância entre as curvas dos valores disponíveis e observados, obter essa diferença aplicou-se o cálculo do EQM que resultou em 247,78 e 1640,85 para a memória e disco, respectivamente.

Figura 25 – **Exp2**: Memória - Com mecanismo de reconfiguração baseado em previsão (Traduzido de (DIAS *et al.*, 2014)).

Figura 26 – **Exp2**: Disco - Com mecanismo de reconfiguração baseado em previsão (Traduzido de (DIAS *et al.*, 2014)).

Com base na análise dos gráficos e dos resultados dos cálculos dos erros, nota-se que os resultados do **Exp1** (monitoração) são melhores que o do **Exp2** (previsão). A razão para isso é que a monitoração é executada continuamente, sempre verificando a utilização de recursos e realizando a reconfiguração quando necessária, por outro lado, a previsão executa uma reconfiguração apenas no início de cada período (dia) e não considera as cargas de trabalho que ocorrem durante o tempo de um período.

Para o **Exp3** ambas as abordagens de reconfiguração estavam ativas (monitoração e previsão). Os resultados deste experimentos são exibidos nas Figuras 27 e 28.

Para esse **Exp3** a expectativa era que os resultados fossem melhores do que o **Exp1** e

Figura 27 – **Exp3**: Memória - Reconfiguração baseado em ambas abordagens (Traduzido de (DIAS *et al.*, 2014)).

Figura 28 – **Exp3**: Disco - Reconfiguração baseado em ambas abordagens (Traduzido de (DIAS *et al.*, 2014)).

Exp2, porém o cálculo do EMP para memória e disco foram 10,33% e 28,69%, respectivamente. E o cálculo do EQM obter 156,29 para a memória e 1480,24 para o disco. Portanto, com relação ao **Exp2**, este experimento mostrou uma melhoria relativa no gerenciamento dos recursos. Mas, com resultados inferiores ao **Exp1**. A justificativa para isso é que a previsão realizada no início do período teve estimativas de cargas maiores ou menores do que aquelas que realmente eram necessárias. Assim, a monitoração conseguiu realizar um ajuste mais refinado de acordo com a carga de trabalho que estava sendo imposta em um momento particular.

Embora a monitoração tenha tido o melhor resultado a sua atuação é reativa, enquanto a da previsão é proativa. A proposta da previsão poderia ter melhores resultados se o período observado fosse maior, visto que a carga utilizada nos experimentos tem altos e baixos sazonais.

Contudo, uma boa parte dos sistemas não tem cargas de trabalhos previsíveis, por isso grande partes dos grandes provedores optam pela monitoração, por ser menos complexa e, de modo geral, apresentar resultados satisfatórios. Porém, em alguns casos específicos ou para obter melhores resultados pode ser preciso utilizar uma ou outra abordagem. Isso pode depender do tipo de aplicação e de carga de trabalho ao qual a infraestrutura estará exposta. Além disso, tanto o provedor quanto o cliente podem ter algum tipo de restrição, alguma regra de negócio ou acordo de nível de serviço que exija no mínimo uma monitoração e reconfiguração mais personalizada.

Os resultados da comparação entre o FAMaV e o Virsh do **Exp4** são apresentados nas Figuras 29, 31 e 32 para as funções de gerenciamento simples, de listas e complexas, respectivamente.

Figura 29 – **Exp4**: FAMaV x Virsh - Funções de gerenciamento simples (Adaptado de (NEVES *et al.*, 2014)).

Os resultados de comparação para as funções simples indicam que o Virsh obteve melhor desempenho para todas as funções simples. O resultado que mais merece destaque, como é o caso da função *Restart* que apresentou desempenhos muito inferior com o FAMaV. Isso se deve à forma como a função *Restart* foi implementada (Figura 30).

No FAMaV, essa função faz verificações prévias (verifica o estado da máquina - ligada/desligada) e re-utiliza os métodos (*PowerOff* e *PowerOn*), antes de realmente realizar a reinicialização. Embora haja uma diferença considerável de tempo de execução na comparação entre as duas ferramentas de gerenciamento, que está justificada mais adiante, os tempos apresentados estão em milissegundos, o que torna a FAMaV plausível de utilização.

Outro ponto relevante que precisa ser esclarecido é que esses resultados são apenas dos tempos gastos para a execução das funções (envio do comando ao *hypervisor*), ou seja, eles não consideram o tempo gasto pelas VMs para executar as respectivas ações. Assim, o tempo gasto pela função *PowerOn* (FAMaV) ou *Start* (Virsh) não engloba o tempo gasto para a VM chegar ao estado de ligada/iniciada e estar pronta para o uso ³¹.

³¹ Nos experimentos realizados, o tempo gasto para executar a função "Ligar" em ambas as ferramentas foi inferior

Figura 30 – Implementação da Função *Restart*. Figura 31 – **Exp4:** FAMaV x Virsh - Funções em listas (Adaptado de (NEVES *et al.*, 2014)).

Entretanto resultados das funções de lista estão mais aproximados para as duas ferramentas. Nessas funções de lista o usuário deseja executar uma determinada função sobre uma lista de MVs. Por exemplo, o usuário deseja ligar cinco MVs no mesmo instante. A FAMaV já prevê este tipo de necessidade e então possui métodos que aceitam como parâmetro uma lista de MVs. Contudo, na versão de Virsh utilizada nos experimentos não havia tal funcionalidade, a

à 1 segundo, porém a MV utilizada no experimento gastou um tempo em torno de 12 segundos para a completa inicialização do sistema operacional.

solução para realizar a comparação foi a criação de um *shell script* com um laço de repetição que executasse a determinada função n vezes, sendo n o número de MVs passada como parâmetro para o *script*. O maior destaque nesses resultados vai para a função *RestartList* que obteve um desempenho muito inferior, essa função faz uso da função *Restart* e, conforme justificado anteriormente, esse tempo é justificado pela verificação do estado da MV.

Figura 32 – FAMA V x Virsh - Funções de gerenciamento complexas (Adaptado de (NEVES *et al.*, 2014)).

O resultado da função complexa *VCpuInfo* (Virsh) também teve resultado superior, porém a função *UseCPU* da FAMA V é mais refinada. Ao contrário da Virsh, ela exibe uma porcentagem com a média das vCPUs utilizada pela VM. Finalmente, nos experimentos a função *Migrate* apresentou um resultado melhor quando utilizada a FAMA V. A justificativa para isso é que na FAMA V essa função recebe um parâmetro que determina a largura de banda de transmissão na rede, nos experimentos da FAMA V utilizou-se o valor de 100Mbps, por outro lado, nos experimentos da Virsh utilizou apenas a função "*migrate –live*" que não aceita esse parâmetro. Após uma investigação na documentação da Virsh, descobriu-se que a velocidade pode ser configurada utilizando outra função denominada "*migrate-setspeed*", e ao executar o comando "*migrate-getspeed*" o usuário pode descobrir a velocidade padrão, que no caso da VM utilizada no experimento com Virsh era de apenas 32Mbps (NEVES *et al.*, 2014).

Dessa forma, os resultados indicam que a FAMA V possui desempenho menor que *Virsh*, este resultado já era esperado devido a diferença no tempo de execução entre as linguagens Java e C. Contudo, a FAMA V CLI e especialmente a FAMA V API podem ser estendidas e tornam o gerenciamento de MVs mais dinâmico e os tratamentos de falhas menos complexos. Portanto, avaliar e analisar os resultados do FAMA V foi uma etapa relevante dentro do contexto deste projeto de doutorado, pois era preciso identificar dois fatores: 1) a viabilidade da utilização

da linguagem Java (que permite um desenvolvimento mais produtivo) e 2) se o resultados de desempenho não teriam um impacto que comprometesse o seu uso pelos mecanismos de autoconfiguração.

7.3 Avaliação de Desempenho dos Componentes do Mecanismo de Auto-Otimização

Nesta seção são apresentadas as avaliações de desempenho das propostas relacionadas a auto-otimização que foram discutidas na seção 6.3. Dessa forma, os dois algoritmos de otimização foram avaliados considerando diversas quantidades de máquinas virtuais (entre 10 e 100 MVs). Também foram realizadas avaliações dos tempos de processamento e validação da ontologia e dos tempos de execução de algumas consultas SQWRL pertinentes a este projeto de doutorado.

7.3.1 Configuração do Ambiente de Experimentos

A execução dos experimentos foi realizada utilizando-se o CloudSim versão 3.0.3 que foi executado em um computador composto por um processador AMD Phenom(tm) II X6 1090T, 16 GB de memória RAM, 1,5 TB de disco e sistema operacional Ubuntu 14.04.3 LTS. O CloudSim foi configurado para utilizar três tipos de instâncias de MVs: **m3.medium**, **m3.large** e **m3.xlarge** baseadas nas configurações da Amazon EC2³² (AZEVEDO, 2016).

A respeito da configuração do μ AG é importante lembrar que o espaço de busca é definido permitindo uma variação de 50% para cima ou para baixo do número MVs desejadas. A população para esse algoritmo foi ajustada em 5 indivíduos, sendo cada deles uma possível configuração MV (s, m, l) , que o operador de mutação tem taxa de 20% e que a inicialização é feita de forma aleatória gerando os indivíduos (s, m, l) tais como $\min \leq (s + m + l) \leq \max$, onde o intervalo possível é definido por $[min, max]$. As configurações para definir valores de parâmetros do algoritmo μ AG podem ser encontrados no Anexo A.

Para a configuração do ambiente dos experimentos que avaliou os tempos de processamento e validação da ontologia, bem como os tempos de execução das consultas SQWRL, foi composto por um computador com processador Intel i7-3632QM (2,20GHz), 8GB de memória RAM e 500GB de disco e sistema operacional Microsoft Windows 8.1 Pro 64bits.

7.3.2 Planejamento dos Experimentos

Tanto o algoritmo determinístico quanto o μ AG foram aplicados em dez cenários com diferentes quantidades de MVs. A quantidade de VMs disponíveis é múltiplo de dez de cada cenário, ou seja, no primeiro havia um *data center* com 10 MVs, no segundo um *data center*

³² <https://aws.amazon.com/pt/ec2/instance-types/>

com 20 MVs, no terceiro um *data center* com 30 MVs, e assim por diante até o décimo ter 100 MVs. A variável de resposta coletada foi o tempo de resposta dos algoritmos para obter uma solução (AZEVEDO, 2016).

Em cada experimento o algoritmo μ AG foi executado 100 vezes, sempre utilizando como critério de parada um tempo limite de 3 minutos de execução ou até chegar ao ótimo (comparado ao algoritmo determinístico).

Com relação aos experimentos relativos a ontologia e consultas SQWRL, eles foram conduzidos explorando algumas das principais características de processamento da ontologia e executando algumas consultas relevantes e já apresentadas como exemplos neste trabalho de doutorado na seção 6.4.2.2. Portanto, relacionado ao tempo de processamento foram coletados os seguintes tempos:

- Tempo de carregamento e validação da ontologia (*LoadValidOntology*): consiste no tempo para que, via programação utilizando a OWLAPI se carregue para a memória a ontologia para que possa ser manipulada adicionado ao tempo de validação que consiste na verificação feita pelo *reasoner* HermiT em busca de inconsistências na ontologia.
- Tempo para criação do motor de consulta SQWRL (*CreateSQWRLQueryEngine*): é o tempo gasto pela fábrica (*factory*) da biblioteca SWRLAPI executar o método *createSQWRLQueryEngine* que recebe a ontologia como parâmetro. O retorno desse método é um objeto *SQWRLQueryEngine* utilizado para a execução das consultas SQWRL.

Os tempos de execução de outras quatro consultas SQWRL também foram coletados e analisados, sendo eles os seguintes:

- Tempo de consulta de um Cliente pelo o seu ID (*QueryClientByID*): engloba o tempo de execução da consulta SQWRL que busca um cliente pelo seu ID. O exemplo desta consulta já foi mencionado na seção 6.4.2.2.
- Tempo de consulta de um Cliente pelo o seu e-mail e senha (*QueryClientByEmailAndPwd*): envolve o tempo de execução da consulta SQWRL que busca um cliente pelo seu e-mail e senha. O exemplo desta consulta também já foi apresentado na seção 6.4.2.2.
- Tempo de consulta por um cliente e por suas VMs que estejam com sobrecarga de CPU (*QueryClientAndVMsCPUOverLoad*): consiste no tempo de execução da consulta SQWRL que retorna o cliente e suas VMs que estão com sobrecarga maior que um determinado valor, no caso dos experimentos este valor foi definido em 90%. A sintaxe dessa consulta é a seguinte:

```

Client(?c) ^ VirtualMachine(?vm) ^ hasId(?c, ?id) ^
swrlb:equal(?id, 1) ^ hasMachine(?c, ?vm) ^
hasCPULoad(?vm, ?cpu) ^ swrlb:greaterThan(?cpu,90)
-> sqwrl:select(?c, ?vm)

```

- Tempo de consulta por um cliente e pelos atributos de QoS estipulados em um SLA que pertence a uma política de gerenciamento de SLA escolhida por esse cliente (***QueryClientAndQoSFromSLA***); contabiliza o tempo gasto para realizar uma consulta SQWRL que retorna o cliente e os atributos de QoS de seu SLA, caso eles se enquadrem nos parâmetros da consulta. Nos experimentos esses parâmetros para consulta foram: Disponibilidade maior que 90 (%), Capacidade maior ou igual 0.5, Custo menor ou igual que 0.1 (unidades monetárias) e Tempo de Resposta menor ou igual 0.5. Essa consulta é semelhante a terceira consulta de exemplo da IaaSOnt apresentada na seção seção 6.4.2.2, com a diferença de ser mais completa, pois envolve todos os atributos de QoS considerados na proposta deste trabalho. A sintaxe dessa consulta é a seguinte:

```

Client(?c) ^ hasManagement(?c, ?man) ^ hasPolicy(?man, ?policy)
^ hasSLA(?policy, ?qos)
^ hasQoSAttribute(?qos, ?qosatAV)
^ hasAvailability(?qosatAV, ?availvalue)
^ hasQoSAttribute(?qos, ?qosatCA)
^ hasCapacity(?qosatCA, ?capvalue)
^ hasQoSAttribute(?qos, ?qosatCO)
^ hasCost(?qosatCO, ?costvalue)
^ hasQoSAttribute(?qos, ?qosatRT)
^ hasResponseTime(?qosatRT, ?rtvalue)
^ swrlb:greaterThan(?availvalue,90)
^ swrlb:greaterThanOrEqual(?capvalue, 0.5)
^ swrlb:lessThanOrEqual(?costvalue, 0.1)
^ swrlb:lessThanOrEqual (?rtvalue, 0.5)
-> sqwrl:select(?c, ?qosatAV, ?qosatCA, ?qosatCO, ?qosatRT)

```

Por fim, também foi coletado o tempo necessário para a inclusão de uma regra via programação, quando for inferida, essa regra visa incluir as VMs que estão sobrecarregadas em uma nova subclasse denominada "*LoadedVM*" agrupando os indivíduos de máquinas virtuais que estão sobrecarregadas. Para os experimentos essa regra (***RuleLoadedVM***) apenas considerou a sobrecarga de CPU (carga de CPU > 90%). A sintaxe dessa regra é a seguinte:

```
VirtualMachine(?vm) ^ hasCPULoad(?vm, ?cpu) ^ swrlb:greaterThan(?cpu,90)
-> LoadedVM(?vm)
```

Os tempos de processamento e validação da ontologia e os tempos de execução das consultas e regra são as variáveis de respostas dadas em segundos. Para esses experimentos foram executadas 30 repetições para cada função com o objetivo de obter uma média para os cálculos do desvio padrão e do intervalo de confiança (95%).

7.3.3 Análise dos Resultados

A Tabela 7 apresenta os resultados de todos os experimentos para o algoritmo determinístico. Nessa tabela observam-se os intervalos de soluções exploradas $[min, max]$, o tempo de execução (resposta) e o número de combinações necessárias para explorar todo o espaço de busca pela melhor solução.

Analisando os resultados do algoritmo determinístico é possível observar que quanto maior a quantidade de máquinas virtuais, maior é o intervalo de busca. Esse comportamento era esperado, pois para o número de máquinas disponíveis, os experimentos consideram que o cliente tem 50% delas alocadas na infraestrutura atual, por exemplo, com 10 MVs disponíveis ao cliente 5 já estão alocadas. A configuração do algoritmo ainda permite variar essa quantidade em 50% para cima ou para baixo, criando o intervalo mínimo ($5-50\% = 2,5 \rightarrow 2$) e máximo ($5+50\% = 7,5 \rightarrow 8$) de $[2, 8]$. Dessa forma, aumentando o número de MVs e consequentemente o intervalo (espaço) de busca também aumenta o número de possíveis combinações de VMs (quantidades de VMs para cada instância: **m3.medium**, **m3.large** e **m3.xlarge**) que compõe uma solução (s, m, l) para a nova infraestrutura que atenda ao SLA.

Tabela 7 – Tempo de resposta e número de combinações obtidas com o algoritmo determinístico para a execução de 10 cenários de 10 até 100 VMs disponíveis (NAKAMURA *et al.*, 2017).

Número de MVs	Intervalo $[min, max]$	Tempo de Execução (minutos)	Número de combinações
10	$[2, 8]$	0,35	63
20	$[5, 15]$	2,19	342
30	$[7, 22]$	6,28	1330
40	$[10, 30]$	14,83	2743
50	$[12, 37]$	27,08	4912
60	$[15, 45]$	47,53	9260
70	$[17, 52]$	72,38	13823
80	$[20, 60]$	108,56	19682
90	$[22, 67]$	150,25	29790
100	$[25, 75]$	213,86	39303

Em um *data center* com 20 MVs disponíveis ao cliente, o tempo de resposta já é consideravelmente alto, atingindo um valor de 2,19 (escala em minutos). Para os cenários

seguintes, com uma quantidade maior de MVs, o tempo de resposta aumenta exponencialmente, até o tempo de resposta ser maior do que 3,5 horas. Esses resultados demonstram que o método determinístico é inviável e impraticável para clientes que possuem quantidades de VMs maiores que 10 instâncias e desejam otimizar as configurações de sua infraestrutura em tempo de execução e sob demanda.

Contudo, os experimentos com o algoritmo determinístico foram relevantes para fins comparativos, uma vez que ele busca por todas as possíveis combinações até encontrar a solução ótima. Uma vez encontrada essa solução ela pode ser comparada com outras soluções sub-ótimas de algoritmos meta-heurísticos.

O algoritmo μ AG foi executado 100 vezes e para todos os cenários ele obteve uma taxa de acerto de 100% em menos de 1 minuto. Os resultados desses experimentos estão na Tabela 8.

Tabela 8 – Tempo médio para alcançar soluções ótimas com o algoritmo μ AG (AZEVEDO, 2016) (NAKAMURA *et al.*, 2017).

Número de VMs	Tempo Médio de Resposta (em minutos)
10	0,13
20	0,54
30	0,19
40	0,32
50	0,56
60	0,17
70	0,14
80	0,17
90	0,19
100	0,16

Analisando os resultados do algoritmo μ AG é possível comprovar a sua eficiência para obter uma solução para o problema.

Uma comparação entre os algoritmos é apresentada na Figura 33. A linha contínua com triângulos são os resultados do tempos de resposta obtidos pelo algoritmo determinístico, já a linha pontilhada com retângulos representa um limite de tempo de três minutos (que dentro do contexto do problema é considerado como sendo um tempo de resposta aceitável para o usuário) e a linha pontilhada com losangos compõe os tempos de resposta das execuções do μ AG.

Analisando a Figura 33 observa-se que foi preciso uma ampliação dos resultados para os quatro primeiros cenários para que fosse possível a comparação. Para o primeiro experimento com 10 MVs os resultados de tempo de resposta para os algoritmos estão bem próximos e ambos ainda respeitam o tempo limite de 3 minutos. Com 20 MVs a distância entre os tempos de resposta aumenta, o determinístico tem um tempo de resposta bem maior que o do μ AG, mas ainda ambos respeitam o limite de tempo. Contudo, a partir do terceiro experimento com 30 MVs o algoritmo determinístico ultrapassa o limite de tempo e tem uma tendência exponencial

Figura 33 – Resultado do Tempo de resposta do modelo determinístico e do μ AG para um cenário com 10 até 100 VMs (Extraído de (NAKAMURA *et al.*, 2017)).

no aumento do tempo de resposta, enquanto o μ AG permanece abaixo do limite de tempo e sem grandes alterações no tempo de resposta.

É relevante esclarecer que os tempos de respostas apresentados pelos algoritmos de otimização consideram apenas a busca por uma solução otimizada de reconfiguração da infraestrutura do cliente visando atender ao SLA. Portanto, esses tempos não englobam a ação de efetivamente executar a reconfiguração, pois isso depende dos mais diversos fatores, desde características de *hardware* dos computadores hospedeiros (*hosts*) e tipo/velocidade da rede até questões de *software* como: tipo e versão do sistema operacional, tipo de *hypervisor*, etc.

Os resultados relacionados aos experimentos que mensuram os tempos de processamento e validação da ontologia, estão apresentados na Tabela 9.

Tabela 9 – Médias dos tempos de execução de funções de processamento da ontologia com web semântica (em segundos).

Funções:	LoadValidOntology	CreateSQWRLQueryEngine
Média (seg):	7,00530	22,50593
Desvio Padrão (seg):	0,37236	0,78985
Int. Confiança (seg):	0,13325	0,28264

O tempo médio da função (*LoadValidOntology*) é menor que tempo de criação do motor de processamento de regras e consultas SQWRL (*CreateSQWRLQueryEngine*). O tempo do carregamento/leitura da ontologia depende da quantidade de informação nela armazenada (número de classes, propriedades, indivíduos, regras, etc.), porém a etapa que demanda tempo é acionar o *reasoner* e aguarda verificação de consistência da ontologia. A criação do motor de processamento de consultas SQWRL consome em torno de três vezes mais. Todo esse tempo

gasto é inerente das implementações das bibliotecas OWLAPI e SWRLAPI respectivamente. Contudo, dentro do contexto geral deste projeto, esses tempos não impactam tanto o desempenho, pois são invocados apenas uma única vez, no início da execução, e os objetos gerados por essas funções ficam armazenados em memória.

