Projeto e avaliação de um *broker* como agente de
intermediação e QoS em uma nuvem
computacional híbrida

Mário Henrique de Souza Pardo

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: ____/____/____

Assinatura: _____

Mário Henrique de Souza Pardo

Projeto e avaliação de um *broker* como agente de intermediação e QoS em uma nuvem computacional híbrida

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

Orientadora: Profa. Dra. Regina Helena Carlucci Santana

USP – São Carlos

Agosto de 2016

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

P226p Pardo, Mário Henrique de Souza
Projeto e avaliação de um broker como elemento de
intermediação e QoS em uma nuvem computacional
híbrida / Mário Henrique de Souza Pardo; orientadora
Regina Helena Carlucci Santana. -- São Carlos, 2016.
200 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, 2016.

1. Computação em Nuvem. 2. Cloud Broker. 3.
Escalonamento de tarefas. 4. QoS. 5. Avaliação de
Desempenho. I. Santana, Regina Helena Carlucci,
orient. II. Título.

Mário Henrique de Souza Pardo

*Design and evaluation of a broker as QoS and
intermediation agent in hybrid cloud computing*

*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: Profa. Dra. Regina Helena Carlucci
Santana*

USP – São Carlos

August 2016

Dedicatória

---

*Dedico este trabalho aos meus queridos pais,
Francisco e Laura, por tudo que fizeram por
mim para que meus sonhos fossem possíveis.*

Agradecimentos

Saber reconhecer a importância dos que nos auxiliaram na caminhada é muito importante. Quero manifestar os meus agradecimentos às pessoas que de algum modo me ajudaram durante o desenrolar destes quase cinco anos de jornada de doutorado.

Primeiramente, um agradecimento especial à minha orientadora Profa. Dra. Regina Helena Carlucci Santana, por quem tive a honra de ser orientado, obtendo valiosos ensinamentos proveitosos e competentes. Indo além, agradeço-lhe, querida mestre, pela nossa amizade, pelos incentivos, paciência e por depositar fé no meu trabalho.

Quero também manifestar o meu agradecimento aos queridos professores Edson dos Santos Moreira, Kalinka Regina Lucas Jaquie Castelo Branco, Sarita Mazzini Bruschi, Marcos José Santana, Francisco José Mônaco, Julio Cezar Estrella e Paulo Sérgio Lopes de Souza, do Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, campus de São Carlos, com os quais tive a satisfação e honra de conviver ao cursar as disciplinas, workshops e seminários do grupo de pesquisa em Sistemas Distribuídos e Programação Concorrente.

Não poderia deixar de agradecer a dois colegas de doutorado com os quais tive bastante afinidade ao executar meu trabalho de doutorado integrado ao deles. Refiro-me aos colegas e amigos Adriana Molina Centurion e Paulo Sérgio Franco Eustáquio, com os quais trabalhei intensamente durante os anos 2013, 2014 e 2015 na construção do projeto que daria forma à arquitetura CloudSim BEQoS, a qual poderá, no futuro, beneficiar muitos outros colegas que queiram dar continuidade à pesquisa de sistemas de computação em nuvem pelo grupo SDPC da USP.

Agradeço a todos os colegas do LaSDPC pelo companheirismo.

A todos vocês o meu sincero muito obrigado!

Resumo

A presente tese de doutorado propõe uma arquitetura de *cloud broker* para ambientes de computação em nuvem híbrida. Um *cloud broker* tem o objetivo de executar a mediação entre clientes e provedores, recebendo requisições dos clientes e encaminhando-as ao serviço do provedor que melhor se adaptar aos requisitos de qualidade de serviço (QoS) solicitados. A arquitetura de *broker* de serviços com QoS proposta denomina-se *QBroker*, características de implementação de seu modo de operação bem como sua interação com os recursos virtuais de um ambiente de nuvem são apresentadas. O modelo de nuvem considerado foi o de nuvem híbrida com uma caracterização de arquitetura orientada a serviços (SOA) na qual serviços remotos são disponibilizados aos clientes. A política de escalonamento de tarefas desenvolvida para o *QBroker* foi a de intermediação de serviços, considerando tratativas de QoS, diferenciação das instâncias de serviços (SOA) e alocação dinâmica de serviços. Além disso, toda a caracterização do modo de operação do *QBroker* foi baseada no conceito de intermediação do modelo de referência de nuvem do NIST. O componente *QBroker* foi introduzido numa arquitetura de computação em nuvem BEQoS (*Bursty Energy and Quality of Service*), desenvolvida no Laboratório de Sistemas Distribuídos e Programação Concorrente do ICMC-USP de São Carlos. Avaliações de desempenho para a implementação da arquitetura *QBroker* foram conduzidas por meio de programas de simulação com uso da API do simulador CloudSim e da arquitetura CloudSim-BEQoS. Três cenários experimentais foram avaliados e, segundo a análise de resultados efetuada, foi possível validar que as características arquiteturais implementadas no *QBroker* resultaram em significativo impacto nas variáveis de resposta consideradas. Assim, foi possível comprovar que o uso do *QBroker* como mecanismo de mediação em ambientes de nuvem híbrida com SOA promoveu ganhos em desempenho para o sistema de nuvem e permitiu melhoria na qualidade dos serviços oferecidos.

Palavras-chave: Computação em Nuvem, escalonamento de tarefas, algoritmo, intermediação, SOA, serviços, simulação, CloudSim, avaliação de desempenho, QoS, alocação dinâmica, diferenciação de serviço, nuvem híbrida.

Abstract

This doctoral thesis proposes a cloud broker architecture for hybrid cloud computing environments. A cloud broker aims to perform mediation between clients and providers, receiving customer requests and forwarding them to the service provider that best suits the requested QoS requirements. The broker architecture services with QoS proposal is called QBroker. Implementation features of its mode of operation as well as its interaction with the virtual resources from a cloud environment are presented. The cloud deployment model was considered a hybrid cloud with a characterization of service-oriented architecture (SOA) in which remote services are available to customers. The task scheduling policy developed for QBroker was the intermediation of services, considering negotiations of QoS, differentiation of services instances and dynamic allocation of services. Moreover, the entire characterization of QBroker operation mode is based on the intermediation concept of the NIST cloud reference model. The QBroker component was introduced into a cloud computing architecture BEQoS (Bursty, Energy and Quality of Service), developed in the Laboratory of Distributed Systems and Concurrent Programming at ICMC-USP. Performance evaluations analysis the of results of QBroker architecture were conducted through simulation programs using the CloudSim simulator API and CloudSim-BEQoS architecture. Three experimental scenarios were evaluated and, according to analysis of the results, it was possible to validate that the architectural features implemented in QBroker resulted in significant impact on response variables considered. Thus, it was possible to prove that the use of QBroker as mediation mechanism in hybrid cloud environments with SOA promoted performance gains for the cloud system and allowed improvement in the quality of services offered.

Keywords: *Cloud computing, task scheduling, algorithm, intermediation, SOA, services, simulation, CloudSim, performance evaluation, QoS, dynamic allocation, service differentiation, hybrid cloud.*

Sumário

1.	INTRODUÇÃO.....	31
1.1	CONSIDERAÇÕES INICIAIS.....	31
1.2	CONTEXTUALIZAÇÃO.....	32
1.3	MOTIVAÇÃO.....	33
1.4	HIPÓTESE.....	34
1.5	OBJETIVO.....	35
1.6	METODOLOGIA.....	36
1.7	ESTRUTURA DO DOCUMENTO.....	38
2.	COMPUTAÇÃO EM NUVEM.....	41
2.1	CONSIDERAÇÕES INICIAIS.....	41
2.2	ARQUITETURA DE NUVEM COMPUTACIONAL DO NIST.....	42
2.2.1	<i>Cliente de Serviços de Nuvem</i>	43
2.2.2	<i>Provedor de Serviços de Nuvem</i>	44
2.2.3	<i>Intermediador de Nuvem</i>	44
2.2.4	<i>Auditor de Nuvem</i>	46
2.2.5	<i>Transportador de Nuvem</i>	46
2.2.6	<i>Considerações adicionais sobre a Arquitetura de Nuvem do NIST</i>	46
2.3	MODELOS DE IMPLANTAÇÃO.....	48
2.3.1	<i>Nuvem Privada</i>	49
2.3.2	<i>Nuvem Pública</i>	49
2.3.3	<i>Nuvem Híbrida</i>	50
2.3.4	<i>Nuvem Comunitária</i>	50
2.3.5	<i>Considerações adicionais sobre modelos de implantação</i>	50
2.4	MODELOS DE SERVIÇO.....	52
2.4.1	<i>Software como Serviço</i>	53
2.4.2	<i>Plataforma como Serviço</i>	53
2.4.3	<i>Infraestrutura como Serviço</i>	54
2.5	ESTRUTURA DE NUVEM.....	54
2.5.1	<i>Nuvem Computacional Homogênea</i>	55

2.5.2	<i>Nuvem Computacional Heterogênea</i>	55
2.6	RECURSOS DE NUVEM COMPUTACIONAL	55
2.7	ESCALONAMENTO EM NUVEM COMPUTACIONAL.....	57
2.7.1	<i>Tipos de escalonamento para nuvem computacional</i>	59
2.7.2	<i>Alocação de Recursos em Nuvem Computacional</i>	63
2.8	QUALIDADE DE SERVIÇO EM NUVEM COMPUTACIONAL.....	66
2.9	SIMULAÇÃO EM COMPUTAÇÃO EM NUVEM.....	68
2.10	CONSIDERAÇÕES FINAIS	70
3.	ARQUITETURA ORIENTADA A SERVIÇOS	73
3.1	CONSIDERAÇÕES INICIAIS	73
3.2	SERVIÇOS WEB.....	74
3.2.1	<i>Protocolo de Comunicação SOAP</i>	76
3.2.2	<i>Protocolo de Descoberta de Serviço WSDL</i>	76
3.2.3	<i>Serviço de Diretório UDDI</i>	77
3.2.4	<i>Composição de Serviços</i>	78
3.3	QUALIDADE DE SERVIÇO.....	79
3.4	CONSIDERAÇÕES FINAIS	80
4.	CLOUD BROKERS	81
4.1	COMUNICAÇÃO EM BROKERS DE NUVEM COMPUTACIONAL.....	82
4.2	MODOS DE OPERAÇÃO DE CLOUD BROKER	83
4.3	TRABALHOS RELACIONADOS COM <i>CLOUD BROKERS</i>	84
4.4	CONSIDERAÇÕES FINAIS	88
5.	ARQUITETURA CLOUDSIM-BEQOS E QBROKER.....	91
5.1	CONSIDERAÇÕES INICIAIS	91
5.2	ARQUITETURA CLOUDSIM-BEQOS	92
5.3	CAMADAS DA ARQUITETURA CLOUDSIM-BEQOS	92
5.4	CARACTERÍSTICAS ARQUITETURAIS	98
5.4.1	<i>Detalhamento das classes de objeto da arquitetura CloudSim-BEQoS</i>	100
5.5	ARQUITETURA DO QBROKER (QOS BROKER).....	105
5.5.1	<i>Camadas da Arquitetura QBroker</i>	107
5.6	CONSIDERAÇÕES FINAIS	113
6.	AVALIAÇÃO DE DESEMPENHO DA ARQUITETURA QBROKER	115
6.1	CONSIDERAÇÕES INICIAIS.....	115