Um vez carregada e validada a ontologia e o motor de execução de consultas pronto para ser utilizado, foi possível realizar experimentos com algumas consultas e regras para obter os tempos de execução e comprovar a viabilidade desta tese. Os resultados dos tempos das consultas são apresentados nas Tabelas 10 e 11.

Tabela 10 – Médias dos tempos de execução das consultas SQWRL: *QueryClientByID* e *QueryClientByEmailAndPwd* (em segundos).

Funções:	QueryClientByID	QueryClientByEmailAndPwd
Média (seg):	2,02810	1,17963
Desvio Padrão (seg):	0,25325	0,13700
Int. Confiança (seg):	0,09062	0,04902

Tabela 11 – Médias dos tempos de execução das consultas SQWRL: *QueryClientAndVMsCPUOverLoad* e *QueryClientAndQoSFromSLA* (em segundos).

Funções:	QueryClientAndVMsCPUOverLoad	QueryClientAndQoSFromSLA
Média (seg):	1,37413	1,69910
Desvio Padrão (seg):	0,09852	0,20823
Int. Confiança (seg):	0,03525	0,07451

Analisando os resultados dos tempos médios das consultas SQWRL nota-se que são tempos relativamente baixos e com muito pouco desvio padrão. A Figura 34 exibe um gráfico com um resumo de todos os tempos de processamento e validação da ontologia, criação do motor de consulta SQWRL e das consultas.

Figura 34 – Resultados da avaliação da parte de web semântica e ontologia.

Por fim, na Tabela 12 está o resultado do experimento que verificou o tempo de inclusão de uma regra. A média de tempo para esse resultado foi muito baixa, pois uma vez já criado o motor de execução da SWRLAPI, incluir novas regras não é uma tarefa onerosa para o sistema e pode ser executada em tempo de execução havendo apenas a preocupação de que a regra seja válida e não cause erros ou inconsistências.

Tabela 12 – Médias dos tempos de inclusão da regra SWRL: *RuleLoadedVM* (em segundos).

Funções:	RuleLoadedVM
Média (seg):	0,67667
Desvio Padrão (seg):	0,03237
Int. Confiança (seg):	0,01158

7.4 Considerações Finais

Neste capítulo foram apresentados os resultados de avaliações de desempenho relacionados aos mecanismos de autoconfiguração e auto-otimização. Tais resultados foram apresentados individualmente permitindo analisar características e comportamentos pertinentes.

Nos experimentos da parte de autoconfiguração, a proposta de utilização de monitoração obteve os melhores resultados. A proposta de previsão pode ser interessante para sistemas onde ocorram demanda sazonais com intervalos bem definidos e demandas (carga de trabalho) semelhantes. Contudo, conforme foi apresentado na seção 7.2.3 a utilização de qualquer uma das propostas é melhor do que não utilizá-las.

Nos experimentos da parte de auto-otimização, o algoritmo μ AG permite obter melhores tempos de resposta para otimizar uma nova reconfiguração da infraestrutura do que o algoritmo determinístico. Um resultado relevante é que o μ AG conseguiu uma taxa de 100% na busca pela solução ótima em tempos aceitáveis (inferiores a um minuto). A execução desses algoritmos, que buscam otimizar a infraestrutura visando manter o SLA acordado, demonstra também o emprego da metodologia proposta na seção 6.3.1.

Realizar diversos experimentos que comprovem o emprego das ideias e metodologias propostas neste projeto de doutorado e também o funcionamento e a eficiência de métodos, *frameworks* e algoritmos não é uma tarefa trivial e demandam tempo. Entretanto, a colaboração com outros projetos de pesquisa incluídos e relacionados a este projeto de doutorado permitiu que isso fosse possível, gerando mais contribuições científicas para a área de pesquisa.

Tanto os algoritmos de autoconfiguração quanto os algoritmos de auto-otimização precisam de diversas informações sobre a infraestrutura, atributos e valores de QoS acordados no SLA, qual tipo de análise financeira realizar, etc. Tais informações podem ser extraídas semanticamente da ontologia. Alguns experimentos foram realizados e apresentados neste capítulo com o intuito de comprovar a hipótese e a viabilidade da utilização de recursos semânticos para auxiliar mecanismos baseados em conceitos da computação autônoma.

No próximo capítulo são apresentadas as conclusões deste projeto de doutorado, bem como as suas contribuições e sugestões para trabalhos futuros.

CONCLUSÕES

8.1 Considerações Iniciais

A proposta de desenvolvimento de mecanismos de autoconfiguração e auto-otimização para arquiteturas virtualizadas que visam a provisão de qualidade de serviço vão ao encontro da necessidade de um gerenciamento de recursos computacionais virtualizados dentro do contexto da computação em nuvem. A principal hipótese apresentada nesta tese de doutorado remete à utilização de conceitos da Computação Autônoma (autoconfiguração e auto-otimização) auxiliados por elementos e recursos presentes na área de pesquisa referente à Web Semântica. Trata-se de um tema multidisciplinar cujo o principal objetivo é apresentar uma proposta para automatizar o gerenciamento da infraestrutura virtual visando à garantia de uma qualidade de serviço adequada.

Nesta tese de doutorado foram apresentadas revisões das principais teorias relacionadas ao tema proposto, sendo elas: Web Semântica e Ontologia (capítulo 2), Computação Autônoma (capítulo 3) e Computação em Nuvem (capítulo 4). O capítulo 5 foi destinado à apresentação de uma revisão da literatura que envolveu a pesquisa por trabalhos relacionados às teorias estudadas e que cujo o foco e objetivo proposto tivessem algum tipo de correspondência com a hipótese elencada. Nos capítulos 6 e 7 foram apresentados os mecanismos propostos e a validação dos mesmos, respectivamente, que foram encarregados pela comprovação e validação desta proposta de doutorado.

Este capítulo destina-se à descrição das principais conclusões e dos trabalhos futuros provenientes do desenvolvimento da proposta e do desenvolvimento dos mecanismos de autoconfiguração e auto-otimização. Na seção 8.2 são discutidas as conclusões alcançadas com o desenvolvimento deste projeto de doutorado. Na seção 8.3 são apresentadas as principais contribuições originadas das propostas, elaboração e desenvolvimento do projeto desta tese. Por fim, a seção 8.4 lista alguns trabalhos futuros relacionados com a proposta desta tese.

8.2 Discussão e Conclusões

O advento da Computação em Nuvem proporcionou a provisão de novas aplicações na Web de forma mais eficiente, rápida e escalável. Desde pequenas até grandes empresas podendo reduzir seus gastos com equipamentos, pessoal e manutenção na área de TI, alocando ("alugando") recursos de provedores em nuvem e, assim, podem focar principalmente nos seus ramos de negócios. Contudo, os grandes provedores fornecem sistemas de gerenciamento com funcionalidades genéricas para qualquer tipo de cliente, independente do seu ramo ou necessidade. Geralmente, os princípios utilizados para a realocação (reconfiguração) são feitos de forma manual ou são baseados conforme a demanda observada (sobrecarga). A cobrança geralmente é feita por hora máquina e nenhuma ou poucas restrições podem ser configuradas pelo cliente. Por exemplo, o cliente pode ter alguma restrição de negócio, financeira ou qualitativa, acordada com seus usuários (QoS/SLA), etc.

Nesse ponto, relacionado às questões de fatores não funcionais, como é o caso da garantia de qualidade de serviço prestada, alguns provedores não deixam claro quais são os valores/níveis de parâmetros/atributos de QoS que serão garantidos e nem mencionam quais as ferramentas ou métricas utilizadas para isso (quando existem). Geralmente, são mencionados apenas fatores relacionados à segurança e disponibilidade, ficando a cargo do cliente mensurar outros atributos da infraestrutura contratada.

Além disso, geralmente os provedores de serviço não fornecem um instrumento para o usuário alocar ou liberar recursos de forma otimizada; no atual contexto dos provedores em nuvem, o cliente é responsável por determinar uma infraestrutura mínima inicial e os seus mecanismos irão "escalar para mais", iniciando novas instâncias de máquinas, até que a demanda seja atendida e posteriormente "escalar para menos", finalizando instâncias excedentes, e essa elasticidade geralmente considera apenas a adição e remoção de instâncias de mesma configuração/capacidade (não alteram recursos de CPU, Memória, Disco, etc. e não iniciam instâncias de outras classes).

Para o cliente ter uma infraestrutura ideal e eficiente é preciso uma monitoração adequada e um sistema de reconfiguração de recursos que realize decisões por novas opções otimizadas e que além de suprir a carga/demanda, também leve em consideração as suas necessidades, em especial aquelas relacionadas à parte financeira e de qualidade de serviço. Contudo, construir um sistema deste tipo é um desafio e não se trata de uma tarefa trivial. Por esse motivo, várias pesquisas têm explorado esses assuntos e buscado propor as mais diversas soluções, conforme foi mencionado no capítulo 5, apoiando-se em diversas áreas do conhecimento.

Nesse contexto, este projeto de doutorado, propõe a relação da Web Semântica, Computação Autônoma e Computação em Nuvem, que são temas consolidados, porém que dispõe de diversas pesquisas que almejam a aplicação desses conceitos na resolução de problemas práticos e atuais. Dos trabalhos relacionados pesquisados, poucos envolveram partes das três teorias

dentro de uma única proposta. Dessa forma, foram propostos mecanismos baseados em dois conceitos da Computação Autônoma, um destinado a realizar a reconfiguração da infraestrutura virtualizada de forma automática (autoconfiguração) e outro destinado a otimizar os recursos dessa infraestrutura propondo uma solução eficiente e que atenda aos requisitos do cliente (auto-otimização). Conforme visto no capítulo 3 a *terceira* da Computação Autônoma propõe a execução das atividades de Monitorar, Analisar, Planejar e Executar (MAPE).

As atividades de monitoração e execução são geralmente atividades básicas, realizadas pelos agentes monitor e atuador, que não envolvem nenhuma tomada de decisão e, portanto, o nível de complexidade delas depende da forma como serão implementadas e das características inerentes ao sistema onde atuam. Contudo, a parte de análise e planejamento exigem um nível de complexidade maior, essas atividades que definem a etapa de decisão, ou seja, definem o que deve ser realizado no sistema para se alcançar um funcionamento adequado de forma automática. Neste ponto, enquadra-se a proposta desta tese de aplicar recursos da Web Semântica, mais especificamente o uso de: uma ontologia para representar o conhecimento e recursos de processamento semântico, regras, consultas e motores de inferências para a tomada de decisão.

Portanto, a ontologia *IaaSOnt* foi desenvolvida neste projeto com o intuito de representar a realidade de um ambiente de computação em nuvem cujo o foco está na prestação de infraestruturas como um serviço (IaaS). Essa ontologia também possui classes que visam representar algumas configurações de políticas de gerenciamento em especial ao controle de provisão de qualidade de serviço.

Os mecanismos de autoconfiguração e auto-otimização podem tanto utilizar informações armazenadas ou processadas (resultados de regras, consultas, inferências) na ontologia, quanto também serem acionados por meio de um resultado de decisão, que pode indicar que haja a necessidade de uma otimização e/ou reconfiguração da infraestrutura do cliente.

Baseado nessas propostas tanto os mecanismos baseados na computação autônoma quanto a ontologia e aparato semântico foram desenvolvidos (Capítulo 6). Durante a elaboração dos mecanismos sugeriram possibilidades de colaboração envolvendo projetos de mestrados, responsáveis pela aplicação da metodologia desenvolvida neste projeto de doutorado e pela realização da implementação, comparação e testes. Para a ontologia *IaaSOnt* seguiu-se uma metodologia que demandou um longo período envolvendo: o estudo sobre informações pertinentes a um ambiente em nuvem destinado a IaaS em que houvesse uma forma de gerenciamento de recursos automático conforme previsto na proposta; o planejamento de como deveriam ser criada a hierarquia de classes e subclasses e definir quais as propriedades mais relevantes; e um tempo de desenvolvimento para a criação efetiva da ontologia, bem como a realização de testes de validações e consistência. Outro longo período foi destinado à aprendizagem e utilização de novas sintaxes (SWRL, SQWRL), APIs (SWRLAPI, OWLAPI) e *reasoners*. Por fim, houve um curto intervalo de tempo para a realização de teses e validações das implementações que utilizaram tais linguagens, APIs e ferramentas.

Os resultados individuais dos mecanismos de autoconfiguração e auto-otimização, bem como da parte semântica relacionados a esta tese foram apresentados no capítulo 7. Na parte de autoconfiguração foi proposto e desenvolvido a parte de monitoração e atuação/execução de reconfigurações. Duas abordagens foram exploradas para a realização de uma reconfiguração.

A primeira é uma abordagem reativa, baseada na monitoração ou no conceito "causa-efeito", uma técnica muito comum entre os provedores de serviço, consiste em monitorar os recursos e quando a sua utilização alcançar um determinado patamar uma reconfiguração deve ser executada. As informações ou parâmetros que determinam o(s) patamar(es) podem ser recuperados da ontologia, embora nos experimentos fossem parâmetros fixos, utilizados apenas para validar o mecanismo.

A segunda consiste em uma abordagem pró-ativa, pois por meio de técnicas de previsão ela tenta prever qual será a demanda e, assim, busca-se reconfigurar a infraestrutura antes que a carga afete o desempenho do sistema. Embora a técnica de previsão seja uma abordagem interessante, ela tem dependência por um armazenamento de informações (histórico) para conseguir realizar as previsões e seus resultados dependem do modelo de previsão adotado e do tipo de carga (sazonal, não sazonal) à qual o sistema está exposto. Além disso, a forma que a abordagem de previsão foi implementada e avaliada, ou seja, considerando a previsão apenas para o dia seguinte, envolve um espaço de tempo no qual podem ocorrer diversas variações de carga (dependendo da aplicação a carga pode variar bastante em um dia). Por outro lado, realizar previsões para intervalos de tempo menores pode ser muito oneroso para o sistema.

Dessa forma, nos experimentos realizados a abordagem de monitoração obteve os melhores resultados, ou seja, ela obteve as menores diferenças entre os cálculos de erros do que estava disponível para uso e o que realmente foi utilizado (observado). Assim, quanto mais próximos estavam os níveis de recursos disponíveis do que foi observado/utilizado, melhor foi a reconfiguração, pois gerou um menor desperdício de recursos ociosos.

Posteriormente, foi proposta a elaboração, o desenvolvimento e a realização de testes de avaliação do *framework* FAMaV. A concepção deste *framework* foi elaborada e incentivada por este projeto de doutorado, pois a primeira implementação da parte de autoconfiguração fazia uso de *shell-scripts* para a execução da ferramenta *Virsh* que realmente atuava no gerenciamento e na modificação de recursos de máquinas virtuais com o *hypervisor*. Manter o controle, execução e tratamento de erros com esses *scripts* via programação seria muito complicada, por esse motivo decidiu-se pela criação de um *framework* em Java que realizasse esses procedimentos. Após um estudo de como isso poderia ser feito, foi elaborado um projeto que utilizasse a biblioteca *libvirt* que embora escrita na linguagem C, poderia ser utilizada por meio da utilização da *JAVA API bindings* e *Java Native Access*.

Após a organização e estrutura do projeto FAMaV, bem como a inclusão de bibliotecas/APIs, dependências e a criação de exemplos iniciais de implementação, um projeto de iniciação científica foi estabelecido para implementar os métodos na linguagem Java que tinham

correspondência com as funções das ferramentas de gerenciamento de máquinas virtuais, entre elas, a ferramenta *Virsh*. Para testar, validar e provar o conceito se tal *framework* poderia ser utilizado no propósito deste projeto, alguns experimentos comparando o desempenho da FAMaV-CLI com a ferramenta *Virsh* foram realizados. Como esperado, a FAMaV-CLI obteve tempos de execução maiores que a *Virsh*, esse desempenho inferior era esperado devido a adição de novas camadas com APIs e a utilização da linguagem Java. Porém, os tempos absolutos da FAMaV-CLI provaram ser razoáveis para a sua utilização neste projeto e são justificáveis pelo ganho de produtividade, melhor gerenciamento via programação e melhor integração com os outros mecanismos.

Na proposta para o mecanismo de auto-otimização considera-se como sendo a parte principal a metodologia descrita nesta tese de doutorado para a obtenção de uma nova configuração (solução) de infraestrutura que considere a provisão de qualidade de serviço. Dessa forma, na metodologia proposta neste projeto de doutorado quatro parâmetros de QoS (Capacidade, Disponibilidade, Custo e Tempo de Resposta) são considerados. Como a proposta para realizar a otimização é baseada na composição de máquinas virtuais (como um serviço), que juntas devem atender à demanda prestando um certo nível de qualidade, então foram propostas quatro funções de agregação, uma para cada um dos atributos de QoS.

O atributo Capacidade e sua função de agregação é uma proposta de atributo de QoS original e previsto por este trabalho de doutorado como sendo uma forma de reunir diversas características de uma máquina virtual e um único atributo para otimização. A ideia é que não seja preciso comparar recurso computacional (CPU, Memória, Disco, etc.) por recurso computacional para se determinar se uma instância de máquina virtual é melhor que outra; a Capacidade é um número entre 0 e 1 que define quanto de capacidade computacional tem a MV; quanto mais próximo de 0 menor a sua capacidade e quanto mais próximo de 1 maior é a sua capacidade dentro de um intervalo que represente os tipos de instâncias (ou classes) de MVs dos provedores. Assim, uma instância *m3.medium* da Amazon pode, por exemplo, ter um valor de Capacidade igual a 0.5, uma instância *m3.large* um valor igual 0.75 e *m3.xlarge* igual a 1. Quem define os valores de Capacidade de cada instância é o administrador do provedor ou alguém que tenha conhecimento técnico adequado para criar uma relação proporcional adequada.

Uma vez que foi definida a metodologia, criou-se o processo de otimização que passa a verificar se a atual configuração da infraestrutura está cumprindo com o acordo (SLA). Para isso são realizadas comparações dos atributos de QoS agregados das máquinas virtuais com o que está acordado na política de gerenciamento do cliente.

Os algoritmos de otimização devem buscar por soluções que proponham novas configurações de infraestrutura. Um projeto de mestrado, que foi definido a partir do desenvolvimento desta tese de doutorado, propõe a implementação de diversos algoritmos de otimização com esse propósito e também os compara. Dessa forma, auxiliado pela metodologia deste projeto de doutorado, houve uma colaboração entre os projetos para a implementação desses algoritmos.

Um algoritmo determinístico foi implementado inicialmente para percorrer todo o espaço de busca por uma solução, seu principal objetivo é encontrar a solução ótima e ter o tempo de busca gasto até encontrar essa solução mensurado. Dessa forma, a solução ótima e o tempo gasto para encontrá-la podem ser utilizados para comparações com outras abordagens de algoritmos meta-heurísticos. Os resultados de tempo de busca do algoritmo determinístico foram bem altos: conforme aumentava-se o número de MVs utilizadas (espaço de busca) esse tempo aumentava exponencialmente. Assim, a utilização do algoritmo determinístico é inviável em um ambiente real de produção.

Outro algoritmo implementado no projeto de mestrado e que auxiliou na validação da proposta deste projeto de doutorado, foi o algoritmo *muAG* que depois de testes para o seu refinamento obteve resultados muito melhores que o determinístico, pois nos experimentos realizados, além de obter uma taxa de acerto de 100%, ele ficou bem abaixo do limite máximo de tempo estipulado (3 minutos) para encontrar uma solução; para os cenários analisados o seu tempo médio de resposta sempre estava abaixo de 1 minuto.

Além da utilização da metodologia proposta para determinar se o SLA está ou não sendo cumprido, os algoritmos de otimização estão relacionados com este projeto na parte referente à necessidade dos dados para realizar a otimização. Todos os atributos de QoS descritos fazem parte da ontologia que pode passar, além dos valores destes atributos, a situação da infraestrutura atual, a quantidade de instâncias (MV's) do cliente e outras informações que podem ser consideradas no processamento dos algoritmos como critérios de busca ou restrições. Por fim, informações sobre qual algoritmo de otimização utilizar e quando ele deve ser utilizado também podem estar na ontologia e serem decididos pelos mecanismos semânticos. Por exemplo, um cliente pode optar por não utilizar otimização de recursos baseada no SLA, mas pode optar por uma otimização que vise obter o melhor tempo de resposta (maximizar), porém sem ultrapassar o seu orçamento diário (não se trata de uma minimização do custo, mas sim de uma restrição).

Com base na discussão e nas conclusões apresentadas nesta seção, algumas conclusões gerais podem ser formuladas da seguinte forma:

- A proposta desta tese de doutorado previa a elaboração de mecanismos baseados em conceitos da computação autônoma (autoconfiguração e auto-otimização) para arquiteturas virtualizadas que visam a provisão de qualidade de serviço. Para atender aos requisitos da computação autônoma, especialmente a parte referente à tomada de decisão e representação do conhecimento, a hipótese sustentada por este trabalho era de que recursos da Web Semântica poderiam cumprir esse papel. Por meio da criação da ontologia *IaaSOnt* e do desenvolvimento e execução de recursos da web semântica, tais como: consultas, regras e inferências, pode-se comprovar a hipótese. Portanto, recursos da web semântica podem auxiliar na tomada de decisão dentro do contexto da computação autônoma, porém elaborar a representação do conhecimento (ontologia) é uma tarefa que exige tempo e muito estudo

sobre a área que se deseja representar. Embora os conceitos de web semântica sejam consolidados, muitas das ferramentas e APIs ainda estão em fase de amadurecimento, sendo a maioria delas elaboradas por pesquisadores/desenvolvedores voluntários e ainda existem problemas de incompatibilidade entre algumas delas dependendo de versões utilizadas. Além disso, as APIs utilizadas são desenvolvidas em Java que se por um lado garante uma melhor produtividade no desenvolvimento de código, por outro lado degrada um pouco o desempenho.

- Os mecanismos de autoconfiguração propostos são capazes de reconfigurar infraestruturas virtuais sem a intervenção do administrador. A proposta reativa (monitoração) e pró-ativa (previsão) elaboradas em conjunto e implementadas em um projeto de mestrado (DIAS, 2015), comprovaram que a criação de tal mecanismo é factível para as arquiteturas virtualizadas, como é o caso da computação em nuvem.
- O mecanismo de auto-otimização englobou uma metodologia para a otimização de recursos que visa a provisão de qualidade de serviço. Essa metodologia, proposta nesta tese, foi utilizada em um projeto de mestrado (AZEVEDO, 2016) como proposta para a elaboração de algoritmos de otimização que cumpram com os acordos de níveis de serviço. Além da implementação de diversos algoritmos, o projeto de mestrado também visa compará-los por meio de avaliações de desempenho. Em colaboração com esse projeto de mestrado, a metodologia proposta foi validada uma vez que os algoritmos determinístico e *muAG* foram capazes de determinar reconfigurações que atendessem ao SLA proposto.
- O desenvolvimento de projetos de mestrado e de iniciação científica relacionados com este projeto de doutorado é um ponto interessante a ser destacado, pois contribuiu para a experiência de trabalho em equipe e gerenciamento de projetos, além de comprovar os resultados positivos obtidos a partir deste trabalho de doutorado.
- O objeto principal deste trabalho foi alcançado com o desenvolvimento da ontologia, dos recursos da web semântica e dos protótipos que compõem os mecanismos de auto-otimização e autoconfiguração. Nas avaliações individuais, uma vez configurados executam sem interferência humana e seus resultados são satisfatórios. Contudo, conforme é apresentado na seção 8.4, diversas melhorias ainda podem ser realizadas nesses mecanismos e também futuras avaliações ainda precisam ser executadas para que gerem novas análises e conclusões.

8.3 Contribuições

Durante a execução deste projeto de doutorado foram produzidas diversas contribuições enquanto buscava-se atingir o objetivo e a comprovação da hipótese elencada.

Algumas das principais contribuições que merecem destaque são listadas a seguir:

- **Ontologia IaaSOnt:** A ontologia IaaSOnt é uma contribuição relevante deste projeto, pois representa em detalhes um ambiente de prestação de infraestrutura como um serviço. Diversos elementos pertencentes a este contexto estão representados na IaaSOnt, tais como: Clientes, Provedores, Máquinas (Reais e Virtuais), Protocolos, Estados da Máquinas, etc. Além disso, a IaaSOnt conta também com elementos relacionados à prestação de QoS e ao acordo SLA. Essa ontologia pode ser reutilizada por outras ontologias ou considerada em outros projetos de pesquisa. Por exemplo, pode se estender a IaaSOnt para receber novas classes que representem informações sobre segurança e, dessa forma, torna-se possível utilizá-la para novas propostas relacionadas à segurança em IaaS sem a necessidade da criação de todos os elementos novamente.
- **Proposta de conceitos da computação autônoma no mecanismo de autoconfiguração:** A proposta de criação de um modelo baseado na computação autônoma que propõe a reconfiguração de infraestruturas virtuais é outra contribuição relevante desta tese. A partir deste trabalho outros projetos podem considerar essa proposta para criar novos mecanismos diferentes das propostas de monitoração e previsão discutidas neste trabalho. Assim, baseando-se nesta proposta, novos algoritmos podem ser considerados para a realização de uma reconfiguração automática que não precisam necessariamente estarem relacionados à computação em nuvem, mas sim a outros tipos de recursos dentro dos mais diversos contextos.
- **Framework FAMaV:** Tanto o *framework* FAMaV quanto a ferramenta CLI (FAMaC-CLI) que faz uso deste *framework* são legados deste projeto de doutorado com a colaboração do projeto de iniciação científica. O desenvolvimento de outros projetos e ferramentas podem incluir o FAMaV a fim de utilizar todas as classes e métodos disponíveis para o gerenciamento de recursos computacionais virtualizados. A única restrição é que o *hypervisor* utilizado pela arquitetura virtualizada seja compatível com a biblioteca *libvirt*.
- **Metodologia e mecanismo para otimização de recursos computacionais virtuais baseada em atributos de QoS:** A metodologia proposta para a otimização de recursos visando a provisão adequada de qualidade de serviço é outra contribuição de destaque desta tese. Além de propor a *Capacidade* como um novo atributo de QoS, a metodologia considera a agregação dos atributos de QoS e a realização de uma composição dos recursos (serviços) para que a otimização possa comparar e buscar soluções.
- **Módulo web semântico e de inferência:** Além da ontologia, o projeto criado para carregamento e manipulação da ontologia, com auxílio da OWLAPI, bem como todo o código gerado para a execução de regras, consultas e inferências, com apoio da SWRLAPI e das implementações de *reasoners*, são todas contribuições importantes originados neste projeto de doutorado. Todo esse aparato de *software* que foi desenvolvido poderá ser reutilizado por outros projetos que tenham como objetivo a utilização de recursos da web semântica

pra recuperar informações (inferidas ou não) de uma ontologia que represente algum tipo de conhecimento.

- **Outras propostas de políticas de gerenciamento:** Durante a elaboração da ontologia diversos tipos de políticas de gerenciamento foram propostas (vide seção 6.4.2.1). Apesar de que nem todas essas políticas tenham sido implementadas e avaliadas, as ideias propostas e como seriam os seus funcionamentos foram planejados e discutidos nesta tese, especialmente as políticas (classes e subclasses de *FinancialPolicy*) de controle financeiro.

8.3.1 Produção Científica

Durante o desenvolvimento deste projeto de doutorado, que envolveu a criação da ontologia e dos mecanismos de autoconfiguração e auto-otimização, houve a escrita de artigos científicos relacionados à tese. Os artigos aceitos e publicados estão mencionados a seguir:

- LUIS H. V. NAKAMURA, JULIO C. ESTRELLA, MARCOS J. SANTANA, REGINA H. C. SANTANA, STEPHAN REIFF-MARGANIEC. *A Semantic Approach for Efficient and Customized Management of IaaS Resources*. In: 10th International Conference on Network and Service Management (CNSM 2014). Rio de Janeiro, Brazil, 2014.
- DIAS, A. S. ; NAKAMURA, L. H. V. ; ESTRELLA, J. C. ; SANTANA, REGINA H.C. ; SANTANA, MARCOS J. *Providing IaaS Resources Automatically through Prediction and Monitoring Approaches*. In: THE 19TH IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC 2014), 2014, Madeira, PT. THE 19TH IEEE SYMPOSIUM ON COMPUTERS AND COMMUNICATIONS (ISCC 2014), 2014. v. 19.
- YURI TOLEDO NEVES, LUIS H. V. NAKAMURA, PEDRO F. DO PRADO, MARCOS J. SANTANA. *FAMaV: Análise Comparativa entre Ferramentas de Gerenciamento de Máquinas Virtuais*. In: XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-WIC, 2014). São José dos Campos, São Paulo, Brasil, 2014.

Além dos artigos anteriormente mencionados, novos artigos relacionados a esta tese de doutorado foram submetidos e estão em fase de revisão.

O autor deste projeto também colaborou com outros projetos dentro do grupo de pesquisa que abordaram assuntos similares ou com alguma relação dentro da mesma área de pesquisa. Essas colaborações geraram alguns artigos que embora não tenham ligação direta com esta tese, contribuíram para a produção científica da área. A lista desses artigos é a seguinte:

- NAKAMURA, LUIS H. V.; PRADO, P. F. ; LIBARDI, R. M. O. ; NUNES, L. H. ; MENEGUETTE, R. I. ; ESTRELLA, JULIO C. ; SANTANA, REGINA H. C. ; SANTANA, MARCOS J. ; REIFF-MARGANIEC, S. . **Efficient Web Services Selection based on**

- QoS through a Distributed Parallel Semantic Approach.** In: 7th IFIP International Conference on New Technologies, Mobility and Security (NTMS' 2015), 2015, Paris. Proceedings of the 7th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2015. v. 7. p. 1-5.
- PRADO, P. F. ; NAKAMURA, LUIS H. V. ; SANTANA, MARCOS J. ; SANTANA, REGINA H. C. ; CORTES, O. A. C. . **COBAPAS: Combinatorial Optimization based Approach for Autonomic Systems.** In: International Conference on Autonomic and Autonomous Systems (ICAS 2015), 2015, Rome. Proceedings of The Eleventh International Conference on Autonomic and Autonomous Systems (ICAS 2015), 2015. p. 51-56.
 - BATISTA, BRUNO GUAZZELLI2015BATISTA, BRUNO GUAZZELLI ; ESTRELLA, JULIO CEZAR ; FERREIRA, CARLOS HENRIQUE GOMES ; FILHO, DIONISIO MACHADO LEITE ; NAKAMURA, LUIS HIDEO VASCONCELOS ; REIFF-MARGANIEC, STEPHAN ; SANTANA, MARCOS JOSÉ ; SANTANA, REGINA HELENA CARLUCCI . **Performance Evaluation of Resource Management in Cloud Computing Environments.** Plos One, v. 10, p. e0141914, 2015.

8.3.2 Colaboração em projetos de pesquisa

A seguir são listados alguns dos projetos de pesquisa relacionados e que tiveram colaboração direta com este trabalho de doutorado:

- *Computação em nuvem elástica auxiliada por agentes computacionais e baseada em histórico para web services* (Mestrado - Bolsa Fapesp - Processo No. 12/10777-0).
- *FAMaV – Ferramenta de Administração de Máquinas Virtuais: Uma comparação entre ferramentas de gerenciamento* (Iniciação Científica - Bolsa Santander Universidades).
- *Desenvolvimento e avaliação de algoritmos de otimização para o cumprimento de acordos de níveis de serviços em nuvem* (Mestrado - Bolsa Fapesp - Processo No. 15/11623-4).
- *Desenvolvimento de um Dashboard para o gerenciamento baseado em ontologia de uma infraestrutura como um serviço na nuvem* (Iniciação Científica - Bolsa Fapesp - Processo No. 15/26504-0).

8.4 Trabalhos Futuros

Para a concepção da proposta e desenvolvimento dos mecanismos de autoconfiguração e auto-otimização baseados nos conceitos da computação autônoma e auxiliados por recursos da web semântica foi preciso o estudo e aprendizado de metodologias, tecnologias, linguagens e conceitos. Diversas são as dificuldades para a criação de tais mecanismos, pois envolvem

conceitos que para serem implementados exigem complexidades de desenvolvimento. Assim, não foi uma tarefa trivial a criação da ontologia, do *software* que lida com ela e com os recursos da web semântica. Da mesma forma, não foram tarefas triviais: a criação dos protótipos dos mecanismos e a elaboração das propostas em conjuntos com as metodologias discutidas nesta tese.

Dessa forma, devido à complexidade envolvida na proposta deste projeto de doutorado aliada a questões de tempo e somadas a diversas outras atividades acadêmicas que foram realizadas durante o período de doutoramento, algumas melhorias são elencadas para trabalhos futuros.

A seguir estão listadas as principais melhorias para trabalhos futuros:

- Realizar uma avaliação de desempenho considerando a utilização de ambos os mecanismos propostos em um ambiente mais próximo do real. Os mecanismos de autoconfiguração e auto-otimização foram avaliados de forma individual e isolada. O mecanismo de auto-configuração considerou poucos recursos computacionais e a avaliação dos algoritmos do mecanismo de auto-otimização foi realizada em um simulador. Além disso, os mecanismos também podem ser adaptados para trabalharem com *containers* de serviços ao invés de apenas com instâncias de máquinas virtuais.
- Implementação e avaliação das políticas de gerenciamento. Embora nesta tese, mais especificamente quando foi apresentada a proposta da ontologia, estejam previstas e descritas as propostas e o funcionamento de diversas políticas de gerenciamento (controle financeiro, energético, escalonamento, etc.), nem todas foram implementadas. Cada política deve estar relacionada a um algoritmo que a execute, dessa forma, novos algoritmos devem ser implementados e avaliados. É relevante lembrar que para a correta execução da política é preciso uma configuração adequada na ontologia.
- Criar uma ferramenta de controle (*Dashboard*) para as configurações iniciais e visualização da infraestrutura e ontologia. Embora os mecanismos possam funcionar com o mínimo de intervenção humana (de forma autônoma), diversas configurações, inclusão de informações, parâmetros, criação de regras e consultas na ontologia, alguma preparação inicial da infraestrutura, etc. devem ser realizadas. Boa parte dessas atividades, foi feita de forma manual utilizando diversas ferramentas diferentes. Por exemplo, para criar, visualizar, configurar e gerenciar inicialmente as máquinas virtuais utilizou-se uma ferramenta denominada VirtManager³³ (em um provedor na nuvem utiliza-se um *Dashboard*), para adicionar informações na ontologia utilizou-se o Protégé, para criar regras/consultas o *plugin* SWRLTab, etc. Dessa forma, há um projeto de iniciação científica que visa a criação de um *Dashboard* que permita ao cliente criar a sua infraestrutura virtual inicial

³³ <<https://virt-manager.org/>>

e posteriormente obtenha informações da mesma. Tais informações podem ser salvas na ontologia e em um banco de dados. Esse *Dashboard* também deverá permitir ao cliente configurar informações, parâmetros, regras e consultas presentes na ontologia. A criação de regras e consultas na ontologia para que supram as necessidades das políticas e algoritmos de gerenciamento também está prevista por meio deste *Dashboard* em um perfil administrativo.

- Contemplar os conceitos de Auto-Cura e Auto-Proteção. Neste projeto de doutorado foram considerados dois conceitos da computação autônoma: autoconfiguração e auto-otimização. Outros projetos de doutorado podem contemplar os conceitos de auto-cura e auto-proteção, criando novos mecanismos que irão garantir tolerância a falhas e segurança para as arquiteturas virtualizadas. A auto-cura pode tratar problemas, mal funcionamento ou outros tipos de erros que ocorram com os recursos computacionais ou mesmo a falta deles. A auto-proteção deve se preocupar com a proteção dos recursos de forma automática, problemas com *softwares* maliciosos, infecções por vírus, ataques dos tipos: *Denial of Service* (DoS) ou *Distributed Denial of Service* (DDoS), etc. A ontologia proposta nesta tese pode ser reutilizada sendo necessária a criação de alguns elementos referentes ao contexto e realidade desses conceitos da computação autônoma.

Pretende-se realizar e acompanhar os três primeiros itens desta lista de trabalhos futuros em uma proposta de projeto de pós-doutorado. Além disso, novas melhorias ainda poderão ser incluídas a medida em que novas propostas de funcionalidades sejam agregadas aos mecanismos. Por fim, considera-se importante para trabalhos futuros que artigos científicos sejam elaborados e submetidos para conferências e periódicos de qualidade internacional específicos da área de pesquisa, afim de divulgar as novas ideias e resultados.

REFERÊNCIAS

ABDELMABOUD, A.; JAWAWI, D. N.; GHANI, I.; ELSAFI, A.; KITCHENHAM, B. Quality of service approaches in cloud computing: A systematic mapping study. **Journal of Systems and Software**, v. 101, p. 159 – 179, 2015. ISSN 0164-1212. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121214002830>. Citado na página 74.

AKLOUF, Y.; REZIG, E. An ontological approach for dynamic functionality-based web services discovery using expert systems. In: **Applications of Digital Information and Web Technologies, 2009. ICADIWT '09. Second International Conference on the**. [S.l.: s.n.], 2009. p. 187 –192. Citado na página 38.

AMAZON. **Dynamic Scaling**. 2015. Disponível em: <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.html>. Last Access: 01/06/2015. Citado na página 94.

_____. **Scaling Based on Metrics**. 2015. Available at: <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/policy-creating.html>. Last Access: 11/05/2015. Citado na página 94.

_____. **Scaling the Size of Your Auto Scaling Group**. 2015. Disponível em: <http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/scaling-plan.html>. Last Access: 21/05/2015. Citado na página 94.

AMAZON-EC2. **Definição de preço do Amazon EC2**. 2015. Disponível em: <http://aws.amazon.com/pt/ec2/pricing>. Acesso em: 03/08/2015. Citado na página 106.

_____. **Opções de compra de instâncias do Amazon EC2**. 2015. Disponível em: <http://aws.amazon.com/pt/ec2/purchasing-options/>. Acesso em: 03/08/2015. Citado na página 106.

AMORETTI, M.; ZANICHELLI, F.; CONTE, G. Efficient autonomic cloud computing using online discrete event simulation. **Journal of Parallel and Distributed Computing**, v. 73, n. 6, p. 767 – 776, 2013. ISSN 0743-7315. Citado na página 80.

ANBARASU, A. K.; BATY, J.; BENNETT, S.; WILKINS, M.; MATTOON, S.; CHAPPELLE, D.; HENSLE, B.; BOOTH, C.; COOKSON, M.; STERIADIS, C.; GUPTA, D.; SU, H.; AJANI, T.; HO, V.; JAIN, A.; BAYER, J. **Oracle® Reference Architecture - Cloud Infrastructure - Release 3.0**. 2011. Disponível em: <http://www.oracle.com/technetwork/topics/entarch/oracle-ra-cloud-infrastructure-r3-0-1395892.pdf>. Último acesso: 04/04/2013. Citado 2 vezes nas páginas 53 e 66.

ANGHEL, I.; CIOARA, T.; SALOMIE, I.; COPIL, G.; MOLDOVAN, D. An autonomic algorithm for energy efficiency in service centers. In: **Intelligent Computer Communication and Processing (ICCP), 2010 IEEE International Conference on**. [S.l.: s.n.], 2010. p. 281 –288. Citado 3 vezes nas páginas 63, 75 e 82.

ANTONIOU, G.; HARMELEN F., V. **Semantic Web Primer, 2nd Edition**. [S.l.]: MIT Press, 2008. Citado 4 vezes nas páginas 15, 35, 36 e 38.

ANTONOPOULOS, N.; GILLAM, L. **Cloud Computing: Principles, Systems and Applications**. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 1849962405, 9781849962407. Citado 3 vezes nas páginas 15, 19 e 61.

ARMBRUST, M.; FOX, A.; GRIFFITH, R.; JOSEPH, A. D.; KATZ, R.; KONWINSKI, A.; LEE, G.; PATTERSON, D.; RABKIN, A.; STOICA, I.; ZAHARIA, M. A view of cloud computing. **Commun. ACM**, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Citado 2 vezes nas páginas 15 e 66.

AZEVEDO, L. J. de Melo de. **Desenvolvimento e avaliação de algoritmos de otimização para o cumprimento de acordos de níveis de serviços em nuvem**. Tese (Qualificação de Mestrado) — ICMC-USP, São Carlos, SP, 2016. Citado 7 vezes nas páginas 19, 98, 100, 127, 128, 131 e 141.

BAADER, F.; CALVANESE, D.; MCGUINNESS, D. L.; NARDI, D.; PATEL-SCHNEIDER, P. F. (Ed.). **The description logic handbook: theory, implementation, and applications**. New York, NY, USA: Cambridge University Press, 2003. ISBN 0-521-78176-0. Citado na página 37.

BEHRENDT, M.; GLASNER, B.; KOPP, P.; DIECKMANN, R.; BREITER, G.; PAPPE, S.; KREGER, H.; ARSANJANI, A. **Cloud Computing Reference Architecture v2.0**. 2011. Disponível em: <<https://collaboration.opengroup.org/cloudcomputing/documents/23840/CCRA.IBMSubmission.02282011.doc>>. Último acesso: 12/04/2013. Citado 2 vezes nas páginas 52 e 66.

BELOGLAZOV, A.; ABAWAJY, J.; BUYYA, R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. **Future Generation Computer Systems**, v. 28, n. 5, p. 755 – 768, 2012. ISSN 0167-739X. <ce:title>Special Section: Energy efficiency in large-scale distributed systems</ce:title>. Citado na página 72.

_____. _____. **Future Generation Computer Systems**, v. 28, n. 5, p. 755 – 768, 2012. ISSN 0167-739X. Special Section: Energy efficiency in large-scale distributed systems. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X11000689>>. Citado na página 95.

BENEDICT, S. Energy-aware performance analysis methodologies for {HPC} architectures - an exploratory study. **Journal of Network and Computer Applications**, v. 35, n. 6, p. 1709 – 1719, 2012. ISSN 1084-8045. Citado na página 72.

BERNABE, J. B.; PEREZ, J. M. M.; CALERO, J. M. A.; CLEMENTE, F. J. G.; PEREZ, G. M.; SKARMETA, A. F. G. Semantic-aware multi-tenancy authorization system for cloud architectures. **Future Generation Computer Systems**, p. –, 2012. ISSN 0167-739X. Citado na página 77.

BERNERS-LEE, T. **Semantic web on xml**. 2000. Disponível em: <<http://www.w3.org/2000/Talks/1206-xml2k-tbl/>> Último acesso em 14/12/2009. Citado 3 vezes nas páginas 15, 34 e 35.

BERNERS-LEE, T.; FIELDING, R.; IRVINE, U. C.; MASINTER, L. **Uniform Resource Identifiers (URI): Generic Syntax**. 1998. Disponível em: <<http://www.ietf.org/rfc/rfc2396.txt>> Último acesso em 27/01/2010. Citado na página 35.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. **Scientific American**, v. 284, p. 34–43, 2001. Citado 2 vezes nas páginas 33 e 35.

BLACK, P. E. Manhattan distance. **Dictionary of Algorithms and Data Structures**, US National Institute of Standards and Technology. Retrieved May, v. 18, p. 2012, 2006. Citado na página 98.