6.2	DESCRIÇÃO DO AMBIENTE DE TESTES	115
6.2.1	<i>Configurações dos Datacenters.....</i>	118
6.2.2	<i>Configurações das Máquinas Virtuais.....</i>	119
6.2.3	<i>Configuração da Demanda de Serviços e dos Clientes.....</i>	120
6.2.4	<i>Descrição das variáveis de resposta .....</i>	121
6.3	ANÁLISE DOS RESULTADOS EXPERIMENTAIS.....	122
6.3.1	<i>Cenário 1: Demonstração de QoS.....</i>	123
6.3.2	<i>Cenário 2: Alocação Estática de Serviços.....</i>	132
6.3.3	<i>Cenário 3: Alocação Dinâmica de Serviços.....</i>	140
6.4	CONSIDERAÇÕES FINAIS	150
7.	CONCLUSÕES	153
7.1	CONTRIBUIÇÕES.....	156
7.2	PRODUÇÃO CIENTÍFICA	157
7.3	TRABALHOS FUTUROS	158
	REFERÊNCIAS	163
	APÊNDICE A	177
	MODIFICAÇÕES EFETUADAS NA CLASSE CLOUDLET.....	177
	APÊNDICE B	179
	CODIFICAÇÃO DE UMA CLASSE DE EXPERIMENTO DE SIMULAÇÃO	179
	APÊNDICE C	187
	SIMULAÇÃO COM ARQUITETURA CLOUDSIM-BEQOS E QBROKER.....	187
	<i>Constantes e variáveis de instância da classe QBroker .....</i>	187
	<i>Construtor de Classe .....</i>	189
	<i>Método para processamento de eventos.....</i>	189
	<i>Evento de Chegada de Requisição .....</i>	190
	<i>Método para execução de política de escalamento .....</i>	191
	<i>Método de intermediação .....</i>	194

Lista de Figuras

FIGURA 2.1: ARQUITETURA DE REFERÊNCIA DE COMPUTAÇÃO EM NUVEM DO NIST. (BOHN <i>ET AL.</i> , 2011)	
.....	ERRO! INDICADOR NÃO DEFINIDO.
FIGURA 2.1: ARQUITETURA DE REFERÊNCIA DE COMPUTAÇÃO EM NUVEM DO NIST. (BOHN <i>ET AL.</i> , 2011)	43
FIGURA 2.2: MODELOS DE IMPLEMENTAÇÃO DE COMPUTAÇÃO EM NUVEM. ADAPTADO DE	
(DUMMIES.COM, 2016).....	51
FIGURA 2.3: MODELOS DE SERVIÇO DE COMPUTAÇÃO EM NUVEM. ADAPTADO DE (BUMPUS, 2013)	53
FIGURA 2.4: TIPOS DE REQUISIÇÃO EM ESCALONAMENTO DE NUVENS COMPUTACIONAIS. (AUTORIA PRÓPRIA)	
.....	59
FIGURA 3.1: DINÂMICA DE COMUNICAÇÃO DO AMBIENTE DE SERVIÇOS WEB. (TANENBAUM; STEEN, 2006)	
.....	75
FIGURA 4.1: DIAGRAMA DE SEQUÊNCIA DE COMUNICAÇÃO EM SISTEMAS DE NUVEM. ADAPTADO DE (RAJ,	
2011).....	82
FIGURA 5.1: VISÃO EM CAMADAS DA ARQUITETURA CLOUDSIM-BEQOS. (AUTORIA PRÓPRIA).....	94
FIGURA 5.2: DIAGRAMA DE SEQUÊNCIA COM COMUNICAÇÃO ENTRE OBJETOS DA ARQUITETURA CLOUDSIM-	
BEQOS. (AUTORIA PRÓPRIA)	97
FIGURA 5.3: DIAGRAMA DE CLASSES COM COMPONENTES DA ARQUITETURA CLOUDSIM-BEQOS E	
ASSOCIAÇÕES COM COMPONENTES DA API PADRÃO DO CLOUDSIM. (AUTORIA PRÓPRIA).....	100
FIGURA 5.4: ARQUITETURA EM CAMADAS DO QBROKER. (AUTORIA PRÓPRIA).....	109
FIGURA 5.5: DIAGRAMA DE ATIVIDADES DO ALGORITMO DE INTERMEDIACÃO DO QBROKER. (AUTORIA	
PRÓPRIA)	112
FIGURA 6.1: AMBIENTE DE TESTES MODELADO PARA OS EXPERIMENTOS DE SIMULAÇÃO.....	116
FIGURA 6.2: GRÁFICO DE TEMPO MÉDIO DE RESPOSTA DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.....	124
FIGURA 6.3: GRÁFICO DE PORCENTAGEM MÉDIA DE REQUISIÇÕES PROCESSADAS DO CENÁRIO DE	
DEMONSTRAÇÃO DE QOS.....	125
FIGURA 6.4: GRÁFICO DE PORCENTAGEM MÉDIA DE REQUISIÇÕES NÃO ATENDIDAS DO CENÁRIO DE	
DEMONSTRAÇÃO DE QOS.....	125
FIGURA 6.5: GRÁFICO DE TEMPO MÉDIO DE FILA DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.	126
FIGURA 6.6: GRÁFICO DO NÚMERO MÉDIO DE REQUISIÇÕES PROCESSADAS DO CENÁRIO DE DEMONSTRAÇÃO	
DE QOS.	127

FIGURA 6.7: GRÁFICO DE NÚMERO MÉDIO DE REQUISIÇÕES NÃO ATENDIDAS DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.	128
FIGURA 6.8: DISTRIBUIÇÃO DAS REQUISIÇÕES EM PORCENTAGEM POR TIPO DE PROVEDOR DE NUVEM DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.....	129
FIGURA 6.9: PORCENTAGEM DE USO DE RECURSOS DE NUVEM DOS EXPERIMENTOS A E B DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.....	130
FIGURA 6.10: PORCENTAGEM DE USO DE RECURSOS DE NUVEM DOS EXPERIMENTOS C E D DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.....	130
FIGURA 6.11: PORCENTAGEM DE USO DE RECURSOS DE NUVEM DOS EXPERIMENTOS E E F DO CENÁRIO DE DEMONSTRAÇÃO DE QOS.....	131
FIGURA 6.12: GRÁFICO DE TEMPO MÉDIO DE RESPOSTA DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.	133
FIGURA 6.13: GRÁFICO DE PORCENTAGEM MÉDIA DE REQUISIÇÕES PROCESSADAS DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.	135
FIGURA 6.14: GRÁFICO DE PORCENTAGEM MÉDIA DE REQUISIÇÕES NÃO ATENDIDAS DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.	135
FIGURA 6.15: GRÁFICO DE TEMPO MÉDIO DE FILA DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.....	136
FIGURA 6.16: GRÁFICO DE NÚMERO MÉDIO DE REQUISIÇÕES PROCESSADAS DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.	138
FIGURA 6.17: GRÁFICO DE NÚMERO MÉDIO DE REQUISIÇÕES NÃO ATENDIDAS DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.	139
FIGURA 6.18: GRÁFICO DE TEMPO MÉDIO DE RESPOSTA DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.	141
FIGURA 6.19: GRÁFICO DE PORCENTAGEM MÉDIA DE REQUISIÇÕES PROCESSADAS DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.	142
FIGURA 6.20: GRÁFICO DE PORCENTAGEM MÉDIA DE REQUISIÇÕES NÃO ATENDIDAS DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.....	143
FIGURA 6.21: GRÁFICO DE TEMPO MÉDIO DE FILA DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.	144
FIGURA 6.22: GRÁFICO DE NÚMERO MÉDIO DE REQUISIÇÕES PROCESSADAS DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.	145
FIGURA 6.23: GRÁFICO DE NÚMERO MÉDIO DE REQUISIÇÕES NÃO ATENDIDAS DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.	145
FIGURA 6.24: DISTRIBUIÇÃO EM PORCENTAGEM DAS REQUISIÇÕES POR TIPO DE PROVEDOR DE NUVEM.....	147
FIGURA 6.25: PORCENTAGEM DO USO DE RECURSOS DE NUVEM DOS EXPERIMENTOS C', D', E' E F'.....	148

FIGURA 6.26: PORCENTAGEM DO USO DE RECURSOS DE NUVEM DOS EXPERIMENTOS C'', D'', E'' E F''. 149

Lista de Quadros

QUADRO 3.1: EXEMPLO DE MENSAGEM SOAP.	ERRO! INDICADOR NÃO DEFINIDO.
QUADRO 3.2: EXEMPLO DE MENSAGEM WSDL COM DESCRIÇÃO DE SERVIÇO WEB.	ERRO! INDICADOR NÃO DEFINIDO.
QUADRO B.1: EXEMPLO DE DECLARAÇÃO DE CLASSE DE OBJETOS PARA PROGRAMA DE SIMULAÇÃO.	179
QUADRO B.2: INICIALIZAÇÃO DE OBJETOS DE BASE DO SIMULADOR CLOUDSIM.	180
QUADRO B.3: DECLARAÇÃO E INICIALIZAÇÃO DE INSTÂNCIA DA CLASSE QBROKER.	180
QUADRO B.4: DEFINIÇÃO DE OBJETOS DE CONFIGURAÇÃO E DECLARAÇÃO DAS INSTÂNCIAS DE <i>DATACENTER</i>	181
QUADRO B.5: DEFINIÇÃO DE OBJETOS DE CONFIGURAÇÃO E CRIAÇÃO DE LISTAS DE VMs.	182
QUADRO B.6: CRIAÇÃO DE LISTAS DE VMs NO PROGRAMA DE SIMULAÇÃO.	182
QUADRO B.7: ATRIBUIÇÃO DAS LISTAS DE VMs GERADAS PARA O QBROKER.	182
QUADRO B.8: DEFINIÇÃO DE OBJETO REFERENTE A TABELA DE DEMANDA DE SERVIÇOS (<i>SERVICEDEMANDTABLE</i>).....	183
QUADRO B.9: INSTÂNCIA DE <i>CLOUDLET</i> COMO PROTÓTIPO PARA GERAÇÃO DE REQUISIÇÕES EM TEMPO DE SIMULAÇÃO.	183
QUADRO B.10: CRIAÇÃO E CONFIGURAÇÃO DAS INSTÂNCIAS DE CLIENTES VIRTUAIS NO PROGRAMA DE SIMULAÇÃO.	185
QUADRO B.11: INSTANCIAMENTO DE COMPONENTES MONITORES DA ARQUITETURA CLOUDSIM-BEQOS.	185
QUADRO B.12: COMANDOS DE CONFIGURAÇÃO DE TEMPO DE SIMULAÇÃO E INÍCIO DO <i>RUNTIME</i> DO SIMULADOR CLOUDSIM.	186
QUADRO B.13: COMANDOS DE FINALIZAÇÃO DO PROGRAMA DE SIMULAÇÃO E LEITURA OU GRAVAÇÃO DE DADOS E INFORMAÇÕES.	186
QUADRO C.1: ASSINATURA DA CLASSE QBROKER.	187
QUADRO C.2: CONSTANTES DA CLASSE QBROKER.	188