BOHN, R.; MESSINA, J.; LIU, F.; TONG, J.; MAO, J. Nist cloud computing reference architecture. In: **Services (SERVICES), 2011 IEEE World Congress on**. [S.l.: s.n.], 2011. p. 594–596. Citado na página 67.

BOSSCHE, R. V. den; VANMECHELEN, K.; BROECKHOVE, J. Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. **Future Generation Computer Systems**, v. 29, n. 4, p. 973 – 985, 2013. ISSN 0167-739X. Special Section: Utility and Cloud Computing. Citado 2 vezes nas páginas 58 e 60.

BRAGA, R. **Fundamentos e técnicas de administração financeira**. Atlas, 1989. ISBN 9788522404223. Disponível em: <<http://books.google.com.br/books?id=S3BwPgAACAAJ>>. Citado na página 90.

BRANDIC, I. Towards self-manageable cloud services. In: **Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International**. [S.l.: s.n.], 2009. v. 2, p. 128–133. ISSN 0730-3157. Citado na página 79.

BRIGHAM, E.; HOUSTON, J. **FUNDAMENTOS DA MODERNA ADMINISTRAÇÃO FINANCEIRA**. CAMPUS, 1999. ISBN 9788535203929. Disponível em: <<http://books.google.com.br/books?id=hVAWAAAACAAJ>>. Citado na página 90.

BROTO, L.; HAGIMONT, D.; STOLF, P.; DEPALMA, N.; TEMATE, S. Autonomic management policy specification in tune. In: **Proceedings of the 2008 ACM symposium on Applied computing**. New York, NY, USA: ACM, 2008. (SAC '08), p. 1658–1663. ISBN 978-1-59593-753-7. Citado na página 79.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. **Cloud Computing Principles and Paradigms**. [S.l.]: Wiley Publishing, 2011. ISBN 9780470887998. Citado 6 vezes nas páginas 15, 55, 57, 63, 64 e 74.

BUYYA, R.; CALHEIROS, R.; LI, X. Autonomic cloud computing: Open challenges and architectural elements. In: **Emerging Applications of Information Technology (EAIT), 2012 Third International Conference on**. [S.l.: s.n.], 2012. p. 3–10. Citado 3 vezes nas páginas 74, 75 e 79.

BUYYA, R.; YEO, C. S.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, v. 25, n. 6, p. 599 – 616, 2009. ISSN 0167-739X. Citado 2 vezes nas páginas 53 e 74.

BYUN, E.-K.; KEE, Y.-S.; KIM, J.-S.; MAENG, S. Cost optimized provisioning of elastic resources for application workflows. **Future Generation Computer Systems**, v. 27, n. 8, p. 1011 – 1026, 2011. ISSN 0167-739X. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X11000744>>. Citado na página 95.

CAPRARESCU, B.; PETCU, D. A self-organizing feedback loop for autonomic computing. In: **Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, 2009. COMPUTATIONWORLD '09. Computation World**. [S.l.: s.n.], 2009. p. 126 – 131. Citado na página 75.

CARON, E.; RODERO-MERINO, L.; DESPREZ, F.; MURESAN, A. **Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds**. [S.l.], 2012. 27 p. Citado 3 vezes nas páginas 62, 63 e 87.

CENTURION, A. M. **Impacto das rajadas no desempenho de serviços executados em ambientes em nuvens**. Tese (Tese de doutorado) — ICMC - USP - Universidade de São Paulo, São Carlos, SP, 2015. Citado na página 100.

CHANG, Y.-S.; JUANG, T.-Y.; CHANG, C.-H.; YEN, J.-S. Integrating intelligent agent and ontology for services discovery on cloud environment. In: **Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on**. [S.l.: s.n.], 2012. p. 3215–3220. Citado na página 38.

CHANTRY, D. **Mapping Applications to the Cloud**. 2009. Disponível em: <<http://msdn.microsoft.com/en-us/library/dd430340.aspx>>. Último acesso: 01/08/2013. Citado 2 vezes nas páginas 15 e 61.

CHARLTON, P.; MAGOULAS, G. Autonomic computing and ontologies to enable context-aware learning design. In: **Tools with Artificial Intelligence (ICTAI), 2010 22nd IEEE International Conference on**. [S.l.: s.n.], 2010. v. 2, p. 286–291. ISSN 1082-3409. Citado na página 46.

CHEN, H.; WU, Z.; CUDRE-MAUROUX, P. Semantic web meets computational intelligence: State of the art and perspectives [review article]. **Computational Intelligence Magazine, IEEE**, v. 7, n. 2, p. 67–74, 2012. ISSN 1556-603X. Citado 2 vezes nas páginas 36 e 37.

CHENG, Y.; LEON-GARCIA, A.; FOSTER, I. Toward an autonomic service management framework: A holistic vision of soa, aon, and autonomic computing. **Communications Magazine, IEEE**, v. 46, n. 5, p. 138–146, may 2008. ISSN 0163-6804. Citado 3 vezes nas páginas 15, 43 e 44.

CIURANA, E. **Developing with Google App Engine**. 1. ed. Berkely, CA, USA: Apress, 2009. ISBN 1430218312, 9781430218319. Citado na página 56.

CORRAR, L. J.; THEÓPHILO, C. R. **Pesquisa operacional para decisão em contabilidade e administração: contabilometria**. [S.l.]: Atlas, 2004. Citado 2 vezes nas páginas 88 e 89.

COULOURIS, G.; DOLLIMORE, J.; T., K. **Sistemas Distribuídos: Conceitos e Projetos. 4a. edição**. [S.l.]: Bookmann, 2007. Citado na página 62.

CUNSOLO, V.; DISTEFANO, S.; PULIAFITO, A.; SCARPA, M. Volunteer computing and desktop cloud: The cloud@home paradigm. In: **Network Computing and Applications, 2009. NCA 2009. Eighth IEEE International Symposium on**. [S.l.: s.n.], 2009. p. 134–139. Citado na página 51.

DASTJERDI, A. V.; TABATABAEI, S. G. H.; BUYYA, R. A dependency-aware ontology-based approach for deploying service level agreement monitoring services in cloud. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 42, n. 4, p. 501–518, abr. 2012. ISSN 0038-0644. Citado na página 76.

DAUTOV, R.; KOURTESIS, D.; PARASKAKIS, I.; STANNETT, M. Addressing self-management in cloud platforms: a semantic sensor web approach. In: **Proceedings of the 2013 international workshop on Hot topics in cloud services**. New York, NY, USA: ACM, 2013. (HotTopsCS '13), p. 11–18. ISBN 978-1-4503-2051-1. Citado na página 83.

DIAS, A. da S.; NAKAMURA, L. H. V.; ESTRELLA, J. C.; SANTANA, R. H. C.; SANTANA, M. J. Providing iaas resources automatically through prediction and monitoring approaches. In: **2014 IEEE Symposium on Computers and Communications (ISCC)**. [S.l.: s.n.], 2014. p. 1–7. ISSN 1530-1346. Citado 11 vezes nas páginas 16, 89, 90, 91, 116, 117, 119, 120, 121, 122 e 123.

DIAS, A. S. **Computação em nuvem elástica auxiliada por agentes computacionais e baseada em histórico para web services**. Tese (Dissertação de Mestrado) — ICMC - USP - Universidade de São Paulo, São Carlos, SP, 2015. Citado 8 vezes nas páginas 15, 87, 88, 89, 90, 116, 118 e 141.

DILLON, T.; WU, C.; CHANG, E. Cloud computing: Issues and challenges. In: **Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on**. [S.l.: s.n.], 2010. p. 27–33. ISSN 1550-445X. Citado 3 vezes nas páginas 58, 69 e 74.

DOBSON, S.; STERRITT, R.; NIXON, P.; HINCHEY, M. Fulfilling the vision of autonomic computing. **Computer**, v. 43, n. 1, p. 35–41, jan. 2010. ISSN 0018-9162. Citado 4 vezes nas páginas 29, 41, 42 e 46.

DUKARIC, R.; JURIC, M. B. Towards a unified taxonomy and architecture of cloud frameworks. **Future Generation Computer Systems**, v. 29, n. 5, p. 1196–1210, 2013. ISSN 0167-739X. Special section: Hybrid Cloud Computing. Citado 4 vezes nas páginas 54, 66, 71 e 101.

EGWUTUOHA, I.; CHEN, S.; LEVY, D.; SELIC, B.; CALVO, R. A proactive fault tolerance approach to high performance computing (hpc) in the cloud. In: **Cloud and Green Computing (CGC), 2012 Second International Conference on**. [S.l.: s.n.], 2012. p. 268–273. Citado na página 52.

EIBEN, A.; SMITH, J. **Introduction to Evolutionary Computing**. Springer Berlin Heidelberg, 2007. (Natural Computing Series). ISBN 9783540401841. Disponível em: <<https://books.google.com.br/books?id=7IOE5VIpFpwC>>. Citado na página 99.

EIBEN, A. E.; SMITH, J. E. **Introduction to evolutionary computing**. [S.l.]: Springer, 2003. v. 53. Citado na página 99.

EJARQUE, J.; ALVAREZ, J.; SIRVENT, R.; BADIA, R. A rule-based approach for infrastructure providers' interoperability. In: **Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on**. [S.l.: s.n.], 2011. p. 272–279. Citado na página 76.

EMEAKAROHA, V.; BRANDIC, I.; MAURER, M.; DUSTDAR, S. Low level metrics to high level slas - lom2his framework: Bridging the gap between monitored metrics and sla parameters in cloud environments. In: **High Performance Computing and Simulation (HPCS), 2010 International Conference on**. [S.l.: s.n.], 2010. p. 48–54. Citado na página 79.

EMEAKAROHA, V. C.; NETTO, M. A. S.; CALHEIROS, R. N.; BRANDIC, I.; BUYYA, R.; ROSE, C. A. F. D. Towards autonomic detection of sla violations in cloud infrastructures. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 28, n. 7, p. 1017–1029, jul. 2012. ISSN 0167-739X. Citado na página 79.

ETCHEVERS, X.; COUPAYE, T.; BOYER, F.; PALMA, N. D.; SALAUN, G. Automated configuration of legacy applications in the cloud. In: **Utility and Cloud Computing (UCC)**,

2011 Fourth IEEE International Conference on. [S.l.: s.n.], 2011. p. 170–177. Citado na página 63.

EVOY, G. V. M.; SCHULZE, B. Using clouds to address grid limitations. In: **Proceedings of the 6th international workshop on Middleware for grid computing**. New York, NY, USA: ACM, 2008. (MGC '08), p. 11:1–11:6. ISBN 978-1-60558-365-5. Citado na página 49.

FAMAEY, J.; LATRE, S.; STRASSNER, J.; TURCK, F. D. A hierarchical context dissemination framework for managing federated clouds. **Communications and Networks, Journal of**, v. 13, n. 6, p. 567–582, 2011. ISSN 1229-2370. Citado 2 vezes nas páginas 82 e 83.

FELLER, E.; RILLING, L.; MORIN, C. Energy-aware ant colony based workload placement in clouds. In: **Grid Computing (GRID), 2011 12th IEEE/ACM International Conference on.** [S.l.: s.n.], 2011. p. 26–33. ISSN 1550-5510. Citado na página 72.

FERNÁNDEZ, M. A.; GÓMEZ-PÉREZ, A.; JURISTO, N. Methontology: From ontological art towards ontological engineering. **Proceedings of the AAI Spring Symposium Series**, p. 33–40, 1997. Citado 2 vezes nas páginas 19 e 101.

FOSTER, I.; ZHAO, Y.; RAICU, I.; LU, S. Cloud computing and grid computing 360-degree compared. In: **Grid Computing Environments Workshop, 2008. GCE '08.** [S.l.: s.n.], 2008. p. 1–10. Citado 2 vezes nas páginas 49 e 51.

GANEK, A. G.; CORBI, T. A. The dawning of the autonomic computing era. **IBM Syst. J.**, IBM Corp., Riverton, NJ, USA, v. 42, p. 5–18, January 2003. ISSN 0018-8670. Citado 3 vezes nas páginas 28, 41 e 42.

GARCÍA, J. M.; RUIZ, D.; RUIZ-CORTÉS, A. Improving semantic web services discovery using sparql-based repository filtering. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 17, n. 0, p. 12 – 24, 2012. ISSN 1570-8268. Citado na página 38.

GENNARI, J. H.; MUSEN, M. A.; FERGERTON, R. W.; GROSSO, W. E.; CRUBZY, M.; ERIKSSON, H.; NOY, N. F.; TU, S. W. The evolution of protégé: An environment for knowledge-based systems development. **International Journal of Human-Computer Studies**, 2002. Citado na página 102.

GHANBARI, H.; SIMMONS, B.; LITOIU, M.; ISZLAI, G. Exploring alternative approaches to implement an elasticity policy. In: **Cloud Computing (CLOUD), 2011 IEEE International Conference on.** [S.l.: s.n.], 2011. p. 716–723. ISSN 2159-6182. Citado na página 63.

GITMAN, L. **Principios de Administracao Financeira**. Bookman, 2006. ISBN 9788573077766. Disponível em: <<http://books.google.com.br/books?id=5jjFoDkavoAC>>. Citado na página 90.

GLIMM, B.; HORROCKS, I.; MOTIK, B.; STOILLOS, G.; WANG, Z. Hermit: An owl 2 reasoner. **Journal of Automated Reasoning**, v. 53, n. 3, p. 245–269, 2014. ISSN 1573-0670. Disponível em: <<http://dx.doi.org/10.1007/s10817-014-9305-1>>. Citado na página 111.

GOLDBERG, R. P. Survey of virtual machine research. **Computer**, v. 7, n. 6, p. 34–45, 1974. ISSN 0018-9162. Citado na página 60.

GRAU, B. C.; HORROCKS, I.; MOTIK, B.; PARSIA, B.; PATEL-SCHNEIDER, P.; SATTLER, U. {OWL} 2: The next step for {OWL}. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 6, n. 4, p. 309 – 322, 2008. ISSN 1570-8268. Citado na página 38.

- GUPTA, A.; SAROOD, O.; KALE, L.; MILOJICIC, D. Improving hpc application performance in cloud through dynamic load balancing. In: **Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on**. [S.l.: s.n.], 2013. p. 402–409. Citado na página 52.
- HAASE, P.; MATHÄSS, T.; SCHMIDT, M.; EBERHART, A.; WALTHER, U. Semantic technologies for enterprise cloud management. In: **Proceedings of the 9th international semantic web conference on The semantic web - Volume Part II**. Berlin, Heidelberg: Springer-Verlag, 2010. (ISWC'10), p. 98–113. ISBN 3-642-17748-4, 978-3-642-17748-4. Citado na página 77.
- HAN, R.; GHANEM, M. M.; GUO, L.; GUO, Y.; OSMOND, M. Enabling cost-aware and adaptive elasticity of multi-tier cloud applications. **Future Generation Computer Systems**, p. –, 2012. ISSN 0167-739X. Citado na página 62.
- HASSAN, S.; AL-JUMEILY, D.; HUSSAIN, A. Autonomic computing paradigm to support system's development. In: **Developments in eSystems Engineering (DESE), 2009 Second International Conference on**. [S.l.: s.n.], 2009. p. 273–278. Citado 2 vezes nas páginas 45 e 46.
- HEFLIN, J. D. **Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment**. Tese (Tese de doutorado) — University of Maryland, Maryland - USA, 2001. Citado na página 34.
- HIRANKITTI, V.; XUAN, T. M. A meta-logical approach for reasoning with an owl 2 ontology. In: **Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF International Conference on**. [S.l.: s.n.], 2010. p. 1–6. Citado 2 vezes nas páginas 37 e 38.
- HORROCKS, I.; KUTZ, O.; SATTTLER, U. The even more irresistible sroiq. In: **Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)**. [S.l.: s.n.], 2006. p. 57–67. Citado na página 37.
- HORROCKS, I.; PATEL-SCHNEIDER, P. Reducing {OWL} entailment to description logic satisfiability. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 1, n. 4, p. 345 – 357, 2004. ISSN 1570-8268. Citado na página 37.
- HSU, C.-H.; SLAGTER, K. D.; CHEN, S.-C.; CHUNG, Y.-C. Optimizing energy consumption with task consolidation in clouds. **Information Sciences**, p. –, 2012. ISSN 0020-0255. Citado na página 72.
- HU, L.; YING, S.; JIA, X.; ZHAO, K. Towards an approach of semantic access control for cloud computing. In: JAATUN, M.; ZHAO, G.; RONG, C. (Ed.). **Cloud Computing**. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5931). p. 145–156. ISBN 978-3-642-10664-4. Citado na página 77.
- HUU, T. T.; MONTAGNAT, J. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In: IEEE. **Cluster, Cloud and Grid Computing (CC-Grid), 2010 10th IEEE/ACM International Conference on**. [S.l.], 2010. p. 612–617. Citado na página 94.
- INTEL-CORPORATION. **Cloud computing taxonomy and ecosystem analysis**. 2010. Available at: <http://www.intel.com/content/dam/doc/case-study/intel-it-cloud-computing-taxonomy-ecosystem-analysis-study.pdf>. Last Access: 04/04/2013. Citado 4 vezes nas páginas 15, 56, 57 e 66.

ISLAM, S.; KEUNG, J.; LEE, K.; LIU, A. Empirical prediction models for adaptive resource provisioning in the cloud. **Future Generation Computer Systems**, v. 28, n. 1, p. 155 – 162, 2012. ISSN 0167-739X. Citado na página 65.

ISLAM, S.; LEE, K.; FEKETE, A.; LIU, A. How a consumer can measure elasticity for cloud platforms. In: **Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering**. New York, NY, USA: ACM, 2012. (ICPE '12), p. 85–96. ISBN 978-1-4503-1202-8. Citado na página 62.

ISOTANI, S.; MIZOGUCHI, R.; BITTENCOURT, I. I.; COSTA, E. d. B. Estado da arte em web semântica e web 2.0: Potencialidades e tendências da nova geração de ambientes de ensino na internet. **Revista Brasileira de Informática na Educação**, v. 17, p. 30–42, 2009. Citado na página 36.

ITANI, W.; GHALI, C.; BASSIL, R.; KAYSSI, A.; CHEHAB, A. Servbgp: Bgp-inspired autonomic service routing for multi-provider collaborative architectures in the cloud. **Future Generation Computer Systems**, p. –, 2012. ISSN 0167-739X. Citado na página 80.

JACOB, B.; LANYON-HOGG, R.; NADGIR, D. K.; YASSIN, A. F. **SA Practical Guide to the IBM Autonomic Computing Toolkit**. [S.l.]: IBM Corporation, 2004. Citado 2 vezes nas páginas 15 e 43.

JEFFERY, K.; KYRIAZIS, D.; CASALICCHIO, E.; PALMIRANI, M. 1st international conference on cloud forward: From distributed to complete computing a cloud service broker with legal-rule compliance checking and quality assurance capabilities. **Procedia Computer Science**, v. 68, p. 136 – 150, 2015. ISSN 1877-0509. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1877050915030756>>. Citado na página 84.

JEONG, H.-Y.; HONG, B.-H. A management of resource ontology for cloud computing. In: KIM, T.-h.; ADELI, H.; FANG, W.-c.; VASILAKOS, T.; STOICA, A.; PATRIKAKIS, C.; ZHAO, G.; VILLALBA, J.; XIAO, Y. (Ed.). **Communication and Networking**. [S.l.]: Springer Berlin Heidelberg, 2011, (Communications in Computer and Information Science, v. 266). p. 65–72. ISBN 978-3-642-27200-4. Citado na página 77.

JR., S. O. The problem with cloud-computing standardization. **Computer**, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 44, n. 7, p. 13–16, jul. 2011. ISSN 0018-9162. Citado na página 54.

KALOU, A.; POMONIS, T.; KOUTSOMITROPOULOS, D.; PAPATHEODOROU, T. Intelligent book mashup: Using semantic web ontologies and rules for user personalisation. In: **Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on**. [S.l.: s.n.], 2010. p. 536–541. Citado na página 77.

KANG, J.; SIM, K.-M. Ontology and search engine for cloud computing system. In: **System Science and Engineering (ICSSE), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 276–281. Citado na página 77.

KANG, J.; SIM, K. M. Towards agents and ontology for cloud service discovery. In: **Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 483–490. Citado na página 77.

- KATSAROS, G.; SUBIRATS, J.; FITÓ, J. O.; GUITART, J.; GILET, P.; ESPLING, D. A service framework for energy-aware monitoring and {VM} management in clouds. **Future Generation Computer Systems**, p. –, 2012. ISSN 0167-739X. Citado 2 vezes nas páginas 51 e 72.
- KEAHEY, K.; TSUGAWA, M.; MATSUNAGA, A.; FORTES, J. A. B. Sky computing. **Internet Computing, IEEE**, v. 13, n. 5, p. 43–51, 2009. ISSN 1089-7801. Citado na página 71.
- KEPHART, J.; CHESS, D. The vision of autonomic computing. **Computer**, v. 36, n. 1, p. 41 – 50, jan 2003. Citado 4 vezes nas páginas 15, 28, 43 e 44.
- KERTESZ, A.; KECSKEMETI, G.; BRANDIC, I. An interoperable and self-adaptive approach for sla-based service virtualization in heterogeneous cloud environments. **Future Generation Computer Systems**, p. –, 2012. ISSN 0167-739X. Citado na página 80.
- KHALID, A.; HAYE, M.; KHAN, M.; SHAMAIL, S. Survey of frameworks, architectures and techniques in autonomic computing. In: **Autonomic and Autonomous Systems, 2009. ICAS '09. Fifth International Conference on**. [S.l.: s.n.], 2009. p. 220 –225. Citado 4 vezes nas páginas 29, 42, 75 e 87.
- KO, J. M.; KIM, C. O.; KWON, I.-H. Quality-of-service oriented web service composition algorithm and planning architecture. **Journal of Systems and Software**, Elsevier, v. 81, n. 11, p. 2079–2090, 2008. Citado 2 vezes nas páginas 96 e 97.
- KVM. **Kernel Based Virtual Machine**. 2013. Disponível em: <<http://www.linux-kvm.org/>>. Último acesso: 01/08/2013. Citado na página 61.
- LAILI, Y.; TAO, F.; ZHANG, L.; CHENG, Y.; LUO, Y.; SARKER, B. R. A ranking chaos algorithm for dual scheduling of cloud service and computing resource in private cloud. **Computers in Industry**, v. 64, n. 4, p. 448 – 463, 2013. ISSN 0166-3615. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0166361513000365>>. Citado na página 104.
- LATRÉ, S.; FAMAHEY, J.; STRASSNER, J.; TURCK, F. D. Automated context dissemination for autonomic collaborative networks through semantic subscription filter generation. **Journal of Network and Computer Applications**, p. –, 2013. ISSN 1084-8045. Citado 2 vezes nas páginas 38 e 83.
- L.D., D. B.; KRISHNA, P. V. Honey bee behavior inspired load balancing of tasks in cloud computing environments. **Applied Soft Computing**, v. 13, n. 5, p. 2292 – 2303, 2013. ISSN 1568-4946. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1568494613000446>>. Citado na página 95.
- LEE, S.; SHIN, D. Web service qos in multi-domain. In: **Proceedings of Advanced Communication Technology, - ICACT**. [S.l.: s.n.], 2008. Citado na página 66.
- LINDEN, R. **Algoritmos genéticos: uma importante ferramenta da inteligência computacional**. [S.l.]: Brasport, 2006. Citado na página 163.
- LIU, F.; TONG, J.; MAO, J.; BOHN, R.; MESSINA, J.; BADGER, M.; LEAF, D. **NIST Cloud Computing Reference Architecture**. 2011. Disponível em: <http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909505>. Último acesso: 30/07/2013. Citado 3 vezes nas páginas 15, 66 e 67.

- LIU, T.; LU, T.; WANG, W.; WANG, Q.; LIU, Z.; GU, N.; DING, X. Sdms-o: A service deployment management system for optimization in clouds while guaranteeing users' qos requirements. **Future Generation Computer Systems**, v. 28, n. 7, p. 1100 – 1109, 2012. ISSN 0167-739X. Special section: Quality of Service in Grid and Cloud Computing. Citado na página 59.
- LOUTAS, N.; KAMATERI, E.; TARABANIS, K. A semantic interoperability framework for cloud platform as a service. In: **Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on**. [S.l.: s.n.], 2011. p. 280–287. Citado na página 76.
- LOUTAS, N.; PERISTERAS, V.; BOURAS, T.; KAMATERI, E.; ZEGINIS, D.; TARABANIS, K. Towards a reference architecture for semantically interoperable clouds. In: **Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on**. [S.l.: s.n.], 2010. p. 143–150. Citado na página 75.
- MAKRIDAKIS, S. **Forecasting: Methods and applications**. New York: John Wiley & Sons, 1998. (3rd edition). Citado na página 89.
- MARSTON, S.; LI, Z.; BANDYOPADHYAY, S.; ZHANG, J.; GHALSASI, A. Cloud computing - the business perspective. **Decision Support Systems**, v. 51, n. 1, p. 176 – 189, 2011. ISSN 0167-9236. Citado 3 vezes nas páginas 56, 66 e 68.
- MARTIMIANO, L. A. F. **Sobre a estruturação de informação em sistemas de segurança computacional: o uso de ontologia**. Tese (Tese de doutorado em Ciências de Computação e Matemática Computacional) — ICMC-USP, São Carlos, SP, 2006. Citado 2 vezes nas páginas 34 e 36.
- MASDARI, M.; NABAVI, S. S.; AHMADI, V. An overview of virtual machine placement schemes in cloud computing. **Journal of Network and Computer Applications**, p. – , 2016. ISSN 1084-8045. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1084804516000291>>. Citado na página 95.
- MAURER, M.; BRANDIC, I.; SAKELLARIOU, R. Adaptive resource configuration for cloud infrastructure management. **Future Generation Computer Systems**, v. 29, n. 2, p. 472 – 487, 2013. ISSN 0167-739X. Special section: Recent advances in e-Science. Citado na página 80.
- MAXIMILIEN, E. M.; SINGH, M. P. A framework and ontology for dynamic web services selection. **IEEE Internet Computing**, IEEE, 2004. Citado na página 38.
- MEHROTRA, R.; SRIVASTAVA, S.; BANICESCU, I.; ABDELWAHED, S. Towards an automatic performance management approach for a cloud broker environment using a decomposition–coordination based methodology. **Future Generation Computer Systems**, v. 54, p. 195 – 205, 2016. ISSN 0167-739X. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X15000825>>. Citado na página 81.
- MELL, P.; GRANCE, T. **The NIST Definition of Cloud Computing**. 2012. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>. Acesso em: 04/04/2013. Citado 4 vezes nas páginas 53, 54, 56 e 58.
- MEZMAZ, M.; MELAB, N.; KESSACI, Y.; LEE, Y.; TALBI, E.-G.; ZOMAYA, A.; TUYTTENS, D. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing

systems. **Journal of Parallel and Distributed Computing**, v. 71, n. 11, p. 1497 – 1508, 2011. ISSN 0743-7315. Citado na página 72.

MICROSOFT. **Microsoft Hyper-V Server 2012**. 2012. Disponível em: <<http://www.microsoft.com/en-us/server-cloud/hyper-v-server/default.aspx>>. Último acesso: 01/08/2013. Citado na página 61.

_____. **What is Windows Azure?** 2012. Available at: <http://www.windowsazure.com/en-us/overview/what-is-windows-azure/>. Last Access: 07/29/2013. Citado na página 56.

MODICA, G. D.; TOMARCHIO, O. A semantic model for utility driven discovery of cloud resources. In: **Complex, Intelligent and Software Intensive Systems (CISIS), 2012 Sixth International Conference on**. [S.l.: s.n.], 2012. p. 822–827. Citado na página 76.

MOHAMED, M.; BELAÏD, D.; TATA, S. Extending {OCCI} for autonomic management in the cloud. **Journal of Systems and Software**, p. –, 2016. ISSN 0164-1212. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0164121216000030>>. Citado na página 80.

MURUGESAN, S. Understanding web 2.0. **IT Professional**, v. 9, n. 4, p. 34–41, 2007. Citado na página 27.

NAKAMURA, H. V. **Utilização de Web Semântica para Seleção de Informações de Web Services no Registro UDDI Uma Abordagem com Qualidade de Serviço**. Tese (Dissertação de Mestrado) — ICMC - USP - Universidade de São Paulo, São Carlos, SP, 2012. Citado 5 vezes nas páginas 34, 35, 36, 101 e 110.

NAKAMURA, L. H. V.; AZEVEDO, L. J. M.; BATISTA, R. I. M. B. G.; TOLEDO, J. C. E. C. F. M.; MARGANIEC, S. R.; SANTANA, R. H. C.; SANTANA, M. J. An analysis of optimization algorithms designed to fully comply with sla in cloud computing (aceito para publicação). **IEEE Latin America Transactions**, v. 15, n. 8, p. 1504–1512, Aug 2017. ISSN 1548-0992. Citado 8 vezes nas páginas 16, 19, 96, 97, 98, 130, 131 e 132.

NAKAMURA, L. H. V.; ESTRELLA, J. C.; SANTANA, R. H. C.; SANTANA, M. J.; REIFF-MARGANIEC, S. A semantic approach for efficient and customized management of iaas resources. In: **10th International Conference on Network and Service Management (CNSM) and Workshop**. [S.l.: s.n.], 2014. p. 360–363. ISSN 2165-9605. Citado 2 vezes nas páginas 16 e 112.

NAKAMURA, L. V.; ESTRELLA, J. C.; SANTANA, M. J.; SANTANA, R. H. C. Semantic web and ontology applied to web services discovery with qos. **Proceedings of the 12th Brazilian Symposium on Computer Systems of High Performance (IEEE), 2011, Vitoria, ES, Brasil**, 2011. Citado 3 vezes nas páginas 15, 34 e 38.

NELSON, V.; UMA, V. Semantic based resource provisioning and scheduling in inter-cloud environment. In: **Recent Trends In Information Technology (ICRTIT), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 250–254. Citado na página 76.

NEVES, Y. T.; NAKAMURA, L. H. V.; PRADO, P. F.; SANTANA, M. J. Famav: Análise comparativa entre ferramentas de gerenciamento de máquinas virtuais. In: **WSCAD 2014 - XV Simpósio em Sistemas Computacionais de Alto Desempenho**. [S.l.: s.n.], 2014. v. 15, p. 254–259. Citado 10 vezes nas páginas 15, 16, 19, 92, 93, 117, 118, 124, 125 e 126.

NICOLICIN-GEORGESCU, V.; BENATIER, V.; LEHN, R.; BRIAND, H. Ontology-based autonomic computing for decision support systems management: Shared resources allocation between groups of data warehouses. In: **Communication Theory, Reliability, and Quality of Service (CTRQ), 2010 Third International Conference on**. [S.l.: s.n.], 2010. p. 233 –236. Citado na página 82.

NOY, N. F.; MCGUINNESS, D. L. **Ontology development 101: A guide to create your first ontology**. 2001. Relatório Técnico, Knowledge System Laboratory - Stanford University, 2001. Citado 2 vezes nas páginas 19 e 101.

O’CONNOR, M.; DAS, A. Sqwrl: A query language for owl. In: **Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529**. Aachen, Germany, Germany: CEUR-WS.org, 2009. (OWLED’09), p. 208–215. Disponível em: <<http://dl.acm.org/citation.cfm?id=2890046.2890072>>. Citado na página 110.

OXFORD, I. S. G. U. O. **HermiT OWL Reasoner**. 2015. Disponível em: <<http://www.hermit-reasoner.com/>>. Último acesso: 10/05/2015. Citado na página 38.

PANDEY, S.; VOORSLUYS, W.; NIU, S.; KHANDOKER, A.; BUYYA, R. An autonomic cloud environment for hosting {ECG} data analysis services. **Future Generation Computer Systems**, v. 28, n. 1, p. 147 – 154, 2012. ISSN 0167-739X. Citado na página 63.

PAPAIIOANNOU, I. V.; TSESMETZIS, D. T.; ROUSSAKI, I. G.; ANAGNOSTOU, M. E. A qos ontology language for web-services. **Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA’06)**, 2006. Citado na página 38.

PATIDAR, S.; RANE, D.; JAIN, P. A survey paper on cloud computing. In: **Advanced Computing Communication Technologies (ACCT), 2012 Second International Conference on**. [S.l.: s.n.], 2012. p. 394–398. Citado na página 57.

PRACTICE, P. C. of. **SWRL Language FAQ**. 2016. Disponível em: <<http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ>>. Último acesso: 28/01/2016. Citado na página 38.

PRUD’HOMMEAUX, E.; SEABORNE, A. **SPARQL Query Language for RDF**. 2008. Disponível em: <<http://www.w3.org/TR/rdf-sparql-query/>>. Último acesso: 05/03/2011. Citado na página 38.

QIAN, H.; MILLER, E.; ZHANG, W.; RABINOVICH, M.; WILLS, C. Agility in virtualized utility computing. In: **Virtualization Technology in Distributed Computing (VTDC), 2007 Second International Workshop on**. [S.l.: s.n.], 2007. p. 1–8. Citado na página 51.

QIAN, L.; LUO, Z.; DU, Y.; GUO, L. Cloud computing: An overview. In: JAATUN, M.; ZHAO, G.; RONG, C. (Ed.). **Cloud Computing**. [S.l.]: Springer Berlin Heidelberg, 2009, (Lecture Notes in Computer Science, v. 5931). p. 626–631. ISBN 978-3-642-10664-4. Citado na página 49.

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 54, 58 e 66.

- RIMAL, B. P.; CHOI, E. A service-oriented taxonomical spectrum, cloudy challenges and opportunities of cloud computing. **Int. J. Commun. Syst.**, John Wiley and Sons Ltd., Chichester, UK, v. 25, n. 6, p. 796–819, jun. 2012. ISSN 1074-5351. Citado 6 vezes nas páginas 54, 58, 66, 69, 71 e 74.
- RODERO, I.; JARAMILLO, J.; QUIROZ, A.; PARASHAR, M.; GUIM, F.; POOLE, S. Energy-efficient application-aware online provisioning for virtualized clouds and data centers. In: **Green Computing Conference, 2010 International**. [S.l.: s.n.], 2010. p. 31–45. Citado na página 80.
- SCHROTH, C.; JANNER, T. Web 2.0 and soa: Converging concepts enabling the internet of services. **IT Professional**, v. 9, n. 3, p. 36 –41, may-june 2007. ISSN 1520-9202. Citado na página 27.
- SHEN, L.; WANG, J.; WANG, K.; ZHANG, H. The design of intelligent security defensive software based on autonomic computing. In: **The Design of Intelligent Security Defensive Software Based on Autonomic Computing**. [S.l.: s.n.], 2009. v. 1, p. 489 –491. Citado 2 vezes nas páginas 42 e 43.
- SIRIN, E.; PARSIA, B.; GRAU, B. C.; KALYANPUR, A.; KATZ, Y. Pellet: A practical owl-dl reasoner. **ACM - Journal of Web Semantics**, ACM, 2007. Citado 2 vezes nas páginas 38 e 111.
- SOFTLAYER. **Virtual Servers Real Performers**. 2016. Disponível em: <http://www.softlayer.com/virtual-servers>. Acesso em: 23/12/2016. Citado na página 106.
- SOLOMON, B.; IONESCU, D.; LITOIU, M.; ISZLAI, G. Designing autonomic management systems for cloud computing. In: **Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on**. [S.l.: s.n.], 2010. p. 631–636. Citado 2 vezes nas páginas 63 e 79.
- STATHIS, K. Autonomic computing with self-governed super-agents. In: **Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010 Fourth IEEE International Conference on**. [S.l.: s.n.], 2010. p. 76 –79. Citado na página 75.
- SULEIMAN, B.; SAKR, S.; JEFFERY, R.; LIU, A. On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. **Journal of Internet Services and Applications**, Springer-Verlag, v. 3, n. 2, p. 173–193, 2012. ISSN 1867-4828. Citado na página 65.
- TAKABI, H.; JOSHI, J.; AHN, G.-J. Security and privacy challenges in cloud computing environments. **Security Privacy, IEEE**, v. 8, n. 6, p. 24–31, 2010. ISSN 1540-7993. Citado na página 58.
- TANENBAUM, A. S. **Redes de computadores**. 4th.. ed. [S.l.]: Editora Campus, 2003. Citado na página 27.
- TCHANA, A.; TRAN, G. S.; BROTO, L.; DEPALMA, N.; HAGIMONT, D. Two levels autonomic resource management in virtualized iaas. **Future Generation Computer Systems**, v. 29, n. 6, p. 1319 – 1332, 2013. ISSN 0167-739X. Citado na página 79.
- TRAN, X. V. Ws-qosonto: A qos ontology for web services. In: **In: 2008 IEEE International Symposium on Service-Oriented System Engineering**. [S.l.]: IEEE International Symposium, 2008. Citado na página 38.

- TSARKOV, D.; HORROCKS, I. Fact++ description logic reasoner: system description. In: **Proceedings of the Third international joint conference on Automated Reasoning**. Berlin, Heidelberg: Springer-Verlag, 2006. (IJCAR'06), p. 292–297. ISBN 3-540-37187-7, 978-3-540-37187-8. Citado na página 38.
- VECCHIOLA, C.; PANDEY, S.; BUYYA, R. High-performance cloud computing: A view of scientific applications. In: **Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on**. [S.l.: s.n.], 2009. p. 4–16. Citado na página 52.
- VMWARE. **Virtual Infrastructure Software**. 2013. Disponível em: <<http://www.vmware.com/>>. Último acesso: 01/08/2013. Citado na página 61.
- VOAS, J.; ZHANG, J. Cloud computing: New wine or just a new bottle? **IT Professional**, v. 11, n. 2, p. 15–17, 2009. ISSN 1520-9202. Citado 3 vezes nas páginas 15, 49 e 50.
- W3C. **SWRL: A Semantic Web Rule Language**. 2004. Disponível em: <<https://www.w3.org/Submission/SWRL/>>. Último acesso: 28/01/2016. Citado na página 38.
- _____. **Web Ontology Language Overview**. 2004b. Disponível em: <<http://www.w3.org/TR/owl-features/>> Último acesso: 15/07/2013. Citado 3 vezes nas páginas 33, 34 e 36.
- _____. **Resource Description Framework (RDF): Concepts and Abstract Syntax**. 2004c. Disponível em: <<http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>> Último acesso: 21/04/2009. Citado na página 34.
- _____. **OWL 2 Web Ontology Language - Document Overview (Second Edition)**. 2012a. Disponível em: <<http://www.w3.org/TR/owl2-overview/>> Último acesso: 15/07/2013. Citado 2 vezes nas páginas 36 e 37.
- _____. **OWL 2 Web Ontology Language - New Features and Rationale (Second Edition)**. 2012b. Disponível em: <<http://www.w3.org/TR/2012/REC-owl2-new-features-20121211/>> Último acesso: 15/07/2013. Citado 2 vezes nas páginas 37 e 38.
- WEI, Y.; JUNPENG, C. Ontology management in cloud computing. In: **Computational Intelligence and Natural Computing Proceedings (CINC), 2010 Second International Conference on**. [S.l.: s.n.], 2010. v. 2, p. 57–59. Citado na página 77.
- WSMO, W. G. **Web Service Modeling Ontology**. 2013. Disponível em: <<http://www.wsmo.org/>>. Último acesso: 04/08/2013. Citado na página 76.
- WU, L.; GARG, S.; BUYYA, R. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In: **Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on**. [S.l.: s.n.], 2011. p. 195–204. Citado na página 66.
- XEN. **Xen Project**. 2013. Disponível em: <<http://www.xenproject.org/>>. Último acesso: 01/08/2013. Citado na página 61.
- XU, C.; YANG, J.; LING, X.; WANG, Y.; LI, L. Architecture design for management as a service cloud. In: **Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on**. [S.l.: s.n.], 2013. p. 860–863. Citado na página 77.
- XU, X. From cloud computing to cloud manufacturing. **Robotics and Computer-Integrated Manufacturing**, v. 28, n. 1, p. 75 – 86, 2012. ISSN 0736-5845. Citado na página 54.

YANG, S.-Y. A novel cloud information agent system with web service techniques: Example of an energy-saving multi-agent system. **Expert Systems with Applications**, v. 40, n. 5, p. 1758 – 1785, 2013. ISSN 0957-4174. Citado na página 72.

YEO, C. S.; VENUGOPAL, S.; CHU, X.; BUYYA, R. Autonomic metered pricing for a utility computing service. **Future Generation Computer Systems**, v. 26, n. 8, p. 1368 – 1380, 2010. ISSN 0167-739X. Citado na página 80.

YONGSIRIWIT, K.; ASSY, N.; GAALOUL, W. A semantic framework for configurable business process as a service in the cloud. **Journal of Network and Computer Applications**, v. 59, p. 168 – 184, 2016. ISSN 1084-8045. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1084804515001708>>. Citado na página 78.

YOUSEFF, L.; BUTRICO, M.; SILVA, D. D. Toward a unified ontology of cloud computing. In: **Grid Computing Environments Workshop, 2008. GCE '08**. [S.l.: s.n.], 2008. p. 1–10. Citado na página 77.

YUAN, Q.; LIU, Z.; PENG, J.; WU, X.; LI, J.; HAN, F.; LI, Q.; ZHANG, W.; FAN, X.; KONG, S. A leasing instances based billing model for cloud computing. In: _____. **Advances in Grid and Pervasive Computing: 6th International Conference, GPC 2011, Oulu, Finland, May 11-13, 2011. Proceedings**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 33–41. ISBN 978-3-642-20754-9. Disponível em: <http://dx.doi.org/10.1007/978-3-642-20754-9_5>. Citado na página 106.

ZHANG, M.; RANJAN, R.; HALLER, A.; GEORGAKOPOULOS, D.; MENZEL, M.; NEPAL, S. An ontology-based system for cloud infrastructure services' discovery. In: **Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on**. [S.l.: s.n.], 2012. p. 524–530. Citado 2 vezes nas páginas 76 e 101.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of Internet Services and Applications**, Springer-Verlag, v. 1, n. 1, p. 7–18, 2010. ISSN 1867-4828. Citado 4 vezes nas páginas 54, 69, 70 e 74.

ZHAO, H.; ZHANG, S.; ZHAO, J. Research of using protégé to build ontology. In: **Computer and Information Science (ICIS), 2012 IEEE/ACIS 11th International Conference on**. [S.l.: s.n.], 2012. p. 697–700. Citado na página 109.

ZHAO, Z.; GAO, C.; DUAN, F. A survey on autonomic computing research. In: **Computational Intelligence and Industrial Applications, 2009. PACIIA 2009. Asia-Pacific Conference on**. [S.l.: s.n.], 2009. v. 2, p. 288 –291. Citado 2 vezes nas páginas 28 e 43.

ZHOU, H.; YANG, H.; HUGILL, A. An ontology-based approach to reengineering enterprise software for cloud computing. In: **Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference**. Washington, DC, USA: IEEE Computer Society, 2010. (COMPSAC '10), p. 383–388. ISBN 978-0-7695-4085-6. Citado 2 vezes nas páginas 38 e 77.

ZHOU, M.; ZHANG, R.; ZENG, D.; QIAN, W. Services in the cloud computing era: A survey. In: **Universal Communication Symposium (IUCS), 2010 4th International**. [S.l.: s.n.], 2010. p. 40–46. Citado na página 57.

METODOLOGIA DE "*TUNNING*" DO ALGORITMO μ AG

Os algoritmos genéticos são largamente aplicados em problemas intratáveis, aqueles em que o tempo necessário para solucioná-lo é inaceitável de acordo com as exigências dos usuários. Tais problemas apresentam um limite superior de complexidade polinomial, um exemplo seria o problema do caixeiro viajante (LINDEN, 2006). Contudo, esse tipo de algoritmo possui parâmetros que influenciam diretamente no resultado, tanto no tempo computacional gasto quanto na qualidade da solução. Dessa forma, o ajuste dos parâmetros tornam-se de extrema importância para um bom desempenho do algoritmo.