Lista de Tabelas

TABELA 4-1: RESUMO DAS PROPOSTAS DE ARQUITETURAS DE <i>CLOUD BROKER</i>	89
TABELA 6-1: CONFIGURAÇÕES DE <i>HOST</i> DA NUVEM PRIVADA.	118
TABELA 6-2: CONFIGURAÇÕES DE <i>HOST</i> DA NUVEM PÚBLICA.	118
TABELA 6-3: CONFIGURAÇÕES DE <i>VMs</i> DO <i>DATACENTER</i> DA NUVEM PRIVADA.	119
TABELA 6-4: CONFIGURAÇÕES DE <i>VMs</i> DO <i>DATACENTER</i> DA NUVEM PÚBLICA.	120
TABELA 6-5: CONFIGURAÇÃO DA DEMANDA DE SERVIÇOS.....	120
TABELA 6-6: DISTRIBUIÇÃO DA GERAÇÃO ALEATÓRIA DE SERVIÇOS PARA AS REQUISIÇÕES DAS ENTIDADES CLIENTE.....	121
TABELA 6-7: PLANEJAMENTO DE EXPERIMENTOS DO CENÁRIO DE EVIDENCIAÇÃO DE QoS.	123
TABELA 6-8: PLANEJAMENTO DE EXPERIMENTOS DO CENÁRIO DE ALOCAÇÃO ESTÁTICA DE SERVIÇOS.....	132
TABELA 6-9: PLANEJAMENTO DE EXPERIMENTOS DO CENÁRIO DE ALOCAÇÃO DINÂMICA DE SERVIÇOS.....	140
TABELA A-1: DESCRIÇÃO DOS CAMPOS DE INSTÂNCIA ADICIONADOS NA CLASSE <i>CLOUDLET</i>	177

Lista de Abreviaturas

- AAT:** *Autonomous Adaptive Thresholding*
- ADMB:** *Adaptive Decision Making Broker*
- ALS:** *Adaptative List Scheduling*
- AMMS:** *Adaptive Min-Min Scheduling*
- API:** *Application Programming Interface*
- B2B:** *Business-to-Business*
- B2C:** *Business-to-Customer*
- BCM:** *Broker Cloud Management*
- BEQoS:** *Bursty, Energy and Quality of Service*
- BP:** *Business Process*
- BVCF:** *Broker Virtual Machine Communication Framework*
- C2C:** *Customer-to-Customer*
- CEP:** *Código de Endereçamento Postal*
- CFP:** *Call For Proposal*
- CIS:** *Cloud Information Service*
- CORBA:** *Common Object Request Broker Architecture*
- CPU:** *Central Processing Unit*
- DCLS:** *Dynamic Cloud List Scheduling*
- DCMMS:** *Dynamic Cloud Min–Min Scheduling*
- DTP:** *Deadline-aware Tasks Packing*
- EC2:** *Elastic Compute Cloud*
- ESB:** *Enterprise Service Bus*

FCFS: *First-Come-First-Serve*

FIFO: *First-In-First-Out*

G2C: *Government-to-Citizen*

GPU: *Graphics Processing Unit*

HPC: *High Performance Computing*

HTTP: *Hyper Text Transfer Protocol*

HTTPS: *Hyper Text Transfer Protocol Secure*

IaaS: *Infrastructure as a Service*

ICMC: Instituto de Ciências Matemáticas e de Computação

ICMS: *Inter-Cloud Meta-Scheduling*

ID: Identificador

IDE: *Integrated Development Environment*

ILP: *Integer Linear Programming*

LFF: *List and First-Fit*

MAP: *Markovian Arrival Process*

MI: *Millions of Instructions*

MIPS: *Millions Instruction Per Second*

NIST: *National Institute of Standards and Technology*

ORCF: *Optimum Route Cost Finder*

PaaS: *Platform as a Service*

QoS: *Quality of Service*

RAM: *Random Access Memory*

RMI: *Remote Method Invocation*

RPC: *Remote Procedure Call*

RTT: *Round Trip Time*

SaaS: *Software as a Service*

SDPC: Sistemas Distribuídos e Programação Concorrente

SLA: *Service Level Agreement*

SMI: *Service Measurement Index*

SMP: *Symmetric MultiProcessing*

SMTP: *Simple Mail Transfer Protocol*

SOA: *Service Oriented Architecture*

SOAP: *Simple Object Access Protocol*

TOPSIS: *Technique for Order Preference by Similarity to Ideal Solution*

UDDI: *Universal Description, Discovery and Integration*

UML: *Unified Modeling Language*

URL: *Uniform Resource Locator*

USP: Universidade de São Paulo

VCB: *Virtual Cloud Bank*

VM: *Virtual Machine*

WSDL: *Web Services Description Language*

XML: *eXtensible Markup Language*

1. Introdução

1.1 Considerações Iniciais

Nos últimos anos as aplicações contemporâneas para Web têm aderido cada vez mais ao paradigma da Arquitetura Orientada a Serviços (do inglês, *Service Oriented Architecture* ou SOA). Isto se dá pelo fato de que os projetos atuais de aplicativos estão dando preferência aos padrões abertos, de modo a aumentar as possibilidades de expansão, de acoplamento de novas funcionalidades ao software e de interoperabilidade entre sistemas, tornando-os, de certo modo, independentes de plataforma. Com relação às implementações de SOA, destacam-se os serviços Web, os quais estão se firmando como solução bem aceita por desenvolvedores para criar aplicações para Internet (DHARA, 2015).

Alguns tipos de aplicações Web como E-Commerce, portais de notícias, redes sociais e *websites* de compartilhamento de vídeos e conteúdo multimídia, podem necessitar de grande poder computacional quando recebem um grande volume de acessos de usuários (ALAM *et al.*, 2012). Para que seja possível atender à tal demanda, novos paradigmas de computação distribuída estão emergindo. A computação em nuvem, a qual traz consigo o provimento de recursos de computação com base em modelos de serviço e várias características dinâmicas no provimento de recursos tem sido muito bem aceita pelo mercado. A computação em nuvem ganhou forte adesão devido, entre várias de suas características positivas, ao modelo de cobrança denominado *pay-as-you-go* (GILL, 2015).

O padrão de serviços Web e os sistemas de computação em nuvem estão convergindo de tal forma que as nuvens computacionais proveem infraestrutura, plataforma e aplicações como serviços de nuvem disponibilizados como serviços Web (MOHTASHIM *et al.*, 2015). Tendo-se por princípio que serviços Web também podem ser disponibilizados como *software* em nuvens computacionais, pode-se afirmar que SOA poderá ser tornar, no futuro, uma arquitetura padrão das aplicações modernas de Internet em ambientes de computação em nuvem.

1.2 Contextualização

Atualmente são adotados por empresas e governos os modelos de relacionamento ou negócios eletrônicos, tais como B2B (*Business-to-Business*), B2C (*Business-to-Customer*), C2C (*Customer-to-Customer*), G2C (*Government-to-Citizen*), entre outros. Para cobrir a demanda, as aplicações baseadas em serviços Web estão se firmando no mercado como solução viável para auxiliar na implementação dos modelos, pois, como serviços Web são implementações SOA, acabam por prover características interessantes como fraco acoplamento, transparência, interoperabilidade e independência de plataforma (LIU, X. *et al.*, 2015).

Paradigma emergente no mercado de computação, os sistemas de nuvem computacional são alvo de muitas pesquisas para alcançar os objetivos lançados pelas características de funcionamento que esses sistemas possuem, isto é, obter eficiência na alocação de recursos sob demanda (PIRAGHAJ; CALHEIROS, 2014), cumprir a contratos de nível de serviço (HUSSEIN; MOREAU; SASSONE, 2015) e observar os requisitos de qualidade de serviço (SANDHU; SOOD, 2014) conforme as requisições dos usuários. Esses objetivos podem ser alcançados por meio de uma das principais características apresentadas pela computação em nuvem, a chamada elasticidade computacional (COUTINHO *et al.*, 2014). Elasticidade computacional relaciona-se com a capacidade do sistema de computação em alocar quantidades de recursos variáveis, de forma automática, visando alcançar metas como as propostas para sistemas de computação em nuvem.

Um elemento importante e comum entre os padrões de arquiteturas para serviços Web e sistemas de computação em nuvem é o *cloud broker*, o qual atua como agente de mediação entre as requisições de usuário e os recursos dos sistemas de nuvem disponíveis no ambiente. Por possuir o papel de intermediário entre as principais partes interessadas no funcionamento e desempenho de ambientes de computação em nuvem, muitos trabalhos envolvendo melhorias para o desempenho para *cloud brokers* têm sido publicados (FINGAR, 2012) (AAZAM; HUH, 2014) (ACHAR; THILAGAM, 2014) (PARDESI; KHAMPARIA; BAGDE, 2014) (KHANNA; JAIN, 2015). Sua posição estratégica e central, tanto nas arquiteturas baseadas em SOA quanto nas plataformas de computação em nuvem, as quais estão aderindo ao paradigma SOA, fazem do *cloud broker* a entidade de negociação que ampara as partes interessadas no funcionamento do ecossistema (clientes e provedores) efetuando a distribuição da carga de trabalho aos recursos de computação em nuvem. Assim, o *broker* necessita possuir, em suas estratégias, políticas de escalonamento que possam negociar parâmetros de QoS requisitados

relacionados a vários aspectos de interesse do requisitante (cliente), tais como confiabilidade, disponibilidade, escalabilidade, segurança e custo (TEWARI, 2012) (SANDHU; SOOD, 2014). O *broker* deve ainda servir ao provedor de nuvem (GENEZ; BITTENCOURT; MADEIRA, 2012) (TYAGI; KUMAR, 2015) auxiliando no balanceamento da carga e alocação de recursos com objetivo de tirar o melhor proveito da infraestrutura computacional disponível.

Uma outra questão importante são os modelos de implementação de nuvem computacional, os quais influenciam o modo como o ambiente é organizado e têm impacto sobre a sistemática de intermediação de *cloud brokers*. Os modelos de implantação fundamentais são os de nuvem privada, pública, híbrida e comunitária (PUTHAL *et al.*, 2015). No caso da nuvem privada, tem-se que a infraestrutura de nuvem pode ser da própria organização ou alugada como recurso exclusivo em provedores de nuvem externo. Em alguns estudos de caso específicos, cenários que contemplam o uso de aplicações considerando nuvem privada e pública são avaliados (CHUNLIN; LAYUAN, 2015). Comumente, este tipo de organização, a qual utiliza dois modelos de implementação distintos concomitantemente no mesmo ambiente, remete ao conceito de nuvem híbrida e, nestes casos, o que se busca é uma alternativa para ampliar a escala de uma nuvem privada corporativa utilizando também recursos externos (isto é, num provedor de nuvem pública) sem perder de vista custo, eficiência e desempenho (CHUNLIN; LAYUAN, 2015). Há também trabalhos cujos cenários experimentais consideram o modelo de implementação de nuvem comunitária, o qual prevê o compartilhamento de recursos de computação em nuvem de diversos *datacenters* de distintos provedores ou instituições com o intuito de escalar maior potência computacional por meio da união dos recursos computacionais de diversos parceiros para um interesse comum.