Assim, visando encontrar os parâmetros mais adequados para o μ AG, foi realizado um processo de aperfeiçoamento ("*tunning*") do algoritmo, baseado no ajuste dos parâmetros, a execução de experimentos e análise dos resultados. Para os experimentos descritos neste projeto foram realizados 100 execuções. Primeiramente foi executado o algoritmo com os seguintes parâmetros: uma população com 20 indivíduos e uma taxa de mutação de 10% e 20%. A Tabela 13 exibe os resultados obtidos.

Tabela 13 – Tempo de resposta e taxa de sucesso para a população com 20 indivíduos.

VMs	10% taxa de mutação		20% taxa de mutação	
	Tempo Médio (minutos)	Taxa de Sucesso (%)	Tempo Médio (minutos)	Taxa de Sucesso (%)
10	2.1	43	1.8	53
20	2.1	40	2.2	42
30	2.5	36	2.4	30
40	2.6	35	2.6	29
50	2.6	28	2.5	32
60	2.4	23	2.4	35
70	2.7	15	2.6	28
80	2.7	13	2.5	26
90	2.8	10	2.6	20
100	2.8	10	2.8	13

Os resultados obtidos com a execução desse primeiro experimento resultaram em uma taxa de acerto consideravelmente baixa. Assim, a população foi novamente ajustada para 10

indivíduos, mantendo a taxa de mutação de 10% e 20%. A Tabela 14 exibe os resultados obtidos com tais parâmetros.

Tabela 14 – Tempo de resposta e taxa de sucesso para a população com 10 indivíduos.

VMs	10% taxa de mutação		20% taxa de mutação	
	Tempo Médio (minutos)	Taxa de Sucesso (%)	Tempo Médio (minutos)	Taxa de Sucesso (%)
10	1.4	85	1.6	85
20	1.6	76	1.6	86
30	1.8	76	1.7	82
40	1.8	62	1.5	78
50	1.7	62	1.6	78
60	1.7	65	1.8	69
70	1.9	67	1.7	73
80	2.2	69	1.6	79
90	2.5	65	1.5	77
100	2.5	59	1.9	74

Os resultados obtidos com a execução do segundo experimento se sobressaíram ao primeiro experimento, porém a taxa de acerto ainda é consideravelmente baixa. Dessa forma, a população foi ajustada para 5 indivíduos, alterando a taxa de mutação para 20% e 50%, pois com uma taxa de mutação de 10% nenhum indivíduo seria mutado. A Tabela 15 exibe os resultados obtidos com tais parâmetros.

Tabela 15 – Tempo de resposta e taxa de sucesso para a população com 5 indivíduos.

VMs	10% taxa de mutação		20% taxa de mutação	
	Tempo Médio (minutos)	Taxa de Sucesso (%)	Tempo Médio (minutos)	Taxa de Sucesso (%)
10	0.58	100	0.13	100
20	0.79	100	0.54	100
30	0.66	98	0.19	100
40	0.78	100	0.32	100
50	0.86	100	0.56	100
60	0.77	95	0.17	100
70	0.99	98	0.14	100
80	0.97	95	0.17	100
90	0.84	97	0.19	100
100	1.74	92	0.16	100

Finalmente, observou-se que com uma população com 5 indivíduos e uma taxa de mutação de 20% o μ AG obteve 100% de acerto e um tempo de resposta consideravelmente baixo. Portanto, esses foram os valores escolhidos para execução desse algoritmo nos experimentos apresentados neste trabalho.

DOCUMENTOS DE ESPECIFICAÇÕES DA ONTOLOGIA

Infrastructure as a Service Ontology – IaaSOnt

Documentos de Especificações da Ontologia

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Cliente (*Client*)

Data: 20 de outubro de 2013.

Última Atualização: 03 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os Clientes (*Clients*) que solicitam os serviços IaaS (Infrastructure as a Service) que são as Máquinas Virtuais – MV ou VM (*Virtual Machines*). Esta ontologia pode ser utilizada para obter informações a respeito dos clientes, por exemplo, o Nome do cliente, o ID do cliente, E-mail do cliente, Senha do cliente e Descrição do cliente.

Nível de Formalidade: Semi-formal.

Escopo: Lista de clientes atendidos pelo *Provedor (Provider)*, o número de clientes é variável. Cada *Cliente* possui um *Provedor* com o qual estabelece um acordo (SLA) que é expresso pelo *Gerenciamento de SLA (SLAManagement)* e que por sua vez é relacionado com uma qualidade de serviço (*QualityOfService*) que contém *Atributos de QoS (QoSAttribute)* exigido pelos clientes. Além do gerenciamento de QoS e do Gerenciamento de Recursos (*ResourceManagement*), que são obrigatórios, o cliente pode optar pelo Gerenciamento Financeiro (*FinancialManagement*) e permitir ou não que o provedor realize um Gerenciamento de Energia (*PowerManagement*) estando ciente que isso poderá prejudicar um pouco o desempenho e disponibilidade da sua infraestrutura, porém ele estará colaborando com uma menor emissão de gás carbônico (*Green Computing*). O *Cliente* possui *Usuários (Users)* que consomem os serviços provisionados nas MVs do *Cliente*.

Lista de conceitos: Cliente (*Client*), Gerenciamento (*Management*: [*Financial*, *Power*, *Resource* e *SLA*]), Provedor (*Provider*), Máquina Virtual (*VirtualMachine*) e Usuário (*User*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Provedor (*Provider*)

Data: 20 de outubro de 2013.

Última Atualização: 20 de outubro de 2013.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os *Provedores (Providers)* corresponde ao fornecedor de serviços (Máquinas Virtuais MVs que pertencem a um determinado *Cliente (Client)*). Dessa forma, o *Provedor* possui uma relação com *Clientes*. Os *Provedores* também possuem *Máquinas Reais* (classe: *Machine*, subclasse: *RealMachine*). Finalmente, *Provedores* utiliza e oferece (tem) *Ferramentas de Gerenciamento (ManagementTools)*. As informações que podem ser recuperadas da ontologia *Provedor* são: Nome do provedor, Email do provedor (administrador), ID do provedor, Senha do provedor (administrador) e Descrição do provedor.

Nível de Formalidade: Semi-formal.

Escopo: Os *Provedores* fornecem MVs como serviço para os *Clientes*. Os *Provedores* possuem Máquinas Reais - MR (*RealMachines*).

Lista de conceitos: Cliente (*Client*), Máquinas Reais (*RealMachines*) e Ferramentas de Gerenciamento (*ManagementTools*)

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Usuário (*User*)

Data: 20 de outubro de 2013.

Última Atualização: 20 de outubro de 2013.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os *Usuários (User)* que consomem os serviços provisionados nas Máquinas Virtuais MVs (classe: *Machine*, subclasse: *VirtualMachine*) que pertencem a um determinado *Cliente (Client)*. As informações que podem ser recuperadas da ontologia *Usuário* são: Nome do usuário, Email do usuário, ID do usuário, ID do cliente, Senha do usuário e Descrição do usuário.

Nível de Formalidade: Semi-formal.

Escopo: Os Usuários simplesmente acessam os serviços providos nas MVs, as suas requisições geram demanda para a infraestrutura do *Cliente*. O cliente pode ter de zero a N ($0 \sim N$) usuários.

Lista de conceitos: Cliente (*Client*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Faturamento (*Billing*)

Data: 10 de julho de 2014.

Última Atualização: 15 de outubro de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os Faturamentos (*Billing*) representam o faturamento do cliente, o valor que ele deve pagar pela utilização da infraestrutura do provedor. Cada Faturamento (*Billing*) tem uma relação com a classe Histórico de Faturamento (*BillingHistory*).

Nível de Formalidade: Semi-formal.

Escopo: A classe Cliente (*Client*) tem uma relação com esta classe Faturamento (*Billing*) por meio da propriedade *hasBilling*.

Lista de conceitos: Cliente (*Client*), Faturamento (*Billing*), Histórico de Faturamento (*BillingHistory*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Histórico de Faturamento (*BillingHistory*)

Data: 11 de julho de 2014.

Última Atualização: 15 de outubro de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os Históricos de Faturamentos (*BillingHistory*) representam os históricos de um faturamento (*Billing*) do cliente, são armazenadas o valor a ser pago pela utilização de cada MV alocada pelo cliente que utiliza a infraestrutura do provedor, data de início e fim da utilização e o valor cobrado por hora. Cada Histórico de Faturamento (*BillingHistory*) tem uma relação com a classe Faturamento (*Billing*).

Nível de Formalidade: Semi-formal.

Escopo: A classe Faturamento (*Billing*) está relacionada com esta classe Histórico de Faturamento (*BillingHistory*) por meio da propriedade *hasBillingHistory*.

Lista de conceitos: Faturamento (*Billing*), Histórico de Faturamento (*BillingHistory*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Gerenciamento (*Management*)

Data: 20 de outubro de 2013.

Última Atualização: 12 de março de 2016

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os tipos de *Gerenciamento (Management)* que estão associados aos clientes. Existem quatro tipos de Gerenciamento: Financeiro (*FinancialManagement*), Energia (*PowerManagement*), Recursos (*ResourceManagement*) e SLA (*SLAManagement*). A classe Gerenciamento (*Management*) pode estar relacionada com a classe Ferramentas de Gerenciamento (*ManagementTools*). As informações que podem ser recuperadas da ontologia *Gerenciamento* são: Nome do gerenciamento, ID do gerenciamento, Política(s) do gerenciamento e Descrição do gerenciamento.

Nível de Formalidade: Semi-formal.

Escopo: Dentro do escopo desta ontologia, o *Cliente* deve ter dois tipos de gerenciamentos obrigatórios que são os de Recursos e de SLA e ainda por optar pelo gerenciamento Financeiro. Quanto ao gerenciamento de energia, caso ele possua uma relação com esse gerenciamento então ele estará concordando em permitir que o provedor faça um gerenciamento de energia que talvez prejudique o seu desempenho e disponibilidade, porém poderá contribuir com uma menor emissão de gás carbônico (*Green Computing*). A classe *Gerenciamento (Management)* e suas subclasses que determinam os seus tipos (supracitados) estão relacionados a uma Política (*Policy*) do sistema que definirá detalhes de seu funcionamento. Opcionalmente a classe *Gerenciamento (Management)* e suas subclasses podem estar relacionadas com algumas *Ferramentas de Gerenciamento (ManagementTools)*.

Lista de conceitos: Cliente (*Client*), Política (*Policy*) e Ferramentas de Gerenciamento (*ManagementTools*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Política (*Policy*)

Data: 22 de outubro de 2013.

Última Atualização: 12 de março de 2016

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os tipos de *Políticas (Policies)* que estão associados aos tipos de *Gerenciamento (Management)* dos clientes. Cada uma delas também está relacionada a um Algoritmo (*Algorithm*) que deve executá-la na infraestrutura. Existem cinco tipos de *Políticas*: Financeira (*FinancialPolicy*), Energia (*PowerPolicy*), Escalonamento (*ScalingPolicy*), SLA (*SLAPolicy*) e Otimização (*OptimizationPolicy*). As informações que podem ser recuperadas da ontologia *Política* são: Nome da política, ID da política, Período (*Period*) da política, Status da política e Descrição da política.

Nível de Formalidade: Semi-formal.

Escopo: Dentro do escopo desta ontologia, cada Política (*Policy*) deve estar associada a um tipo de gerenciamento. Os funcionamentos das políticas são definidos pelos provedores (*Providers*) e clientes (*Clients*) de acordo com a necessidade de ambos. A Política de SLA (*SLAPolicy*) trata do gerenciamento de SLA/QoS, portanto está ligada a classe *Qualidade de Serviço (QualityOfService)* que estará relacionada a classe *Atributo de QoS (QoSAttributes)* a qual armazena os níveis de QoS do acordo SLA entre provedor e cliente. A Política de Escalonamento (*ScalingPolicy*) indica apenas qual é o tipo de elasticidade que a infraestrutura deve seguir, o escalonamento pode ocorrer de forma Vertical ou Horizontal (*VerticalPolicy* e *HorizontalPolicy* que são sub-classes de *ScalingPolicy*). O escalonamento vertical corresponde a variação de recursos na própria máquina virtual (*VirtualMachine*) como Memória, Disco, Cpu, já o escalonamento horizontal corresponde a variação no número de máquinas virtuais. A Política de Energia (*PowerPolicy*) indica se os recursos serão ou não otimizados buscando diminuir os gastos com o consumo de energia e preservando o meio ambiente (*Green Computing*), esta política pode ser mais interessante para os provedores de energia. A Política de Otimização (*OptimizationPolicy*) pode ser utilizada por qualquer um dos quatros tipos de gerenciamento. Ela consiste na utilização de algoritmos de otimização na tentativa de otimizar o gerenciamento. A Política Financeira (*FinancialPolicy*) visa oferecer aos clientes várias opções de gerenciamento financeiros de seus recursos. Essa política possui subdivisões (subclasses) que oferecem diferentes formas de controle orçamentário que são: Política baseada no ROI (*Return on Investment*) – (*ROIPolicy*) que é baseada no conceito do retorno do investimento. Ela gerencia o investimento inicial do cliente, que contrata recursos virtualizados por um período determinado, e que gostaria de verificar periodicamente se esse investimento está sendo utilizado de forma consciente no decorrer deste período. Outras sub-políticas Financeiras são as Políticas de Custo Definido e as Políticas Sob Demanda (*DefinedCostPolicy* e *OnDemandPolicy*, respectivamente). As Políticas de Custo Definido (*DefinedCostPolicy*) exigem que o período inicial seja de no mínimo mensal, o cliente informa o valor inicial que pretende gastar com IaaS no período, em seguida ele deve escolher entre duas outras políticas: 1) Política Diária Limitada (*LimitedDailyPolicy*), na qual o cliente deve informar um valor fixo de utilização para cada dia do período (valor diário), ou, 2) Política de Agendamento (*ScheduledPolicy*), na qual o cliente deve dividir o valor inicial durante os dias do período. A *LimitedDailyPolicy* e *ScheduledPolicy* oferecem vantagens antagônicas, enquanto a primeira permite um controle mais simples, dividindo o valor inicial em partes iguais durante os dias do mês, sendo assim menos flexível, e a segunda é justamente o contrário, pois permite que o cliente escolha a quantidade de horas/máquina que deseja utilizar em determinados dias do mês. Já nas Políticas Sob Demanda (*OnDemandPolicy*), o cliente também define um valor inicial e ambas são baseadas no conceito *on-demand*, permitindo que o gasto siga conforme a demanda, porém há algumas diferenças sutis. A primeira sub-política *ondemand* é a *MaximumValuePolicy*, que faz uma analogia ao sistema de serviços pré-pagos, ou seja, os gastos com infraestrutura ocorrem sem até quando o valor do gasto ultrapassar o valor inicial estipulado, neste ponto toda a infraestrutura será interrompida até que “se insira créditos” para continuar a utilizar o serviço. A segunda é a *AssistedCostPolicy*, que é semelhante ao sistema de *Billing Alert* da Amazon, o gasto também é livre para atender a demanda, porém quando valor inicial é ultrapassado, uma mensagem de alerta é enviada ao cliente. Contudo, o gasto com infraestrutura não é interrompido e, dessa forma, carece de uma assistência do cliente para

que o controle financeiro não extrapole o seu orçamento.

Lista de conceitos: Cliente (*Client*), Provedor (*Provider*), Qualidade de Serviço (*QualityOfService*), Gerenciamento (*Management*) e Algoritmo (*Algorithm*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- E.Caron, L. Rodero-Merino, F. Desprez, and A. Muresan, "Auto-scaling, load balancing and monitoring in commercial and open-source clouds," *INRIA, Rapport de recherche RR-7857*, Fevereiro, 2012.
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

Web Sites:

- AMAZON EC2. *Opções de compra de instâncias do Amazon EC2*. Disponível em <<http://aws.amazon.com/pt/ec2/purchasing-options/>>. Acesso em: 03 de Agosto de 2014.
- AMAZON EC2. *Definição de preço do Amazon EC2*. Disponível em <<http://aws.amazon.com/pt/ec2/pricing/>>. Acesso em: 03 de Agosto de 2014.

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Período (*Period*)

Data: 14 de janeiro de 2015.

Última Atualização: 11 de outubro de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os Períodos (*Period*) que estão associados a política (*Policy*) *PowerPolicy*. Definem o período inicial e final que a política (*Policy*) *PowerPolicy* deve atuar.

Nível de Formalidade: Semi-formal.

Escopo: Esta classe de Período (*Period*) está associada a subclasse *PowerPolicy* por meio das propriedades de objeto *hasPeriodStart* e *hasPeriodEnd*.

Lista de conceitos: *PowerPolicy* subclasse da classe Política (*Policy*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: ROI (*ROI*)

Data: 15 de janeiro de 2015.

Última Atualização: 11 de outubro de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre o ROI (*ROI – Return on Investment*) que está associado a política *ROI*Policy subclasse de Política (*Policy*). Definem o ROI alcançado pela política.

Nível de Formalidade: Semi-formal.

Escopo: Esta classe de ROI (*ROI*) está associada a subclasse *ROI*Policy por meio da propriedade de objeto *hasROI*.

Lista de conceitos: *ROI*Policy subclasse da classe Política (*Policy*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Dias Programados (*ScheduledDays*)

Data: 15 de janeiro de 2015.

Última Atualização: 11 de outubro de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os Dias Programados (*ScheduledDays*) que estão associados a política *ScheduledPolicy* subclasse de Política (*Policy*). Definem os dias programados que serão considerados pela política.

Nível de Formalidade: Semi-formal.

Escopo: Esta classe de Dias Programados (*ScheduledDays*) está associada a subclasse *ScheduledPolicy* por meio da propriedade de objeto *hasScheduledDays*.

Lista de conceitos: *ScheduledPolicy* subclasse da classe Política (*Policy*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Qualidade de Serviço (*QualityOfService*)

Data: 25 de outubro de 2013.

Última Atualização: 08 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre a *Qualidade de Serviço (QualityOfService)* dos recursos da IaaS, mais especificamente das Máquinas Virtuais (*VirtualMachines*) que são utilizados pelos usuários (*Users*) dos clientes (*Clients*) que contrataram IaaS. Esta ontologia pode ser utilizada para obter informações a respeito da qualidade de serviço das Máquinas Virtuais, por exemplo, a disponibilidade de utilização de uma determinada VM. Essa ontologia também está relacionada a classe de Política de SLA (*SLAPolicy*). A classe *QualityOfService* está relacionada com a classe Atributo de QoS (*QoSAttribute*) em uma relação de 1 para N por meio da propriedade *hasQoSAttribute*.

Nível de Formalidade: Semi-formal.

Escopo: Classe que serve como um “link” entre os recursos (*VirtualMachines*) e os seus atributos de qualidades (*QoSAttributes*) e também entre a Política de SLA (*SLAPolicy*) e os atributos de QoS que definem o acordo.

Lista de conceitos: Máquina Virtual (*VirtualMachine*), Política de SLA (*SLAPolicy*) e Atributo de QoS (*QoSAttribute*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Yuanjun Laili, Fei Tao, Lin Zhang, Ying Cheng, Yongliang Luo, Bhaba R. Sarker, *A Ranking Chaos Algorithm for dual scheduling of cloud service and computing resource in private cloud, Computers in Industry*, Volume 64, Issue 4, May 2013, Pages 448-463.
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

Web Sites:

- AMAZON EC2. *Opções de compra de instâncias do Amazon EC2*. Disponível em <<http://aws.amazon.com/pt/ec2/purchasing-options/>>. Acesso em: 03 de Agosto de 2014.
- AMAZON EC2. *Definição de preço do Amazon EC2*. Disponível em <<http://aws.amazon.com/pt/ec2/pricing/>>. Acesso em: 03 de Agosto de 2014.

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Atributo de QoS (*QoSAttribute*)

Data: 26 de outubro de 2013.

Última Atualização: 08 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre os *Atributos de QoS (QoSAttributes)* que são solicitados pelos clientes. Esta ontologia pode ser utilizada para obter informações a respeito dos atributos de QoS das Máquinas Virtuais (*VirtualMachines*) ou dos atributos de QoS acordados entre cliente e provedor que são definidos na Política de SLA (*SLAPolicy*). As sub-classes de *QoSAttribute* são os atributos de QoS propriamente ditos: Disponibilidade (*Availability*), Custo (*Cost*), Tempo de Resposta (*ResponseTime*) e Capacidade (*Capacity*).

Nível de Formalidade: Semi-formal.

Escopo: Classe e sub-classes servem para armazenar os valores (ou níveis) de QoS de cada atributo. Todos os atributos de QoS possuem uma propriedade (*hasValue*) que assume um valor decimal, sendo que para a Disponibilidade esse valor varia entre um valor definido pelo provedor (0,9 até 1,0), para o custo esse valor assume apenas valores positivos ou iguais a zero (≥ 0), para tempo de resposta apenas valores positivos (> 0) e para capacidade assume um dos quatro valores fixos iniciais (0,25 ou 0,5 ou 0,75 ou 1,00) que representam capacidades desde de pequenas (*Small*), médias (*Medium*), grandes (*Large*) até extra grandes (*ExtraLarge* or *XLarge*). Cada capacidade é um valor de *index* para uma determina configuração da Máquina Virtual (*VirtualMachine*). Novos tipos (sub-classes) de capacidade podem ser criados conforme a necessidade do provedor e clientes.

Lista de conceitos: Qualidade de Serviço (*QualityOfService*).