1.3 Motivação

Conforme já explanado na contextualização deste trabalho, o *cloud broker* é um elemento presente nas arquiteturas orientadas a serviços e de computação em nuvem que executa um processo de intermediação e negociação das requisições de clientes encaminhando-as para o recurso ou serviço no qual a tarefa deverá ser processada.

A motivação deste trabalho é a avaliação de políticas ou algoritmos de escalonamento de tarefas para que um *cloud broker*, considerando um ambiente com arquitetura orientada a serviços, possa distribuir as requisições segundo modos de operação distintos. Os algoritmos de escalonamento considerados por um *broker* para intermediar requisições de clientes

dependem de um considerável número de fatores. Dentre os fatores pode-se citar o objetivo do escalonamento, por exemplo, minimizar o tempo de resposta para o cliente (ALAM *et al.*, 2012) (PATEL, G.; MEHTA; BHOI, 2015), maximizar o número de requisições atendidas pelos *hosts* (HUA; CHANG, 2012) ou máquinas virtuais (PATEL, K. S.; SARJE, 2012), economizar energia (KLIASOVICH; BOUVRY, 2012) (MASTELIC *et al.*, 2014), minimizar o descarte de requisições (GENEZ; BITTENCOURT; MADEIRA, 2012), o ambiente de computação em nuvem no qual o *broker* coexiste (nuvem privada, pública, comunitária ou híbrida), a quantidade estimada de clientes, tipo de aplicações executadas, modelo de carga de trabalho estimada, entre outros.

Os fatores supracitados constituem uma parte dos desafios existentes na concepção do projeto e na implementação de *cloud brokers*, dado que, por serem elementos intermediários, sofrem influência de todos os componentes do ambiente de computação, principalmente dos clientes e dos provedores de serviço, que possuem grande interesse no desempenho dos sistemas de nuvem, contudo, por meio de perspectivas completamente diferentes.

Com o crescimento do setor de computação em nuvem, por meio do surgimento de novos provedores, os quais investem continuamente na aquisição, manutenção, atualização e expansão de *datacenters*, a pluralidade da oferta de serviços de nuvem é cada vez maior e, com a tendência de tornar o ambiente da internet um grande ecossistema orientado a serviços no futuro, os serviços de *brokerage* estão em posição de destaque para auxiliar os clientes a encontrar os serviços pelos quais procuram. Desta forma, a complexidade do ambiente e o grau de transparência dos serviços torna-se cada vez maior. Nesse contexto, o componente intermediador estará sempre presente buscando uma relação custo/benefício interessante para atender aos requisitos de QoS dos clientes e dos provedores.

1.4 Hipótese

A hipótese desta tese é a de que é possível obter melhores resultados em desempenho a partir do emprego de um *cloud broker* como elemento de intermediação com tratativas de QoS num ambiente de computação em nuvem híbrida do que com emprego de um *cloud broker* com políticas tradicionais de mediação ou escalonamento de tarefas.

A partir da proposição desta hipótese, os processos de negociação e escalonamento de tarefas executados, poderão oferecer o desempenho esperado na execução das requisições e, ao mesmo tempo, balancear a carga dando preferência ao *datacenter* de nuvem privada, minimizando, desta forma, o tempo e o custo para a execução dos serviços. Considera-se, nesta

tese, a infraestrutura de nuvem privada como sendo uma infraestrutura de propriedade da organização que utiliza o ambiente de nuvem híbrida.

Entre os ganhos esperados com a proposição desta hipótese para a investigação conduzida nesta tese estão os benefícios oferecidos pelas tratativas de qualidade de serviço, tendo em vista que poderão beneficiar tanto a clientes quanto aos provedores de nuvem. Além disso, em relação ao aproveitamento da infraestrutura que é disponibilizada no contexto do ambiente de computação em nuvem, espera-se que a utilização de recursos seja suficientemente melhor, justificando que o emprego de um componente intermediador com um modo de operação eficiente provoque uma otimização do uso de recursos e o aumento do processamento de tarefas dentro do mesmo intervalo de tempo de experimentação.

1.5 Objetivo

O objetivo central do presente trabalho de doutorado, para comprovar a tese definida, é realizar uma análise de requisitos, o projeto e o desenvolvimento de um *Cloud Broker*, baseado em arquitetura orientada a serviços, para ambientes de computação em nuvem híbrida (considerando, para tal ambiente, uma nuvem privada corporativa e uma nuvem pública externa) focando o processo de negociação e escalonamento de tarefas de modo a atender a requisitos de QoS.

A partir do objetivo geral, um conjunto de objetivos específicos foram delineados:

- Caracterização de um ambiente computacional de nuvem híbrida, especificar uma infraestrutura de *datacenter* de nuvem privada em ambiente de rede local e uma infraestrutura de datacenter de nuvem pública em ambiente externo, ou seja, na Internet.
- Criação de uma arquitetura orientada a serviços, prover a abstração geral de serviço (por exemplo, implementações de Serviços Web), prover uma sistemática de diferenciação de serviços. Também deverá ser possível executar a instanciação de serviços em ambas as infraestruturas de nuvem privada e pública.
- Criação de um Mecanismo para Alocação/ Desalocação Dinâmica de serviços nas infraestruturas: as máquinas virtuais, instanciadas nas infraestruturas de nuvem consideradas, serão as plataformas para o mecanismo de gerenciamento

de serviços atuar e efetuar a manipulação dos serviços conforme a demanda. Esse mecanismo deverá ser utilizado pelo *broker* quando houver a necessidade.

- Desenvolvimento de uma sistemática de intermediação, para todos os modos de operação do *Cloud Broker*, para o atendimento a, pelo menos, um requisito de qualidade de serviço denominado de tempo máximo de execução da tarefa.
- Criação de uma sistemática para emular clientes virtuais: esses clientes devem ter a possibilidade de enviar cargas de trabalho sintéticas e cujo modo de submissão possa ser tanto de tarefas individuais quanto de grupos de tarefas.

1.6 Metodologia

Para atingir o conjunto de elementos descritos como objetivos específicos e comprovar a tese proposta, algumas etapas metodológicas devem ser realizadas.

Os resultados da execução de experimentos que considerem um algoritmo ou política de escalonamento de tarefas sem tratativas de QoS devem ser comparados aos algoritmos ou políticas de escalonamento de tarefas que considerem QoS considerando uma arquitetura SOA. As comparações devem ser efetuadas considerando o mesmo ambiente de execução para as tarefas, isto é, um ambiente de nuvem computacional híbrida, pois, tal ambiente apresenta diversos desafios ao procedimento de mediação das requisições dos clientes, dado que pode ser composto por vários modelos de nuvem e que o desempenho e o custo final das tarefas dependerão da melhoria no aproveitamento da infraestrutura de *datacenters* disponível no ambiente híbrido.

Para chegar aos resultados experimentais, o método empregado nesta tese baseia-se na investigação de trabalhos correlatos e recentes da área de computação em nuvem e na utilização de ferramentas de desenvolvimento de *software* do tipo IDE (*Integrated Development Environment*) e a utilização de ferramentas e APIs (*Application Programming Interface*) de simulação de computação em nuvem para a implementação dos cenários necessários e de uma arquitetura de computação em nuvem que consolide todos os elementos delineados nos objetivos deste trabalho.

As etapas metodológicas realizadas para a construção deste trabalho, em sua respectiva ordem, foram:

1. Estudo dos trabalhos que apresentam arquiteturas de *Cloud Brokers*, técnicas ou algoritmos de escalonamento de tarefas para sistemas de nuvem em

ambientes com ou sem a consideração de serviços e técnicas de gerenciamento e provimento de qualidade de serviço.

2. A partir do estudo realizado, foi realizada uma análise, por meio da qual foi possível identificar e selecionar os elementos básicos a serem acoplados à arquitetura de *Cloud Broker* idealizada para o presente trabalho.
3. Foram estudados e implementados diferentes modos de operação no *Cloud Broker* proposto, considerando os modos que são descritos no modelo de referência para sistemas de computação em nuvem do NIST, isto é, intermediação, agregação e arbitragem.
4. Caracterização de uma nova arquitetura denominada CloudSim-BEQoS (BEQoS trata-se de um acrônimo em inglês, *Bursty, Energy and Quality of Service*), a qual constitui-se de um ambiente computacional de nuvem híbrida, com base em arquitetura orientada a serviços e tratativas para qualidade dos serviços considerados. A nova arquitetura foi implementada como extensão da API do simulador CloudSim *Simulation Toolkit* (CALHEIROS *et al.*, 2011).
5. Realização de projeto e execução de experimentos para possibilitar a obtenção de resultados cujas informações foram usadas para comprovar a importância da nova arquitetura de *Cloud Broker* nos cenários de computação em nuvem híbrida e a eficiência do mesmo em manter os níveis de QoS considerando um ambiente com arquitetura seja orientada a serviços.

A arquitetura proposta neste trabalho chama-se QBroker (proveniente de QoS *Broker*), a qual representa um *cloud broker* com características de negociação que implementam a intermediação de serviços. A implementação da arquitetura QBroker difere-se das demais propostas de *cloud brokers*, pois possui a característica de abertura, isto é, foi conceitualmente concebida de forma que se possa acoplar ao *cloud broker* vários modos de operação (com base nos modos de operação descritos no modelo de referência de Computação em Nuvem do NIST (BUMPUS, 2013)) que podem ser associados a diferentes políticas de escalonamento de tarefas.

A metodologia de desenvolvimento do trabalho foi a simulação de computação em nuvem por meio do CloudSim *Simulation Toolkit* (CALHEIROS *et al.*, 2011). Através desse simulador, foi possível implementar os elementos necessários nos cenários de simulação para a execução do trabalho, isto é, a definição de *datacenters* de provedores de nuvem contendo as máquinas reais (*hosts*) e suas configurações, as máquinas virtuais (do inglês, *virtual machines*

ou VMs) e suas definições, o componente intermediador *cloud broker*, os *links* de interconexão de rede (entre clientes e *broker* e também entre *broker* e provedores de nuvem), a caracterização das aplicações por meio de arquitetura orientada a serviços, a geração de tarefas e das cargas de trabalho.

Com relação à execução de experimentos referentes a este trabalho, foram desenvolvidos três cenários com diferentes planejamentos experimentais, de modo a avaliar o impacto do componente QBroker nas variáveis de resposta consideradas. Além disso, a metodologia de abordagem para o delineamento de fatores, níveis e variáveis de resposta do ambiente computacional foi baseada numa sistemática de avaliação de desempenho para sistemas computacionais (JAIN, R., 1991) bem difundida na comunidade de pesquisa em Ciência da Computação. Todos os cenários experimentais foram realizados com planejamento de experimentos do tipo fatorial completo.

1.7 Estrutura do documento

A estrutura de documento desta tese está organizada em 7 capítulos.

No **capítulo 2** é apresentada a fundamentação teórica sobre Computação em Nuvem, a descrição de sua arquitetura e componentes baseando-se no modelo de referência do NIST. São apresentados os modelos básicos de serviço e de implantação, estruturas homogêneas e heterogêneas, abstração de recursos e sistemática de alocação, escalonamento, qualidade de serviço e simulação.

O **capítulo 3** discorre sobre Arquiteturas Orientadas a Serviço, conceitua os serviços Web e provê um detalhamento de sua implementação apresentando o modelo de serviço, os protocolos envolvidos na comunicação, composição de serviços Web e qualidade de serviço para serviços Web.

O **capítulo 4** introduz um conteúdo direcionado aos *Cloud Brokers*, por meio de um resumo de trabalhos recentes que se relacionam com o tema. Este capítulo também provê a conceituação essencial com componente *broker*, a comunicação entre o *cloud broker* e demais componentes de um ambiente de nuvem e os tipos de implementação comuns de *broker* nos trabalhos recentes.

No **capítulo 5** é apresentada a CloudSim-BEQoS, uma arquitetura de computação em nuvem que possui um conjunto de funcionalidades projetado para permitir a implementação, teste e validação de modelos computacionais para ambientes de computação em nuvem híbrida.

Uma explanação geral da arquitetura, de seus componentes e até alguns exemplos de programação baseados em linguagem Java são apresentados. Também é provida uma discussão da arquitetura QBroker proposta por esta tese de doutorado. Um detalhamento do funcionamento e do modo de operação deste *Cloud Broker* é provido, juntamente com alguns exemplos da aplicação do mesmo em programas de simulação usando linguagem Java.

O **capítulo 6** contém a avaliação de desempenho realizada para validar o funcionamento da arquitetura QBroker em diferentes cenários, comparando seu desempenho com o algoritmo de escalonamento de tarefa Round Robin. Considerações a respeito da análise realizada após a obtenção dos resultados experimentais também são apresentadas.

No **capítulo 7** são apresentadas as conclusões finais desta tese em conjunto com as contribuições do presente trabalho. São também discutidos os trabalhos futuros para a continuidade do projeto de pesquisa.

---

2. Computação em Nuvem

2.1 Considerações iniciais

Neste capítulo serão apresentados tópicos referentes a Computação em Nuvem, detalhes da arquitetura desse tipo de sistema computacional, modelos de implantação e de serviço, qualidade de serviços em nuvem, tipos de organização de estrutura, escalonamento, alocação de recursos e simulação.

Uma Nuvem Computacional pode ser definida como um sistema distribuído, constituído por um conjunto de computadores interconectados. Esses recursos são provisionados dinamicamente, apresentando-se de forma virtualizada ou não, regidos por contratos de nível de serviço (do inglês, *Service Level Agreement* ou SLA) estabelecidos entre o provedor de serviços e clientes (BUYYA; TOOSI; CALHEIROS, 2014).

Os sistemas de computação em nuvem continuam emergindo no mercado de computação devido a algumas de suas características, as quais são consideradas interessantes para as aplicações modernas. Entre as características importantes, pode-se citar:

- Alocação dinâmica de recursos computacionais, tais como unidades de processamento ou CPU (do inglês, *Central Processing Unit*, neste caso fazendo referência direta a processadores), memória dinâmica, armazenamento em disco, máquinas virtuais e muitos outros tipos de abstrações para recursos. O gerenciamento dessa sistemática fica encapsulado dentro do sistema de nuvem, logo, tal atividade é transparente ao usuário final.
- Gerenciamento da alocação de recursos através da propriedade conhecida como elasticidade computacional, isto é, os recursos são escalados para a aplicação de acordo com a necessidade conforme houver demanda de clientes ou de carga de trabalho.

- Para a cobrança pelos serviços prestados, os sistemas de nuvem adotam o modelo de custo “*pay-as-you-go*”. Esse modelo tornou-se uma tendência, pois o cliente não paga pela locação mensal de um recurso e sim pelo tempo que o estiver usando. Desta forma, o cliente estará pagando apenas pelos recursos.

Esse padrão para desenvolvimento e implantação de aplicações na Web com apoio de sistemas de computação em nuvem é ideal para alguns tipos de aplicações, como por exemplo as redes sociais, os *websites* ou portais de conteúdo, hospedagem Web, provimento de serviços Web utilitários (por exemplo, consulta de CEP, consulta e agendamento de voos, hotéis e aluguel de carros) e páginas de eventos (por exemplo, copa do mundo e olimpíadas), que podem ter demanda variável de acessos (JADEJA; MODI, 2012). Percebendo as possibilidades lucrativas desse novo paradigma, algumas empresas já se firmaram nesse mercado como provedores de tecnologia e serviços de nuvem, por exemplo, a Microsoft possui a plataforma *Azure* (MICROSOFT, 2016), a *Amazon* disponibiliza a plataforma EC2 (do inglês, *Elastic Compute Cloud*) (AMAZON, 2016), a *Google* fornece vários serviços de nuvem por meio da *Google Cloud Platform* (GOOGLE, 2016b) e a *Oracle* provê serviços de nuvem com a plataforma *Oracle Cloud* (ORACLE, 2016).

2.2 Arquitetura de Nuvem Computacional do NIST

O NIST (*National Institute of Standards and Technology*), instituição norte-americana de padrões tecnológicos publicou uma arquitetura de referência de computação em nuvem a pedido do governo norte-americano (BOHN *et al.*, 2011). O objetivo principal de tal documento foi descrever as interações existentes entre os elementos (denominados atores no modelo proposto) que se encontram no âmbito de sistemas de computação em nuvem, por meio de um modelo de referência independente de fabricante.

O modelo de referência de computação em nuvem do NIST resultou na identificação dos seguintes atores principais: Cliente, Provedor, Intermediador, Auditor e Transportador. É possível, conforme apresentado na Figura 2.1, observar quais são os componentes da arquitetura de referência em computação em nuvem do NIST e seus respectivos componentes ou funcionalidades internas.

Alguns dos atores presentes na **Erro! Fonte de referência não encontrada.** possuem diversas funcionalidades internas (como é o caso do Auditor de Nuvem, Provedor de Serviços de Nuvem e Intermediador de Nuvem), de modo que estas devem ser melhor explanadas para a compreensão de qual o papel de cada ator no funcionamento de um ambiente de nuvem

Figura 2.1: Arquitetura de Referência de Computação em Nuvem do NIST. (BOHN *et al.*, 2011)

computacional.

O detalhamento dos atores considerados no modelo de computação em nuvem do NIST, bem como seus elementos internos serão discutidos na sequência.

2.2.1 Cliente de Serviços de Nuvem

O primeiro ator da esfera de interações de nuvem computacional a ser considerado é o **cliente** ou **consumidor de serviços de nuvem**. Sua atividade principal é a de pesquisar e localizar os serviços de nuvem de seu interesse, enviar requisições e obter as respostas das tarefas enviadas. Além disso, faz parte das rotinas do cliente a efetuação de pagamento pelos serviços prestados (ou não, para o caso de serviços de nuvem gratuitos) (BOHN *et al.*, 2011).

O cliente pode interagir diretamente com sistemas de nuvem computacional a partir dos modelos básicos de serviços disponibilizados: infraestrutura, plataforma, software como serviço ou a partir de derivações criadas a partir destes modelos. Todavia, em muitas situações

o cliente poderá consultar um intermediador de nuvem (ou *Cloud Broker*) de modo a pesquisa e descobrir o provedor de serviços de nuvem que possui uma oferta adequada às restrições de qualidade de serviço desejadas (BOHN *et al.*, 2011).

2.2.2 Provedor de Serviços de Nuvem

Elemento fundamental para que o ambiente de computação em nuvem exista, o ator denominado **provedor de serviços de nuvem** é também um componente da arquitetura. O provedor executa como atividades principais: o provimento dos serviços de nuvem, gerenciamento de alocação de recursos, controle das camadas de infraestrutura e recursos físicos de nuvem (BOHN *et al.*, 2011). É oportuno destacar que o provedor de serviços é o ator mais complexo existente no ambiente de nuvem, pois, toda a infraestrutura física e lógica da gestão dos recursos reais e virtuais estão concentrados no mesmo. Além disso, a sistemática de modelos de serviços prestados, modelos de custo e o provisionamento e alocação de recursos de forma dinâmica, são tarefas complexas que devem ser administradas por meio das políticas internas de cada provedor.

2.2.3 Intermediador de Nuvem

O **Intermediador de Nuvem** ou *Cloud Broker* se posiciona, como ator, entre os clientes e os provedores de serviço de nuvem, assim, sua principal atividade é a de mediador das atividades entre esses atores. Uma de suas funções é facilitar o gerenciamento da integração entre os clientes e os serviços de nuvem. Além disso, o *broker* é um elemento especial da arquitetura de nuvem computacional do NIST por desempenhar um papel duplo, isto é, ele se comporta como um provedor quando interage com o cliente e, por outro lado, se comporta como cliente quando interage com o provedor (BOHN *et al.*, 2011).

No detalhamento de suas atividades, pode-se citar três funcionalidades ou modos de operação principais:

- Intermediação;
- Agregação;
- Arbitragem.

Quando há a necessidade do *broker* criar uma nova instância de um serviço já existente para a melhora da qualidade de serviço ou é necessário prover uma instância de um novo serviço não existente, essa atividade tem o nome de modo de operação de **intermediação**. O modo de operação não está associado apenas com a manipulação da coletânea de serviços de nuvem

disponibilizados, mas também com o atendimento dos clientes conforme restrições (ou requisitos de QoS) que forem solicitadas (BOHN *et al.*, 2011).

A **agregação** é uma atividade que, normalmente, está relacionada ao atendimento de grupos de tarefas e trata-se da combinação ou agrupamento de vários serviços de nuvem ou quando há a necessidade de melhorar o desempenho de um serviço já existente. Na agregação é importante citar que normalmente se usa uma espécie de mapa de equivalência dos serviços, por exemplo, se for preciso substituir um serviço Web de locação de veículo, encontra-se nesse mapa quem são os possíveis serviços substitutos compatíveis, sendo que os substitutos não precisam pertencer necessariamente ao mesmo provedor de serviços de nuvem. Da mesma forma que na intermediação, no modo de operação de agregação é usual o atendimento à requisitos de QoS solicitados pelos clientes (BOHN *et al.*, 2011).

O modo de operação de **arbitragem** realiza uma atividade parecida com a agregação, isto é, é um modo de operação mais ideal para ambiente nos quais considera-se grupos de requisições. Todavia, na arbitragem, a atividade de combinação de serviços, quando executada, não utiliza, necessariamente, um mapa de serviços substitutos fixos. Isto é, pode explorar a disponibilidade dos serviços e provedores a cada negociação executada de modo dinâmico por meio de informações de estado dos recursos atualizadas continuamente (BOHN *et al.*, 2011).