Fonte de Conhecimento:**Especialistas:**

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Yuanjun Laili, Fei Tao, Lin Zhang, Ying Cheng, Yongliang Luo, Bhaba R. Sarker, *A Ranking Chaos Algorithm for dual scheduling of cloud service and computing resource in private cloud*, *Computers in Industry*, Volume 64, Issue 4, May 2013, Pages 448-463.
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

Web Sites:

- AMAZON EC2. *Opções de compra de instâncias do Amazon EC2*. Disponível em <<http://aws.amazon.com/pt/ec2/purchasing-options/>>. Acesso em: 03 de Agosto de 2014.
- AMAZON EC2. *Definição de preço do Amazon EC2*. Disponível em <<http://aws.amazon.com/pt/ec2/pricing/>>. Acesso em: 03 de Agosto de 2014.

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Máquina (*Machine*)

Data: 26 de outubro de 2013.

Última Atualização: 13 de setembro de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre *Máquina (Machine)* representa dois tipos de máquinas: *Máquina Real (RealMachine)* e *Máquina Virtual (VirtualMachine)*. As máquinas reais estão associadas aos provedores (*Providers*) e as máquinas virtuais estão associadas as máquinas reais e aos Clientes (*Clients*). As máquinas virtuais também estão associadas a classe (e subclasses) de *Estado das Máquinas Virtuais (VirtualMachineStatus)* que definem o estado das máquinas virtuais. As máquinas reais estão associadas a classe (subclasses) *Estado de Energia (PowerStatus)* que define o estado das máquinas reais. Ambas as máquinas (real e virtual) estão relacionadas a classe (e subclasses) *Capacidade da Máquina (MachineCapacity)* que através de uma relação com a classe Recursos (*Resources*) irá definir a configuração das máquinas e de suas conexões de rede.

As informações que podem ser recuperadas da ontologia *Machine* são: Nome da máquina, ID da máquina, Id do provedor, Id do cliente, Id do status da máquina virtual, Id do status da energia da máquina real e Descrição da máquina.

Nível de Formalidade: Semi-formal.

Escopo: As sub-classes de Máquina (*Machine*): Máquina Real (*RealMachine*) e Máquina Virtual (*VirtualMachine*) são as classes que representam os principais itens de uma IaaS, diversos indivíduos (instâncias) devem ser criados para representar o mundo real. Essa ontologia possui uma relação com a classe *Capacidade da Máquina (MachineCapacity)* que através de uma propriedade denominada “*hasMachineCapacity*”.

Lista de conceitos: Cliente (*Client*), Estado das Máquinas Virtuais (*VirtualMachineStatus*), Estado da Energia das Máquinas Reais (*PowerStatus*), Provedor (*Provider*) e Capacidade da Máquina (*MachineCapacity*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Capacidade da Máquina (*MachineCapacity*)

Data: 28 de outubro de 2013.

Última Atualização: 12 de março de 2016

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: Ontologia sobre a *Capacidade da Máquina* (*MachineCapacity*) representa um conjunto de configurações de recursos que define um “tipo” de máquina, seja ela real ou virtual. Os atuais tipos de capacidade de máquinas disponíveis na ontologia são: *Capacidade da Máquina Real* (*RealMachineCapacity*), *Pequena Capacidade da Máquina* (*MachineCapacitySmall*), *Média Capacidade da Máquina* (*MachineCapacityMedium*), *Grande Capacidade da Máquina* (*MachineCapacityLarge*) e *ExtraGrande Capacidade da Máquina* (*MachineCapacityExtraLarge*), sendo que essas últimas quatro se referem as *Máquinas Virtuais* (*VirtualMachines*). Essa classe e subclasses são utilizadas pelas ontologias *Máquina* (*Machine*) e suas subclasses (*RealMachine* e *VirtualMachine*). Uma *Capacidade da Máquina* (*MachineCapacity*) está sempre associada as subclasses da classe *Recursos* (*Resources*). Por fim, uma *Capacidade da Máquina* (*MachineCapacity*) também está sempre associada com um *Software* (*Software*). As informações que podem ser recuperadas da ontologia *Capacidade da Máquina* (*MachineCapacity*) são: ID da capacidade da máquina e Descrição da capacidade da máquina.

Nível de Formalidade: Semi-formal.

Escopo: As sub-classes de *Capacidade da Máquina* (*MachineCapacity*) representam um tipo de configuração (*Real*, *Small*, *Medium*, *Large* e *ExtraLarge*) que uma *Máquina Real* (*RealMachine*) ou *Máquina Virtual* (*VirtualMachine*) podem ter dentro do domínio da ontologia. A classe *Capacidade da Máquina* (*MachineCapacity*) se relaciona com a classe *Máquina* (*Machine*) através de uma propriedade denominada “*hasMachineCapacity*”, há outra relação com a classe *Recursos* (*Resources*) por meio da propriedade “*hasResources*” e com a classe *Software* (*Software*) por meio das propriedades “*hasSoftware*” ou a sua subclasse “*hasOperationalSystem*”, sendo esta última a mais utilizada dentro do contexto de IaaS.

Lista de conceitos: *Máquina* (*Machine*), *Recursos* (*Resources*) e *Software* (*Software*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Recursos (*Resources*)

Data: 28 de outubro de 2013.

Última Atualização: 08 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre os *Recursos (Resources)* representa dentro do domínio um recurso computacional (virtual ou real). As subclasses de Recursos (*Resources*), que são: *Computação (Compute)* [subclasses: *CPU* e *GPU*], *Memória (Memory)* [subclasses: *RAM* e *Cache* [subclasses: *L1*, *L2* e *L3*]], *Rede (Network)*, *Armazenamento (Storage)* [subclasses: *Armazenamento Local (LocalStorage)* e *Armazenamento na Rede (NetworkStorage)*] e *Volume* [subclasse: *Imagem (Image)*], representam cada um dos recursos computacionais disponível na infraestrutura de uma *Máquina Real (RealMachine)* ou *Máquina Virtual (VirtualMachine)*, mas o relacionamento com as subclasses de *Máquina (Machine)* ocorre somente através da classe (e suas subclasses) *Capacidade da Máquina (MachineCapacity)*. Um ponto em particular é que a subclasse *Rede (Network)* possui uma relação com a classe *Protocolo (Protocol)* para definir qual é o protocolo utilizado pela rede. Além disso, também há relacionamentos com as classes *Modelo do Recurso (ResourceModel)* e *Fabricante do Recurso (ResourceManufacturer)*. As informações que podem ser recuperadas da ontologia *Recursos (Resources)* são: ID do recurso, Consumo de Energia e Descrição do recurso.

Nível de Formalidade: Semi-formal.

Escopo: As sub-classes de *Recursos (Resources)* representam um tipo de recurso computacional (*Compute*, *Memory*, *Network*, *Storage* e *Volume*) que uma *Máquina Real (RealMachine)* ou *Máquina Virtual (VirtualMachine)* podem ter dentro do domínio da ontologia. A classe *Capacidade da Máquina (MachineCapacity)* se relaciona com a classe *Recursos (Resources)* por meio da propriedade “*hasResources*”. Diversas propriedades de dados (*data properties*) estão relacionadas as subclasses de *Recursos (Resources)*, para *Computação (Compute)*, por exemplo, existem as propriedades: *hasNumberOfCores*, *hasFrequency*, *hasId*, *hasDesc*, *hasCost*, *hasPowerConsume*, *hasHyperThreading* e *hasArchitecture*. Além das propriedades de dados (*data properties*), há algumas propriedades de objetos (*object properties*), como, ainda no exemplo de *Computação (Compute)*; *hasResourceModel* que relaciona com a classe *Modelo do Recurso (ResourceModel)* e a *hasResourceManufacturer* que relaciona com a classe *Fabricante do Recurso (ResourceManufacturer)*.

Lista de conceitos: *Capacidade da Máquina (MachineCapacity)*, *Modelo do Recurso (ResourceModel)*, *Fabricante do Recurso (ResourceManufacturer)* e *Protocolo (Protocol)*.

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Modelo do Recurso (*ResourceModel*)

Data: 28 de outubro de 2013.

Última Atualização: 08 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre o *Modelo do Recurso (ResourceModel)* representa dentro do domínio o modelo de um recurso computacional (virtual ou real). Essa classe está relacionada com a classe Recursos (*Resources*). As informações que podem ser recuperadas da ontologia *Modelo do Recurso (ResourceModel)* são: ID do modelo do recurso, Descrição do modelo do recurso.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe Recursos (*Resources*) por meio da propriedade “*hasResourceModel*”.

Lista de conceitos: Recursos (*Resources*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Fabricante do Recurso (*ResourceManufacturer*)

Data: 28 de outubro de 2013.

Última Atualização: 08 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre o *Fabricante do Recurso (ResourceManufacturer)* representa dentro do domínio o fabricante de um recurso computacional (virtual ou real). Essa classe está relacionada com a classe Recursos (*Resources*). As informações que podem ser recuperadas da ontologia *Fabricante do Recurso (ResourceManufacturer)* são: ID do fabricante do recurso, Nome do fabricante do recurso, E-mail do fabricante do recurso e Marca do fabricante do recurso.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe Recursos (*Resources*) por meio da propriedade “*hasResourceManufacturer*”.

Lista de conceitos: Recursos (*Resources*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Estado de Energia (*PowerStatus*)

Data: 28 de outubro de 2013.

Última Atualização: 08 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre o *Estado de Energia (PowerStatus)* define em qual estado uma *Máquina Real (RealMachine)* pode estar. As subclasses da classe *Estado de Energia (PowerStatus)* são: *Hibernating*, *Off*, *On*, *Restarting*, *Shutdowing*, *Starting* e *Unknown*. As informações que podem ser recuperadas da ontologia *Estado de Energia (PowerStatus)* são: ID do estado de energia e descrição do estado de energia.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe *Máquina Real (RealMachine)* por meio da propriedade “*hasPowerStatus*”.

Lista de conceitos: *Máquina Real (RealMachine)*.

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Estado da Máquina Virtual (*VirtualMachineStatus*)

Data: 28 de outubro de 2013.

Última Atualização: 09 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre o *Estado da Máquina Virtual* (*VirtualMachineStatus*) define em qual estado uma *Máquina Virtual* (*virtualMachine*) pode estar. As subclasses da classe *Estado da Máquina Virtual* (*VirtualMachineStatus*) são: ***Deleting, MergingDrivers, Migrating, Paused, Restoring, Running, Saved, Saving, TurnedOff, TurningOff e TurningOn.*** As informações que podem ser recuperadas da ontologia *Estado da Máquina Virtual* (*VirtualMachineStatus*) são: ID do estado da máquina virtual e a sua descrição.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe *Máquina Virtual* (*VirtualMachine*) por meio da propriedade “*hasVirtualMachineStatus*”.

Lista de conceitos: *Máquina Virtual* (*VirtualMachine*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Algoritmo (*Algorithm*)

Data: 28 de outubro de 2013.

Última Atualização: 12 de março de 2016.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre o *Algoritmo (Algorithm)* define o algoritmo que deve ser utilizado pela Política (*Policy*) de gerenciamento. Os algoritmos servem para o gerenciamento estipulado pela política e uma subclasse é o algoritmo de otimização (*OptimizationAlgorithm*) que além de realizar o gerenciamento tenta otimizá-lo (esse algoritmo é sempre utilizado pela política de gerenciamento de energia (*PowerPolicy*)). As informações que podem ser recuperadas da ontologia *Algoritmo (Algorithm)* são: ID do algoritmo, Classe do algoritmo (nome completo da classe que implementa o algoritmo para ser utilizado para instanciá-lo via *reflection*) e Descrição do algoritmo.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe Política (*Policy*) por meio da propriedade “*hasAlgorithm*”.

Lista de conceitos: Política (*Policy*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Agent (*Agent*)

Data: 28 de outubro de 2013.

Última Atualização: 23 de agosto de 2015.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre o *Agente (Agent)* define o agente que deve ser utilizado pelo Algoritmo (*Algorithm*). Os agentes servem atuar (subclasse agente atuador – *AgentEffector*) ou monitorar (subclasse agente monitor *AgentMonitor*). Cada algoritmo poder um ou mais agentes e os agentes podem pertencer a um ou mais algoritmos, por exemplo, um agente monitor poderá servir para todos os tipos de algoritmos com as suas respectivas políticas (*Policies*).

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe Algoritmo (*Algorithm*) por meio da propriedade “*hasAgent*”.

Lista de conceitos: Algoritmo (*Algorithm*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Ferramentas de Gerenciamento (*ManagementTools*)

Data: 28 de outubro de 2013.

Última Atualização: 09 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A ontologia sobre as Ferramentas de Gerenciamento (*ManagementTools*) define em quais são as ferramentas de gerenciamento associadas à um tipo de Gerenciamento (*Management*). As subclasses de Ferramentas de Gerenciamento (*ManagementTools*) são: *Hypervisor* [subclasses: *Hyper-V*, *KVM*, *VMWare* e *Xen*], *APIs* [subclasse: *Libvirt*], *CLI_TOOLS*, *DashBoard* e *Orchestrator*. As informações que podem ser recuperadas da ontologia *Ferramentas de Gerenciamento* (*ManagementTools*) são: ID da ferramenta de gerenciamento, a sua descrição e opcionalmente os Ids do *Hypervisor*, da ferramenta de CLI, da API, da *DashBoard* e do Orquestrador.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a classe Gerenciamento (*Management*) por meio da propriedade “*hasManagementTools*”.

Lista de conceitos: Gerenciamento (*Management*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks*, *Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Protocolo (*Protocol*)

Data: 29 de outubro de 2013.

Última Atualização: 09 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A princípio a ontologia sobre Protocolo (*Protocol*) define apenas quais são os protocolos utilizados pela subclasse Rede (*Network*) da classe Recursos (*Resources*). Porém, essa classe pode ser utilizada por outras classes no futuro como, por exemplo, uma futura classe sobre segurança que utilizado uma subclasse de protocolo denominada SSH. As subclasses de Protocolo (*Protocol*) são: *HTTP*, *IP* [subclasses: *IPv4* e *IPv6*], *SCP* e *SSH*. As informações que podem ser recuperadas da ontologia *Protocolo* (*Protocol*) são: ID do protocolo, informação de característica do protocolo e a sua descrição.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a subclasse Rede (*Network*) da classe pai Recursos (*Resources*) por meio da propriedade “*hasProtocol*”.

Lista de conceitos: Rede (*Network*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

DOCUMENTO DE ESPECIFICAÇÃO DE REQUISITO DA ONTOLOGIA

Domínio: Software (*Software*)

Data: 29 de outubro de 2013.

Última Atualização: 09 de agosto de 2014.

Conceitualizado por: Luis H. V. Nakamura

Implementado por: Luis H. V. Nakamura.

Propósito: A princípio a ontologia sobre Software (*Software*) define apenas quais são os softwares utilizados nas máquinas virtuais. Como o domínio desta ontologia abrange apenas IaaS então o software mais importante é o Sistema Operacional (*OperationalSystem*) que é uma subclasse de Software (*Software*) e ainda possui as subclasses: **Linux**, **MacOS** e **Windows**. As informações que podem ser recuperadas da ontologia *Software* (*Software*) são: ID do software, Versão do software, Nome do software, Fabricante do software, Email do Fabricante do software, Serial do software, Custo do software, Última atualização do software, número de licenças do software e a sua descrição.

Nível de Formalidade: Semi-formal.

Escopo: Essa classe está relacionada com a subclasse Capacidade da Máquina (*MachineCapacity*) por meio de uma das duas propriedades: “*hasSoftware*” ou “*hasOperationalSystem*”.

Lista de conceitos: Capacidade da Máquina (*MachineCapacity*).

Fonte de Conhecimento:

Especialistas:

- Luis H. V. Nakamura

Artigos:

- R. Dukaric, M.B. Juric, *Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems* (2012).
- Zhang, M.; Ranjan, R.; Haller, A.; Georgakopoulos, D.; Menzel, M.; Nepal, S., "An ontology-based system for Cloud infrastructure services' discovery," in Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on , vol., no., pp.524-530, 14-17 Oct. 2012

Documentos da Conceituação da Ontologia

Infrastructure as a Service Ontology – IaaSOnt Glossário de Termos – GT (*Glossary of Terms*)

Nome	Descrição
Agent (Agente)	O agente de <i>software</i> que é responsável pela execução dos algoritmos e mecanismos de monitoramento (subclasse AgentMonitor) e atuação (subclasse AgentEffector).
Algorithm (Algoritmo)	O algoritmo que é utilizado para uma determinada tarefa, por exemplo o algoritmo de otimização (subclasse OptimizationAlgorithm).
Billing (Faturamento)	O faturamento do cliente, o valor que ele deve pagar pela utilização da infraestrutura do provedor.
BillingHistory (Histórico de Faturamento)	Os Históricos de Faturamentos representam os históricos de um faturamento do cliente, são armazenadas o valor a ser pago pela utilização de cada MV alocada pelo cliente que utiliza a infraestrutura do provedor, data de início e fim da utilização e o valor cobrado por hora.
Client (Cliente)	Os Clientes (<i>Clients</i>) que solicitam os serviços IaaS (Infrastructure as a Service) que são as Máquinas Virtuais – MV ou VM (<i>Virtual Machines</i>).
Machine (Máquina)	A Máquina (<i>Machine</i>) representa dois tipos de máquinas: <i>Máquina Real</i> (<i>RealMachine</i>) e <i>Máquina Virtual</i> (<i>VirtualMachine</i>). As máquinas reais estão associadas aos provedores (<i>Providers</i>) e as máquinas virtuais estão associadas as máquinas reais e aos Clientes (<i>Clients</i>).
MachineCapacity (Capacidade da Máquina)	A Capacidade da Máquina (<i>MachineCapacity</i>) representa um conjunto de configurações de recursos que define um “tipo” de máquina, seja ela real ou virtual. Os atuais tipos de capacidade de máquinas disponíveis na ontologia são: Capacidade da Máquina Real (<i>RealMachineCapacity</i>), Pequena Capacidade da Máquina (<i>MachineCapacitySmall</i>), Média Capacidade da Máquina (<i>MachineCapacityMedium</i>), Grande Capacidade da Máquina (<i>MachineCapacityLarge</i>) e ExtraGrande Capacidade da Máquina (<i>MachineCapacityExtraLarge</i>), sendo que essas últimas quatro se referem as Máquinas Virtuais (<i>VirtualMachines</i>).
Management (Gerenciamento)	Gerenciamento (<i>Management</i>) que estão associados aos clientes. Existem quatro tipos de Gerenciamento: Financeiro (<i>FinancialManagement</i>), Energia (<i>PowerManagement</i>), Recursos (<i>ResourceManagement</i>) e SLA (<i>SLAManagement</i>).
ManagementTools (Ferramentas de Gerenciamento)	Ferramentas de Gerenciamento (<i>ManagementTools</i>) define em quais são as ferramentas de gerenciamento associadas à um tipo de Gerenciamento (<i>Management</i>). As subclasses de Ferramentas de Gerenciamento (<i>ManagementTools</i>) são: <i>Hypervisor</i> [subclasses: <i>Hyper-V</i> , <i>KVM</i> , <i>VMWare</i> e <i>Xen</i>], <i>APIs</i> [subclasse: <i>Libvirt</i>], <i>CLI_TOOLS</i> , <i>DashBoard</i> e <i>Orchestrator</i> .