É importante citar que as atividades desempenhadas pelo *broker* não se limitam a essas que são apresentadas pela arquitetura de referência do NIST. Outros trabalhos (TEWARI, 2012) (ACHAR; THILAGAM, 2014) (AAZAM; HUH, 2014) (PARK; AN; YEOM, 2015) (KHANNA; JAIN, 2015) apresentam arquiteturas de *cloud brokers* com conotações diferentes ou mais específicas em relação aos modos de operação do modelo de referência de NIST (BOHN *et al.*, 2011). Entre essas abordagens diferenciadas, encontram-se *cloud brokers* que consideram vários elementos que são gerenciados pelo *broker*, tais como restrições de SLA (WU, L.; GARG; BUYYA, 2011) (GENEZ; BITTENCOURT; MADEIRA, 2012) (FANIYI; BAHSOON, 2015), requisitos de QoS (TEWARI, 2012) (AAZAM, 2014), o contexto das requisições de clientes (KIANI *et al.*, 2011), informações de estados obtidas a partir de agentes de monitoramento de recursos (VARALAKSHMI; MAHESHWARI, 2013) (ACHAR; THILAGAM, 2014), entre outros. Essas estratégias incrementais ao modelo de referência NIST são empregadas, geralmente, com o intuito de melhorar o desempenho dos serviços de nuvem (FINGAR, 2012).

2.2.4 Auditor de Nuvem

Todo sistema computacional complexo, tal como uma nuvem computacional, deve ser avaliado e auditado com frequência para manter certos indicadores em níveis aceitáveis. Alguns indicadores considerados são o desempenho, gerenciamento de segurança, gestão da privacidade dos clientes, entre outros. Dessa forma, acompanhando e controlando tudo o que é crítico para o bom funcionamento do sistema, pode-se chegar aos resultados esperados, cumprindo, assim, contratos de nível de serviço e os requisitos de segurança. Tendo as premissas anteriores em mente, está presente no ambiente de computação em nuvem o ator denominado **Auditor de Nuvem** (BOHN *et al.*, 2011), o qual realiza avaliações e fiscalizações de características de sistemas de nuvem com o objetivo de efetuar constantes análises sobre os controles referentes, principalmente, ao desempenho e à segurança.

2.2.5 Transportador de Nuvem

O **Transportador de Nuvem** é um intermediário entre os sistemas computacionais e de nuvem e seus ambientes externos no que tange ao ambiente de rede. Esse ator oferece conectividade e transporte de dados de serviços dos provedores de nuvem até os clientes (BOHN *et al.*, 2011). Essa conectividade é provida através das tecnologias de telecomunicação existentes, tais como *links* de comunicação de dados criptografados e dedicados. A atividade principal desempenhada pelos transportadores de nuvem é a da comunicação de dados e, normalmente, existem SLAs entre transportadores e os provedores de nuvem, de modo que se tornam parceiros fortemente acoplados para cumprir os SLAs de clientes. Além de atuarem com a comunicação e transporte dos dados entre os elementos componentes de um ambiente de nuvem, são atividades que também podem ser desempenhadas pelos transportadores: o provimento de mecanismos de segurança para o transporte dos dados, como por exemplo a criptografia dos pacotes de dados transeuntes pelo ambiente de rede.

2.2.6 Considerações adicionais sobre a Arquitetura de Nuvem do NIST

Os elementos conceituais apresentados não explanaram em detalhes os elementos internos de cada ator. Assim, se faz importante esclarecer alguns elementos pontuais.

Dentre os elementos do componente provedor de serviços de nuvem (conforme apresentado na Figura 2.1, pode-se verificar os seguintes modelos de serviço: infraestrutura como serviço (do inglês, *Infrastructure as a Service* ou IaaS), plataforma como serviço (do inglês, *Platform as a Service* ou PaaS) e software como serviço (do inglês, *Software as a Service*

ou SaaS). A implementação do modelo de serviço pode ser entendida como a abstração de recurso que o usuário tem percepção em relação ao sistema de nuvem computacional. Por exemplo, se o modelo de serviço implementado for IaaS, o usuário estará acessando a componentes de infraestrutura (*virtuais ou reais*) tais como servidores, interfaces de rede, discos, memória, CPUs, entre outras possibilidades. Caso o modelo de serviço implementado venha a ser PaaS, o usuário irá ter a percepção do serviço prestado como uma plataforma de implantação de suas aplicações, usando APIs e recursos de programação providos e padronizados pelo provedor de nuvem. Se o modelo de serviço considerado for SaaS, o usuário estará utilizando uma interface de usuário para uma aplicação que está implantada e disponibilizada num sistema de nuvem. É possível perceber que muitas vezes as estruturas internas do sistema são totalmente transparentes ao usuário, logo, o usuário acaba apenas se relacionando com uma interface controlada de acesso aos recursos de computação de acordo com o modelo de serviço disponibilizado.

Para prover esses mecanismos que promovem diferentes interações com os clientes/usuários do sistema de nuvem, o provedor conta com um elemento interno chamado **Camada de Controle e Abstração de Recursos** (do inglês, *Resource Abstraction and Control Layer*), conforme consta da Figura 2.1. Essa camada é responsável pelo provimento da “visão” concedida ao cliente no momento de utilizar os serviços e recursos computacionais de nuvem. Na Figura 2.1, abaixo da camada referida anteriormente, encontra-se a **Camada de Recursos Físicos** (do inglês, *Physical Resource Layer*), a qual é responsável pela gerência dos recursos de *hardware* do *datacenter* da nuvem, alocação e desalocação desses recursos para as aplicações e, finalmente, pelas instalações físicas (dentro do *datacenter*) existentes para execução dessas atividades.

Há também um importante elemento interno do provedor de serviços a ser citado (conforme apresentado na Figura 2.1), chamado de **Gerenciador de Serviços de Nuvem**, o qual é responsável pelo gerenciamento da coletânea de serviços de nuvem oferecidos. Um primeiro papel deste elemento é coordenar o suporte aos negócios estabelecidos com clientes nos termos dos modelos de serviço oferecidos (IaaS, PaaS ou SaaS). Esse elemento é responsável ainda pela gestão de provisionamento e configuração dos recursos, de modo que suas definições são passadas à camada de controle e abstração de recursos a qual é responsável por prover a transparência necessária para os usuários finais dos serviços de nuvem. Por meio dos elementos mencionados, esse gerenciador deve prover portabilidade e interoperabilidade

através de padrões que facilitem a interação do sistema de nuvem com as aplicações do cliente. Considerando ainda a Figura 2.1, ao lado do módulo de gerenciamento de serviços de nuvem, existem os subcomponentes referentes às camadas de segurança e privacidade que zelam pelos aspectos da confiabilidade na execução dos serviços e por assegurar o acesso seguro aos recursos, serviços, dados e informações pertencentes aos clientes.

2.3 Modelos de Implantação

Os ambientes de computação em nuvem tornam-se, cada vez mais, organizações complexas com aglomerados de infraestruturas física (prédios, sistemas de arrefecimento, cabeamentos), sistemas de computação (*hardware* referente aos computadores), equipamento de interconexão de redes (*switches*, roteadores, *links* de comunicação com a internet, entre outros) e também muito software (sistemas operacionais, *middleware*, plataformas de implantação de aplicações, entre outras possibilidades) (KHAN; REHMAN; ANWAR, 2011) (DE ALFONSO *et al.*, 2011) (VEEN *et al.*, 2013) (NUTHULA; CHALLA, 2015). Tal complexidade que, no início da era da computação em nuvem, era gerenciada dentro dos *datacenters*, hoje está ainda mais agravada, partindo-se do princípio que os provedores de nuvem possuem múltiplas unidades de *datacenter* espalhadas pelo mundo e que necessitam, de forma crescente, estabelecer comunicação, interoperabilidade, compartilhamento (de dados, informações e recursos computacionais) e colaboração entre essas infraestruturas distribuídas.

Assim, conforme a abrangência que um ambiente de computação em nuvem passa a atingir, seu tamanho em termos de capacidade computacional também aumenta, surgem então os chamados modelos de implantação. Segundo o modelo de referência em computação em nuvem do NIST (BOHN *et al.*, 2011) (LIU, F. *et al.*, 2012), os modelos básicos de implantação para ambientes de computação em nuvem são quatro, sendo eles:

- Nuvem Privada
- Nuvem Pública
- Nuvem Híbrida
- Nuvem Comunitária

Cada um dos modelos supracitados considera uma ou mais instituições proprietárias dos sistemas de nuvem, a relação entre essas instituições e a sua relação com ambiente no qual estão os clientes. Além disso, admitem diferentes níveis de acessibilidade, privacidade e segurança de acordo com o escopo da infraestrutura e objetivo para o qual a implantação foi realizada.

Em um trabalho que elaborou uma revisão de computação em nuvem, os autores apresentam uma taxonomia dos modelos de implantação de nuvem, na qual discorrem sobre os tipos de modelos de implantação, além disso, o trabalho também foca de maneira mais específica no que tange a questão das nuvens comunitárias (YOKOYAMA; YOSHIOKA, 2012). Em um outro trabalho que dedicou-se ao estudo da complexidade de ecossistemas de computação em nuvem, os autores também proveem a conceituação de cada um dos modelos de implantação de nuvem (KALLONIATIS; MOURATIDIS; ISLAM, 2013). Com base nesses trabalhos e também em outros da literatura, serão apresentadas a seguir as definições de cada um dos modelos de implantação de sistemas de nuvem.

2.3.1 Nuvem Privada

Nuvens privadas são as que possuem infraestrutura (computadores, *links* de Internet, sistemas de interconexão de rede, *storage* entre outros) dedicada a uma dada instituição ou empresa. Isto é, os processos e os dados que são executados e armazenados nesse tipo de sistema de nuvem pertencem somente à organização que é a sua proprietária. Nesse tipo de nuvem computacional, dado que possui um perfil de uso exclusivo para a organização que a possui, não há restrições contratuais quanto ao uso dos recursos de nuvem, tais como largura de banda, espaço em disco, alocação de CPUs e memória, entre outros (LELOGLU; AYAV; ASLAN, 2013). É importante salientar que, embora possua a denominação de nuvem privada, a infraestrutura desse tipo de sistema de nuvem pode se situar tanto dentro do espaço físico da própria organização (quando a mesma possui *datacenter* próprio) como também pode ser uma infraestrutura locada de um provedor de nuvem externo, porém, com restrições contratuais que garantem à organização de que o provedor irá alocar aquela quantia de recursos computacionais para uso exclusivo (SOFTLAYER, 2016).

2.3.2 Nuvem Pública

Sistemas de nuvem pública são pertencentes a provedores de nuvem (empresas especializadas no seguimento e donas da infraestrutura) que provisionam seus recursos com base em algum modelo de serviço via Internet. Dentre os muitos provedores de nuvem pública existentes, o acesso aos recursos é oferecido aos clientes segundo vários modelos de negócio e precificação. Dessa forma, o cliente irá pagar pelo uso dos recursos por meio de um sistema de locação de serviços de infraestrutura, plataforma ou *software* (LELOGLU; AYAV; ASLAN, 2013).