<i>Period</i> (Período)	Períodos (<i>Period</i>) que estão associados a política (<i>Policy</i>) <i>PowerPolicy</i> . Definem o período inicial e final que a política (<i>Policy</i>) <i>PowerPolicy</i> deve atuar.
<i>Policy</i> (Políticas)	<i>Políticas (Policies)</i> que estão associados aos tipos de <i>Gerenciamento (Management)</i> dos clientes. Cada uma delas também está relacionada a um Algoritmo (<i>Algorithm</i>) que deve executá-la na infraestrutura. Existem cinco tipos de <i>Políticas</i> : Financeira (<i>FinancialPolicy</i>), Energia (<i>PowerPolicy</i>), Escalonamento (<i>ScalingPolicy</i>), SLA (<i>SLAPolicy</i>) e Otimização (<i>OptimizationPolicy</i>).
<i>PowerStatus</i> (Estado de Energia)	O <i>Estado de Energia (PowerStatus)</i> define em qual estado uma <i>Máquina Real (RealMachine)</i> pode estar. As subclasses da classe <i>Estado de Energia (PowerStatus)</i> são: <i>Hibernating</i> , <i>Off</i> , <i>On</i> , <i>Restarting</i> , <i>Shutdowing</i> , <i>Starting</i> e <i>Unknown</i> .
<i>Protocol</i> (Protocolo)	Protocolo (<i>Protocol</i>) define apenas quais são os protocolos utilizados pela subclasse Rede (<i>Network</i>) da classe Recursos (<i>Resources</i>). As subclasses de Protocolo (<i>Protocol</i>) são: <i>HTTP</i> , <i>IP</i> [subclasses: <i>IPv4</i> e <i>IPv6</i>], <i>SCP</i> e <i>SSH</i> .
<i>Provider</i> (Provedor)	<i>Provedores (Providers)</i> corresponde ao fornecedor de serviços (MVs que pertencem a um determinado <i>Cliente (Client)</i>).
<i>QoSAttribute</i> (Atributos de QoS)	Os <i>Atributos de QoS (QoSAttributes)</i> que são solicitados pelos clientes. Esta ontologia pode ser utilizada para obter informações a respeito dos atributos de QoS das Máquinas Virtuais (<i>VirtualMachines</i>) ou dos atributos de QoS acordados entre cliente e provedor que são definidos na Política de SLA (<i>SLAPolicy</i>). As sub-classes de <i>QoSAttribute</i> são os atributos de QoS propriamente ditos: Disponibilidade (<i>Availability</i>), Custo (<i>Cost</i>), Tempo de Resposta (<i>ResponseTime</i>) e Capacidade (<i>Capacity</i>).
<i>QualityOfService</i> (Qualidade de Serviço)	A <i>Qualidade de Serviço (QualityOfService)</i> dos recursos da IaaS, mais especificamente das MVs (<i>VirtualMachines</i>) que são utilizados pelos usuários (<i>Users</i>) dos clientes (<i>Clients</i>) que contrataram IaaS.
<i>ResourceManufacturer</i> (Fabricante do Recurso)	O Fabricante do Recurso (<i>ResourceManufacturer</i>) representa dentro do domínio o fabricante de um recurso computacional (virtual ou real).
<i>ResourceModel</i> (Modelo do Recurso)	O Modelo do Recurso (<i>ResourceModel</i>) representa o modelo de um recurso computacional (virtual ou real).
<i>Resources</i> (Recursos)	Os Recursos (<i>Resources</i>) representam recursos computacionais (virtual ou real). As subclasses de Recursos (<i>Resources</i>) são: <i>Computação (Compute)</i> [subclasses: <i>CPU</i> e <i>GPU</i>], <i>Memória (Memory)</i> [subclasses: <i>RAM</i> e <i>Cache</i> [subclasses: <i>L1</i> , <i>L2</i> e <i>L3</i>]], <i>Rede (Network)</i> , <i>Armazenamento (Storage)</i> [subclasses: <i>Armazenamento Local (LocalStorage)</i> e <i>Armazenamento na Rede (NetworkStorage)</i>] e <i>Volume</i> [subclasse: <i>Imagem (Image)</i>].
<i>ROI</i> (ROI)	O ROI (<i>ROI – Return on Investment</i>) que está associado a política <i>ROIPolicy</i> subclasse de Política (<i>Policy</i>).
<i>ScheduleDays</i> (Dias Programados)	Os Dias Programados (<i>ScheduledDays</i>) que estão associados a política <i>ScheduledPolicy</i> subclasse de Política (<i>Policy</i>). Definem os dias programados que serão considerados pela política.
<i>Software</i> (Software)	O Software (<i>Software</i>) define apenas quais são os softwares utilizados nas máquinas virtuais. Como o domínio desta ontologia abrange

	apenas IaaS então o software mais importante é o Sistema Operacional (<i>OperationalSystem</i>) que é uma subclasse de Software (<i>Software</i>) e ainda possui as subclasses: <i>Linux</i> , <i>MacOS</i> e <i>Windows</i> .
User (Usuário)	Os <i>Usuários</i> (<i>User</i>) que consomem os serviços provisionados nas Máquinas Virtuais MVs (classe: <i>Machine</i> , subclasse: <i>VirtualMachine</i>) que pertencem a um determinado <i>Cliente</i> (<i>Client</i>).
VirtualMachineStatus (Estado da Máquina Virtual)	O Estado da Máquina Virtual (<i>VirtualMachineStatus</i>) define em qual estado uma <i>Máquina Virtual</i> (<i>virtualMachine</i>) pode estar. As subclasses da classe <i>Estado da Máquina Virtual</i> (<i>VirtualMachineStatus</i>) são: <i>Deleting</i> , <i>MergingDrivers</i> , <i>Migrating</i> , <i>Paused</i> , <i>Restoring</i> , <i>Running</i> , <i>Saved</i> , <i>Saving</i> , <i>TurnedOff</i> , <i>TurningOff</i> e <i>TurningOn</i> .

Infrastructure as a Service Ontology – IaaSOnt

Árvore de Classificação de Conceitos (Concept-Classification Tree)

Thing

- |
- | --- **Agent**
 - | |- **AgentEffector**
 - | |- **AgentMonitor**
- |
- | --- **Algorithm**
 - | |- **OptimizationAlgorithm**
- |
- | --- **Billing**
- |
- | --- **BillingHistory**
- |
- | --- **Client**
- |
- | --- **Machine**
 - | |- **RealMachine**
 - | |- **VirtualMachine**
- |
- | --**MachineCapacity**
 - | |- **MachineCapacityExtraLarge**
 - | |- **MachineCapacityLarge**
 - | |- **MachineCapacityMedium**
 - | |- **MachineCapacitySmall**
 - | |- **RealMachineCapacity**
- |
- | --**Management**
 - | |- **FinancialManagement**
 - | |- **PowerManagement**
 - | |- **ResourceManagement**
 - | |- **SLAManagement**
- |
- | --**ManagementTools**
 - | |- **APIs**
 - | |- **LibVirt**
 - | |- **CLI_Tools**
 - | |- **DashBoard**
 - | |- **HyperVisor**
 - | |- **Hyper-V**
 - | |- **KVM**
 - | |- **VMWare**
 - | |- **Xen**
 - | |- **Orchestrator**
- |

```
| --- Period
| --Policy
|   |- FinancialPolicy
|   |   |-DefinedCostPolicy
|   |   |   |-LimitedDailyPolicy
|   |   |   |-ScheduledPolicy
|   |   |-OnDemandCostPolicy
|   |   |   |-AssistedCostPolicy
|   |   |   |-MaximumValuePolicy
|   |   |-ROIPolicy
|   |- OptimizationPolicy
|   |- PowerPolicy
|   |- ScalingPolicy
|   |   |-HorizontalPolicy
|   |   |-VerticalPolicy
|   |- SLAPolicy
|
| --PowerStatus
|   |- Hibernating
|   |- Off
|   |- On
|   |- Restarting
|   |- Shutdowning
|   |- Starting
|   |- Unknown
|
| --Protocol
|   |- HTTP
|   |- IP
|   |   |-IPv4
|   |   |-IPv6
|   |- SCP
|   |- SSH
|
| --QoSAttribute
|   |- Availability
|   |- Capacity
|   |- Cost
|   |- ResponseTime
|
| --QualityOfService
|
| --ResourceManufacturer
|
| --ResourceModel
|
```

```
| --Resources
|   |- Compute
|   |   |-CPU
|   |   |-GPU
|   |- Memory
|   |   |-Cache
|   |       |-L1
|   |       |-L2
|   |       |-L3
|   |   |-RAM
|   |- Network
|   |- Storage
|   |   |-LocalStorage
|   |   |-NetworkStorage
|   |- Volume
|   |   |-Image
|
| --ROI
|
| --ScheduledDays
|
| --Software
|   |- OperationalSystem
|       |-Linux
|       |-MacOs
|       |-Windows
|
| --User
|
| --VirtualMachineStatus
|   |- Deleting
|   |- MergingDrivers
|   |- Migrating
|   |- Paused
|   |- Restoring
|   |- Running
|   |- Saved
|   |- Saving
|   |- TurnedOff
|   |- TurningOff
|   |- TurnedOn
```

Infrastructure as a Service Ontology – IaaSOnt

Diagramas de Relações Binárias (Binary-Relations Diagrams)

Infrastructure as a Service Ontology – IaaSOnt

Dicionário de Conceitos (Concept Dictionary)

Nome do Conceito	Instâncias	Atributos	Relações
Agent	[Name]Agent	Algorithm AgentEffector AgentMonitor	hasAgent isAgentOf hasAlgorithm hasId hasAgentClass hasAgentName hasDescription
Algorithm	[Name]Algorithm	Agent Policy OptimizationAlgorithm	hasAgent hasAlgorithm hasId hasDescription hasParameter hasReturn
Billing	[Name]Billing	Client BillingHistory	hasBilling hasBillingHistory hasId hasActualValue hasBillingDate hasBillingDiaryCost hasBillingDiaryHoursUsage hasBillingAlert hasBillingAlertEmail
BillingHistory	[Name]BillingHistory	BillingHistory	hasBillingHistory hasId hasBillingHistoryStartDate hasBillingHistoryEndDate hasBillingHistoryCurrentValue
Client	[Name]Client	Billing Provider User Machine VirtualMachine Management	hasBilling hasProvider isProviderOf hasUser isUserOf hasMachine isMachineOf hasVirtualMachine isVirtualMachineOf hasManagement isManagementOf hasId hasName hasEmail hasLogin hasPassword
Machine	[Name]Machine	AgentMonitor Client MachineCapacity VirtualMachine RealMachine	hasMonitoring isMonitoringBy hasMachine isMachineOf hasMachineCapacity isMachineCapacity

			hasId hasName hasCost hasCPULoad hasDiskLoad hasMemoryLoad hasNetworkLoad hasFileSystem hasNumberOfVirtualMachines hasSerial hasURI
Machine Capacity	[Name]Machine Capacity	Machine Software Resources MachineCapacitySmall MachineCapacityExtraLarge MachineCapacityMedium MachineCapacityLarge RealMachineCapacity	hasMachineCapacity hasSoftware isSoftwareOf hasResources isResourcesOf <hr/> hasId
Management	[Name]Management	Policy Client ManagementTools FinancialManagement ResourceManagement PowerManagement SLAManagement	hasPolicy isPolicyOf hasManagement isManagementOf hasManagementTools isManagementToolsOf <hr/> hasId
Management Tools	[Name]Management Tools	Management APIs Libvirt CLI_Tools DarshBoard Hypervisor Hyper-V KVM VMWare Xen Orchestrator	hasManagementTools isManagementToolsOf <hr/> hasId
Period	[Name][Datetime]Period	PowerPolicy	hasPeriodStart hasPeriodEnd <hr/> hasId hasCost hasHour hasDay hasDayOfMonth hasMonth hasYear hasWeekDay
Policy	[Name]Policy	Algorithm Management FinancialPolicy DefinedCostPolicy LimitedDailyPolicy ScheduledPolicy OnDemandCostPolicy AssistedCostPolicy	hasAlgorithm hasPolicy isPolicyOf <hr/> hasId hasName hasDescription hasCost hasPowerConsume

		MaximumValuePolicy ROIPolicy ScalingPolicy HorizontalPolicy VerticalPolicy PowerPolicy OptimizationPolicy SLAPolicy	hasInitialValue hasLastUpdate hasFixedDiaryValue hasHorizontalScalingLimit hasHorizontalScalingValue hasNumberOfRealMachines hasNumberOfVirtualMachines hasRateApplied hasVerticalScalingLimit hasVerticalScalingValue
PowerStatus	[Name]PowerStatus	RealMachine On Off Restarting Hibernating Shuttingdown Starting Unknown	hasPowerStatus hasId
Protocol	[Name]Protocol	Network SCP IP IPv4 IPv6 HTTP SSH	hasProtocol hasId hasDescription
Provider	[Name]Provider	Client RealMachine	hasClient hasProvider hasRealMachine hasId hasName hasDescription hasURI
QoSAttribute	[Name]QoSAttribute	QualityOfService Availability Capacity Cost ResponseTime	hasQoSAttribute isQoSAttribute hasId hasName hasDescription hasAmount hasValue hasInitialValue hasFinalValue
QualityOfService	[Name]QualityOfService	QoSAttribute SLAPolicy VirtualMachine	hasQoSAttribute hasSLA hasQualityOfService hasId
Resources	[Name]Resource	MachineCapacity ResourceManufacturer ResourceModel Compute CPU GPU Memory Cache L1 L2 L3	hasResources hasResourceManufacturer hasResourceModel hasId hasName hasDescription hasArchitecture hasBandwidth hasCost hasFrequency hasNumberOfCores

		RAM Network	hasPowerConsumption hasSerial hasSize isHyperThreading hasBrandName
Resource Manufacturer	[Name]Resource Manufacture	Resources	hasResource Manufacturer hasId hasName hasDescription hasManufacturerEmail
ResourceModel	[Name]ResourceModel	Resources Storage LocalStorage NetworkStorage Volume Image	hasResourceModel hasId hasName hasDescription hasStorageMaxSize hasStorageMinSize hasStorageSize hasSize hasType
ROI	[Name]ROI	ROIPolicy	hasROI hasId hasName hasDescription hasReturn hasInvestment hasValue
ScheduledDays	[Name]ScheduledDays	ScheduledPolicy	hasScheduledDays hasId hasDay hasMonth hasYear hasWeekDay
Software	[Name]Software	MachineCapacity OperationalSystem Linux MacOS Windows	hasSoftware hasId hasName hasDescription hasSerial hasCost hasDeveloper
User	[Name]User	Client	hasUser hasId hasName hasDescription
VirtualMachine Status	[Name]VirtualMachine Status	VirtualMachine Migrating MergingDrivers Paused Restoring TurningOn Deleting Running Saved TurnedOff TurningOff Saving	hasVirtualMachine Status hasId hasName hasDescription

Infrastructure as a Service Ontology – IaaSOnt

Tabelas de Relações Binárias (Binary-Relations Tables)

Nome da Relação	hasAgent
Conceito fonte	Algorithm
Cardinalidade Fonte	(1,1)
Conceito Alvo	Agent
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isAgentOf

Nome da Relação	hasAlgorithm
Conceito fonte	Policy
Cardinalidade Fonte	(1,1)
Conceito Alvo	Algorithm
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isAlgorithmOf

Nome da Relação	hasBilling
Conceito fonte	Client
Cardinalidade Fonte	(1,1)
Conceito Alvo	Billing
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isBillingOf

Nome da Relação	hasBillingHistory
Conceito fonte	Billing
Cardinalidade Fonte	(1,n)
Conceito Alvo	BillingHistory
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isBillingHistoryOf

Nome da Relação	hasClient
Conceito fonte	Provider
Cardinalidade Fonte	(1,n)
Conceito Alvo	Client
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isClientOf

Nome da Relação	hasMachine
Conceito fonte	Client
Cardinalidade Fonte	(1,n)

Conceito Alvo	Machine
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isMachineOf

Nome da Relação	hasMachineCapacity
Conceito fonte	Machine
Cardinalidade Fonte	(1,1)
Conceito Alvo	MachineCapacity
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isMachineCapacityOf

Nome da Relação	hasManagement
Conceito fonte	Client
Cardinalidade Fonte	(1,n)
Conceito Alvo	Management
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isManagementOf

Nome da Relação	hasManagementTools
Conceito fonte	Management
Cardinalidade Fonte	(1,n)
Conceito Alvo	ManagementTools
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isManagementToolsOf

Nome da Relação	hasMonitoring
Conceito fonte	Machine
Cardinalidade Fonte	(1,1)
Conceito Alvo	AgentMonitor
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isMonitoredBy

Nome da Relação	hasPeriodStart
Conceito fonte	PowerPolicy
Cardinalidade Fonte	(1,n)
Conceito Alvo	Period
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isPeriodOf

Nome da Relação	hasPeriodEnd
Conceito fonte	PowerPolicy

Cardinalidade Fonte	(1,n)
Conceito Alvo	Period
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isPeriodOf

Nome da Relação	hasPolicy
Conceito fonte	Management
Cardinalidade Fonte	(1,1)
Conceito Alvo	Policy
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isPolicyOf

Nome da Relação	hasPowerStatus
Conceito fonte	RealMachine
Cardinalidade Fonte	(1,1)
Conceito Alvo	PowerStatus
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isPowerStatusOf

Nome da Relação	hasProtocol
Conceito fonte	Network
Cardinalidade Fonte	(1,n)
Conceito Alvo	Protocol
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isProtocolOf

Nome da Relação	hasProvider
Conceito fonte	Client
Cardinalidade Fonte	(1,1)
Conceito Alvo	Provider
Cardinalidade Alvo	(1,n)
Propriedades Matemáticas	-
Relação inversa	isProviderOf

Nome da Relação	hasQoSAttribute
Conceito fonte	QualityOfService
Cardinalidade Fonte	(1,n)
Conceito Alvo	QoSAttribute
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isQoSAttributeOf

Nome da Relação	hasQualityOfService
-----------------	----------------------------

Conceito fonte	VirtualMachine
Cardinalidade Fonte	(1,1)
Conceito Alvo	QualityOfService
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isQualityOfService

Nome da Relação	hasRealMachine
Conceito fonte	Provider
Cardinalidade Fonte	(1,n)
Conceito Alvo	RealMachine
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isRealMachineOf

Nome da Relação	hasResourceManufacturer
Conceito fonte	Resources
Cardinalidade Fonte	(1,1)
Conceito Alvo	ResourceManufacturer
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isResourceManufacturerOf

Nome da Relação	hasResourceModel
Conceito fonte	Resources
Cardinalidade Fonte	(1,1)
Conceito Alvo	ResourceModel
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isResourceModelOf

Nome da Relação	hasResources
Conceito fonte	MachineCapacity
Cardinalidade Fonte	(1,n)
Conceito Alvo	Resources
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isResourcesOf

Nome da Relação	hasROI
Conceito fonte	ROIPolicy
Cardinalidade Fonte	(1,1)
Conceito Alvo	ROI
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	-

Nome da Relação	hasScheduledDays
Conceito fonte	ScheduledPolicy
Cardinalidade Fonte	(1,1)
Conceito Alvo	hasScheduledDays
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	isScheduledDaysOf

Nome da Relação	hasSLA
Conceito fonte	SLAPolicy
Cardinalidade Fonte	(1,1)
Conceito Alvo	QualityOfService
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	-

Nome da Relação	hasSoftware
Conceito fonte	MachineCapacity
Cardinalidade Fonte	(1,n)
Conceito Alvo	Software
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isSoftwareOf

Nome da Relação	hasUser
Conceito fonte	Client
Cardinalidade Fonte	(1,n)
Conceito Alvo	User
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isUserOf

Nome da Relação	hasVirtualMachine
Conceito fonte	Client
Cardinalidade Fonte	(1,n)
Conceito Alvo	VirtualMachine
Cardinalidade Alvo	(n,1)
Propriedades Matemáticas	-
Relação inversa	isVirtualMachineOf

Nome da Relação	hasVirtualMachineStatus
Conceito fonte	VirtualMachine
Cardinalidade Fonte	(1,1)
Conceito Alvo	VirtualMachineStatus
Cardinalidade Alvo	(1,1)
Propriedades Matemáticas	-
Relação inversa	-

Infrastructure as a Service Ontology – IaaSOnt

Tabela de Instâncias (Instance Table)¹

Instância	Atributo	Valor
[SubClass] Management [Number] (Gerenciamento)	hasPolicy	Policy [Number]

Instância	Atributo	Valor
[SubClass] Policy [Number] (Política)	hasQoS	QoS [Number]

Instância	Atributo	Valor
Client [Number] (Clientes)	hasManagement	Management [Number]
	hasId	(Integer) – ID do Cliente
	hasName	(String) – Nome do Cliente
	hasEmail	(String) – Email do Cliente
	hasPassword	(String) – Senha do Cliente
	hasMachine	Machine [Number]

Instância	Atributo	Valor
Provider [Number] (Provedores)	hasId	(Integer) – ID do Provedor
	hasName	(String) – Nome do Provedor
	hasEmail	(String) – Email do Provedor
	hasPassword	(String) – Senha do Provedor
	hasCliente	Client [Number]

Instância	Atributo	Valor
Machine [Number] (Máquinas (Real ou Virtual - Serviços))	hasPowerStatus	[SubClass]PowerStatus
	hasID	(Integer) – ID da Máquina
	hasCPULoad	(Decimal) – Carga da CPU
	hasMemoryLoad	(Decimal) – Carga da Memória
	hasDiskLoad	(Decimal) – Carga de Disco
	hasCost	(Decimal) – Custo da Máquina/h
	hasCapacity	(Decimal) – Capacidade da Máq.
	hasResponseTime	(Decimal) – Tempo de Resposta
	hasAvailability	(Decimal) – Disponibilidade

Instância	Atributo	Valor
[SubClass] QoS [Number] (QoS – Qualidade de Serviço)	hasID	(Integer) – ID da QoS
	hasQoSAttribute	QoSAttribute [Number]

Instância	Atributo	Valor
QoSAttribute [Number] (Atributo de QoS)	has[atributo]	(Decimal) – Valor do Atributo de QoS (Cost, Capacity, ResponseTime, Availability)

¹ Não são listadas todas as instâncias e nem todas as propriedades para evitar uma grande repetição de informação. Apenas são listadas uma instância de cada classe de maior relevância para o entendimento da ontologia.

Infrastructure as a Service Ontology – IaaSOnt

Tabelas de Atributos das Instâncias (Instance-Attribute Table)

Nome do atributo	hasDeveloper
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasInvestment
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasType
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasFinalValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasActualValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasAgentClass
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasAgentName
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasAmount
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasArchitecture
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-

Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBandwidth
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingAlert
Tipo de Valor (Tipo de dado)	Boolean
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingAlertEmail
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingDate
Tipo de Valor (Tipo de dado)	dateTime
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingDiaryCost
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingDiaryHoursUsage
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingHistoryCurrentValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingHistoryEndDate
Tipo de Valor (Tipo de dado)	Datetime
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBillingHistoryStartDate
Tipo de Valor (Tipo de dado)	Datetime
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasBrandName
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)
Nome do atributo	hasCost
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasCPULoad
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasDay
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasDayOfMonth
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasDescription
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasDiskLoad
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasEmail
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasFileSystem
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasFixedDiaryValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-

Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasFrequency
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasHorizontalScalingLimit
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasHorizontalScalingValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasHour
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasId
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasInitialValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasLastUpdate
Tipo de Valor (Tipo de dado)	dateTime
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasLogin
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasManufacturerEmail
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,n)

Nome do atributo	hasMemoryLoad
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasMonth
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasName
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasNetworkLoad
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasNumberOfCores
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasNumberOfRealMachines
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasNumberOfVirtualMachines
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasParameter
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasPassword
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasPowerConsume
Tipo de Valor (Tipo de dado)	Decimal

Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasPowerConsumption
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasRateApplied
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasReturn
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasSerial
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasSize
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasStorageMaxSize
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasStorageMinSize
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasStorageSize
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasURI
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-

Cardinalidade	(1,1)
----------------------	-------

Nome do atributo	has
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasVersion
Tipo de Valor (Tipo de dado)	String
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasVerticalScalingLimit
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasVerticalScalingValue
Tipo de Valor (Tipo de dado)	Decimal
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasWeekDay
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	hasYear
Tipo de Valor (Tipo de dado)	nonNegativeInteger
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)

Nome do atributo	isHyperThreading
Tipo de Valor (Tipo de dado)	Boolean
Intervalo de valores	-
Valor Padrão	-
Cardinalidade	(1,1)