2.3.3 Nuvem Híbrida

O modelo de implantação de sistemas de nuvem híbrida é concebido a partir da combinação de dois ou mais modelos de implantação. Por exemplo, pode-se conceber um ambiente de nuvem híbrida a partir da conjunção colaborativa entre uma nuvem privada e uma nuvem pública. É comum que as nuvens híbridas surjam a partir da necessidade de maior poder computacional para uma empresa, a qual já possui uma estrutura de nuvem computacional e está disposta a expandir o seu sistema delegando parte da carga de trabalho, por exemplo, ao provedor de nuvem pública (LELOGLU; AYAV; ASLAN, 2013). Em suma, o ambiente de nuvem híbrida consiste de múltiplos provedores internos/externos de computação em nuvem trabalhando de modo colaborativo para suprir as necessidades de uma ou mais organizações buscando atender à demanda de carga de trabalho existente.

2.3.4 Nuvem Comunitária

No modelo de implantação de nuvem comunitária apresenta-se também um aglomerado de vários sistemas de nuvem de organizações diferentes, o que torna sua definição um pouco parecida com a de nuvem híbrida. Porém, numa nuvem comunitária, diversas instituições (normalmente órgãos governamentais, institutos de pesquisa ou universidades) juntam o poder computacional de seus sistemas de nuvem privados com o objetivo de se alcançar uma escala de recursos computacionais maior, sem a necessidade de dispor de investimentos para tal. Além disso, essa junção de recursos computacionais dos *datacenters* de cada instituição não precisa ser total, ou seja, os participantes dessa comunidade de nuvem computacional podem ceder apenas uma parte de seus recursos. Desta forma, as organizações compartilham recursos de seus sistemas de computação com as demais, formando um grande sistema de computação em nuvem com escopo multi-institucional (LELOGLU; AYAV; ASLAN, 2013) (BHATT; PROF; PATEL, 2012) (AL-ROOMI *et al.*, 2013). Essa abordagem é bastante atrativa para a execução de aplicações científicas que necessitam de grande poder computacional para domínios de problema como processamento de imagens, mineração de dados, cálculos científicos complexos, processamento de dados de informações geográficas ou climáticas, entre outros.

2.3.5 Considerações adicionais sobre modelos de implantação

Os quatro modelos básicos de implantação apresentados são composições envolvendo um ou mais sistemas de computação em nuvem que, normalmente, são construídos a partir de

hardware de diferentes fabricantes, pertencem a organizações proprietárias distintas e sua estrutura de *software* é implantada cada qual à sua maneira.

Na Figura 2.2, são apresentados os modelos de implantação já discutidos. Todos os modelos podem estar conectados por meio de um ambiente de rede de qualquer topologia, conforme a necessidade.

É importante verificar que a criação de um ambiente computacional de nuvem híbrida denota a criação de uma interação entre o sistema de computação em nuvem privado de uma dada organização cliente e um provedor de serviços de nuvem externo. Todavia, as políticas que definem como e quando os recursos da nuvem externa serão utilizados, são estratégias proprietárias da organização cliente.

A ideia da nuvem computacional comunitária se insere no contexto da Figura 2.2, sendo que para esse cenário se consumir é importante perceber a necessidade do agrupamento de diversos sistemas de nuvem formando um “novo” e maior sistema de nuvem, por meio da concessão colaborativa de recursos computacionais.

Figura 2.2: Modelos de Implementação de Computação em Nuvem. Adaptado de (DUMMIES.COM, 2016)

Os modelos de implantação, para serem efetivos devem ser planejados em associação com o modelo de serviço de nuvem que o provedor ou organização proprietária da infraestrutura

de *datacenter* julgar adequado. Na próxima seção são apresentados os modelos básicos de serviço em nuvem.

2.4 Modelos de Serviço

Os recursos computacionais que são apresentados aos clientes de sistemas de nuvem precisam possuir um formato. A forma (pode-se dizer também, a abstração) como se apresentam os recursos aos clientes define uma série de propriedades importantes para que os provedores possam conceder o acesso aos recursos para os clientes, bem como também serve para a definição dos controles necessários a gestão dos recursos.

Para definir a forma como os serviços são criados e disponibilizados aos clientes, foram definidos os chamados modelos de serviço. É consenso entre a indústria de computação em nuvem e os trabalhos recentes que definem esse conceito (BOHN *et al.*, 2011) (KAVIS, 2014) (BUMPUS, 2013) que os modelos básicos de serviço de nuvem configuram-se como três:

- Infraestrutura como Serviço
- Plataforma como Serviço
- *Software* como Serviço

A Figura 2.3 possui a representação dos três modelos básicos de serviço de nuvem computacional organizados em camadas.

Quanto mais inferior for a camada, maior será o nível de detalhamento, de elementos e de controles que o usuário obterá acesso. Os três modelos básicos da especificação não são exaustivos, isto é, existem muitas derivações dos modelos implementadas atualmente como um tipo de especialização a partir dos modelos básicos.

É importante destacar que logo abaixo da camada referente ao modelo de serviço de Infraestrutura como Serviço apresentado pela Figura 2.3 está o *hardware* provido como serviço, pois, é também um comportamento e característica padrão dos sistemas de nuvem que os equipamentos de sistema computacional sejam alocados sob demanda para os clientes (KAVIS, 2014).

Figura 2.3: Modelos de Serviço de Computação em Nuvem. Adaptado de (BUMPUS, 2013)

A definição de cada um dos modelos de serviço, suas siglas, o modelo de abstração de recurso adotado e alguns exemplos de provedores de serviço de cada modelo são apresentados nas subseções a seguir.

2.4.1 Software como Serviço

Frequentemente citado como modelo de Provedor de Serviço de Aplicação, o modelo de serviço denominado como *Software* como Serviço é empregado como nova forma de distribuição de aplicações por meio da Internet possuindo, normalmente, em sua retaguarda, todos os atributos providos pelas camadas de serviço inferiores de nuvem. Trata-se de um modelo de serviço no qual a visão (ou abstração de recurso) para o usuário será a interface de aplicação, provendo transparência em relação aos elementos estruturais internos da nuvem. Alguns exemplos de aplicações que são modeladas como SaaS são Google Docs, Microsoft Office 365, NetSuite, GMail, entre outros (BUMPUS, 2013).

2.4.2 Plataforma como Serviço

Oferecer os serviços de nuvem como plataforma significa prover aos desenvolvedores uma base sobre a qual podem desenvolver suas aplicações. Prover uma plataforma significa colocar à disposição do desenvolvedor os recursos necessários para o ciclo de vida de *software*, isto é, prover os recursos para desenvolver, testar, implantar e hospedar aplicações Web dos mais variados tipos (BUMPUS, 2013). Um dos provedores populares PaaS é o Google *Application Engine*, cuja plataforma oferece recursos para o desenvolvimento de aplicações programadas nas linguagens Java e *Python*, além de componentes adicionais de sua API customizada. É oferecido também um painel gestor das aplicações (*dashboard*), sistema de

implantação (*deployment*) de aplicação na nuvem e API de persistência de dados com o banco de dados *Big Table* (GOOGLE, 2016a).

2.4.3 Infraestrutura como Serviço

O provimento de *hardware* ou infraestrutura como serviço, como é conhecido, tornou o IaaS um dos mais populares e utilizados dentre os modelos de serviço de nuvem. Em geral, trata-se do fornecimento de *hardware* virtualizado (ou não virtualizado), de modo que os provedores geralmente oferecem planos e configurações prontas de VMs ou, em algumas ocasiões, permitem ao cliente customizar a configuração desejada para a sua VM (BUMPUS, 2013). O modelo de custo que frequentemente é utilizado para o modelo de serviço IaaS é também o “*pay as you go*”, porém, não é o único modelo de custo existente, pois alguns provedores fornece a opção de pagamento pelo uso mensal com descontos, entre outras variações.

Algumas vantagens do modelo de serviço IaaS são a alta disponibilidade (o recurso fica disponível 24 horas por dia na Internet), facilidade de alteração de configuração quando necessário, ausência de investimento em equipamentos de infraestrutura por parte do cliente e o pagamento feito conforme o uso dos recursos computacionais alocados. Exemplos de provedores desse modelo de serviço são Amazon EC2, Joyent, Mosso, RackSpace, entre outros.

2.5 Estrutura de Nuvem

Devido à característica de transparência, a qual está embutida nos modelos de serviço, os sistemas de computação em nuvem, na maioria das vezes, não concedem acesso direto ao *hardware* real na camada abaixo dos sistemas de virtualização dos recursos computacionais. Nos modelos de serviço de mais baixo nível, como por exemplo, o IaaS, o acesso aos equipamentos é provido por meio de máquinas virtuais.

Usando a técnica de virtualização, alguns provedores possuem uma estratégia de modelos prontos (ou *commodity hardware*) de máquinas virtuais para comercializar, formato no qual não se possui detalhes do hardware subjacente à VM. Geralmente o controle sobre a disponibilidade e localidade fica sob a gerência do provedor. Essa estratégia de estruturação dos recursos de nuvem a qual emprega transparência, padronização e generalização do acesso ao hardware dos *datacenters* é chamado de Computação em Nuvem Homogênea (CRAGO *et al.*, 2011).

2.5.1 Nuvem Computacional Homogênea

Nas nuvens cuja estrutura de recursos computacionais é homogênea, o *hardware* disponível em seus *datacenters* é disponibilizado aos usuários sempre de modo igual. Isso significa que, apesar de poder configurar suas máquinas virtuais com características diferentes, o usuário terá sempre perfis de máquinas físicas idênticos. Esse padrão estrutural, o qual torna-se quase uma política de gestão dos recursos de infraestrutura, é seguido por vários provedores de sistemas de nuvem atuais, podendo-se citar como exemplos a Amazon, Microsoft e Rackspace. Alguns provedores de nuvem permitem uma pequena configuração quanto ao controle da localidade onde os recursos serão processados (por exemplo Microsoft e Amazon), porém, continua-se na política de transparência e *commodity hardware* (CRAGO *et al.*, 2011).

2.5.2 Nuvem Computacional Heterogênea

Nuvens Heterogêneas são caracterizadas por permitir ao usuário acesso a um conjunto de recursos computacionais com diferentes caracterizações.

Para se definir melhor o campo de atuação de nuvens heterogêneas é necessário observar que há numerosos problemas que são adequados a composições de recursos heterogêneos. Um exemplo que necessita de recursos heterogêneos é o de aplicações do tipo HPC (*High Performance Computing*), pois, nesse tipo de aplicações, pode ser necessária a solicitação de diversos tipos de recursos computacionais de diferentes arquiteturas e configurações, para trabalhar em conjunto e processar a tarefa de modo eficiente. Dentre os recursos com diferentes arquiteturas estão os processadores de sinais digitais, processadores de pacotes de rede, unidades de processamento gráfica (GPUs), multiprocessadores simétricos (SMPs), processadores convencionais e muitas outras possibilidades (CRAGO *et al.*, 2011).

Assim, nuvens computacionais adequadas para as aplicações específicas como cálculos científicos, HPC, computação gráfica, processamento de imagens, processamento de dados para ciências biológicas e outras, podem se beneficiar de uma estrutura heterogênea em nuvem.

2.6 Recursos de Nuvem Computacional

Os sistemas de nuvem podem prover diversos tipos de recursos computacionais aos usuários. Contudo, a quantidade e a forma como os recursos se apresentam depende muito do modelo de serviço adotado pelo provedor. Assim, a compreensão de como os recursos são

disponibilizados auxilia no entendimento de como o sistema se comporta ao lidar com as requisições, alocar os recursos e devolver as respostas.

Em trabalhos sobre de arquiteturas abertas de nuvem computacional são citados quatro tipos de recursos para nuvem: recursos de infraestrutura, recursos de *software*, recursos de aplicação e recursos de processos de negócio (BELOGLAZOV; ABAWAJY; BUYYA, 2012) (NANDHINI, 2012) (CHUNG *et al.*, 2013) (PARDESI; KHAMPARIA; BAGDE, 2014).

Os recursos de nuvem implementados como **infraestrutura** são acessados e visualizados pelos usuários como elementos de *hardware* virtualizados, tais como provisionamento de máquinas (virtuais), espaço em disco (*storage*) e poder computacional (expresso em quantidade de CPUs virtuais alocadas, por exemplo). Exemplos desse tipo de implementação de recursos podem ser encontrados em sistemas como o Amazon EC2, Xdrive Box e Microsoft OneDrive.

Quando recursos de computação em nuvem tem seu foco em **software**, isso direciona o usuário a integrar seus programas com uma camada de *middleware* e vários outros recursos de desenvolvimento. Assim, de acordo com as especificações, o *middleware* é implementado como sistemas operacionais, servidores de aplicações, servidores de banco de dados, entre muitos outros modos de disponibilização possíveis. Assim, para disponibilização de *software* como recurso, podem-se desenvolver artefatos como plataformas, ferramentas de desenvolvimento, teste, implantação, e projetos do tipo *template* (WU, L.; KUMAR GARG; BUYYA, 2012).

Os recursos de nuvem implementados como **aplicação** associam-se frequentemente com o modelo de serviços SaaS e suas derivações. Assim, fabricantes do mercado de software, tais como Microsoft, Oracle, SAP e outros, estão movendo seus programas para nuvem computacional, disponibilizando a sua aplicação como serviço. Exemplos desse tipo de abordagem são o Microsoft Office 365, Google Docs, Facebook entre outros, que disponibilizam seus aplicativos via Internet apoiados por um poderoso sistema de computação em nuvem em seu *background*. Seguindo a ideia de aplicação como recurso, algumas empresas líderes de mercado estão disponibilizando suas aplicações corporativas ou pessoais como serviços, logo, nestes casos, para o usuário, a aplicação passa a ser o recurso disponibilizado, acessado e manipulado em seu perfil de acesso (WU, L.; KUMAR GARG; BUYYA, 2012).

O último tipo de recurso disponibilizado em nuvens computacionais são os **processos de negócio**. Esse tipo de modelo de disponibilização de recurso é mais complexo que os

anteriores, pois, precisa vincular um processo de negócio com empresas diferentes e prover o serviço de acordo com os requisitos desejados por cada uma. É possível modelar os processos de negócio como aplicativos utilitários de baixo acoplamento que atuam como subprocesso dentro de um processo de cliente. De acordo com a referida estratégia, os provedores de *Business Processes* (BP) podem compartilhar BPs, e os seus clientes, em contrapartida, terceirizam mais uma parte de sua aplicação com a vantagem do suporte a reutilização, composição e provisionamento (CHANG; WALTERS; WILLS, 2013).

2.7 Escalonamento em Nuvem Computacional

Uma das áreas em que mais são encontrados trabalhos em andamento e uma diversidade grande de metodologias e propostas novas é a de escalonamento para sistemas de nuvem computacional (VOS, 2014) (SINGH; PAUL; KUMAR, 2014). O escalonamento de requisições para o ambiente de nuvem computacional é tarefa primordial para oferecer alto desempenho, alcançar as metas contratuais de SLAs ou mesmo para atender restrições individuais de clientes esporádicos dos serviços.

Nesta seção, são discutidos alguns trabalhos sobre escalonamento para ambientes de nuvem computacional. Alguns trabalhos são diretamente relacionados ao tema da tese, isto é, lidam com cenários de computação em nuvem e outros lidam com cenários que consideram *clusters* e *grids* de computadores.

Autores de um trabalho sobre computação em grade afirmam que há diferenças em se implementar escalonamento em sistemas de *cluster* e sistemas de grade. Nos *clusters* o foco do escalonamento é prover melhoria do desempenho geral do sistema, já para sistemas de grade o foco é a melhoria do desempenho das aplicações individualmente de modo a cumprir requisitos de QoS de usuários (BUY YA *et al.*, 2009). Nos sistemas de grade computacional, o elemento intermediador e escalonador principal do ambiente em questão é denominado de *resource broker* ou *broker de recursos*.

A computação em nuvem escala suas tarefas usando características das duas técnicas supracitadas a respeito de *clusters* e *grids* (BUY YA *et al.*, 2009). Nos sistemas de nuvem devido à virtualização, alocação dinâmica de serviços, interoperabilidade por meio de serviços Web e suporte a terceirizações de serviço, existem vários outros elementos que são levados em conta na atividade de escalonamento. Por sua caracterização de modelos de serviço,

sistemas de nuvem computacional precisam levar em conta os parâmetros específicos de QoS de cada cliente, de modo a cumprir os contratos de nível de serviço.

Até o ano de 2011, devido a necessidade dos sistemas de nuvem atenderem a múltiplos requisitos de QoS de clientes, bem como, trabalharem com compromissos assumidos perante contratos de nível de serviço, o escalonamento em nuvem era considerado como problema combinatório que não podia ser resolvido por programação linear, por um algoritmo ou regra mais simples (DUTTA, 2011). Desta forma, com as soluções propostas até então, não se alcançava uma solução ótima em tempo aceitável. Todavia, a partir de novos trabalhos publicados no ano 2012, o atendimento ao QoS multiobjetivo de clientes foi resolvido por meio de programação linear inteira, considerando fluxos de trabalho de clientes para um ambiente de múltiplos provedores de nuvem IaaS com dois níveis de SLA distintos (GENEZ; BITTENCOURT; MADEIRA, 2012).

A questão do escalonamento em nuvem computacional tem um notável histórico de pesquisas. Isso pode ser comprovado observando outros trabalhos da área que apresentam diversas propostas de soluções para este domínio de problema. Dentre as estratégias de escalonamento implementadas nos trabalhos publicados, são utilizados algoritmos computacionais como colônia de formigas (GAO *et al.*, 2013), orientado a custos de modo eficiente (VAN DEN BOSSCHE; VANMECHELEN; BROECKHOVE, 2013), tarefa mais curta para elemento de processamento mais rápido (ou *Min-Min*) (CHEN, H. *et al.*, 2013), tarefa mais longa para elemento de processamento mais rápido (ou *Max-Min*) (DEVIPRIYA; RAMESH, 2013), otimização de custos (MALAWSKI *et al.*, 2015), múltiplos fluxos de trabalho (MAHESHWARI *et al.*, 2014), heurística de clusterização (DELDARI *et al.*, 2014), algoritmos adaptativos para redução de custo (KANG *et al.*, 2014), teoria de jogos (DUAN; PRODAN; LI, 2014), mecanismo de leilão reverso (CATARINA, 2015), fluxos de trabalho de tarefas dependentes (BABU; KRISHNA, 2014), processo decisório de Markov (TEREFE *et al.*, 2015), Hadoop e MapReduce (CHEN, T. Y. *et al.*, 2013) (RAJU *et al.*, 2014), baseada em custo para grupos de tarefas (POOLA *et al.*, 2014), heurística FIFO (TSAI *et al.*, 2014), alocação de recursos adaptativa (ADHianto *et al.*, 2010), baseada em consumo eficiente de energia (KLIAZOVICH *et al.*, 2013), entre muitos outros. É importante destacar que todos os trabalhos mencionados possuem contribuições relevantes e importantes para o campo de pesquisa de escalonamento de tarefa para sistemas de nuvem computacional e permitem a visualização de como cada técnica, conforme o cenário experimental selecionado, pode ser profícua ao seu modo.

2.7.1 Tipos de escalonamento para nuvem computacional

A computação em nuvem dispõe de grande variedade de algoritmos de escalonamento devido ao grande número de cenários, modelos de negócio, modelos de serviços, tipo de implantação possíveis de se implementar. Nos trabalhos recentes é possível notar uma diversidade de abordagens quanto ao escalonamento para as nuvens computacionais. Isso se deve ao fato de que os diversos autores estão trabalhando em diferentes níveis de abstração da nuvem e, portanto, visualizam diferentes cenários de aplicação, de infraestrutura, de alocação de recursos, SLAs, requisitos de QoS e outros fatores mais que são considerados.

Uma característica que possibilita uma considerável diferenciação de formas de escalonamento está relacionada ao tipo de requisição solicitada ao ambiente de nuvem. Pode-se tomar por premissa que o escalonamento realizado em computação em nuvem pode possuir diferentes tipos de elementos divididos em três vertentes básicas: tarefas, recursos e serviços. Na Figura 2.4 são apresentados os tipos de requisição de acordo com o nível de abstração em que se trabalha com escalonamento para sistemas de nuvem.

Figura 2.4: Tipos de requisição em escalonamento de nuvens computacionais. (Autoria própria)

Uma dos referidos tipos de requisição é quando o conteúdo da requisição se refere a uma dada tarefa que necessita ser executada, isto é, a carga de trabalho em dados ou parâmetros de requisição serão entradas para um programa instalado no ambiente de nuvem, o qual deverá lidar com esses valores, processá-los e devolver uma resposta. Quando se considera o escalonamento de tarefas para ambientes de nuvem, considera-se modelos de serviço com níveis de abstração mais altos, tal como SaaS ou PaaS.

Uma segunda forma de requisição (ainda conforme Figura 2.4) é a de solicitação de elementos referentes a recursos, isto é, o cliente especifica parâmetros de requisição que se reportam a um nível de abstração mais baixo, como por exemplo, uma VM com 2 CPUs, 4 *Gigabytes* de memória RAM, 1 *Gigabit*/segundo de largura de banda, 2 *Terabytes* de disco virtual, entre outros. Esse tipo de solicitação leva a um escalonamento que, geralmente, manipula diretamente as características de *hardware* virtual, ou seja, de máquinas virtuais. Esse tipo de granularidade é aplicável a cenários nos quais se considera o modelo de serviço IaaS.

A terceira forma de escalonamento para nuvem (Figura 2.4), reporta-se a trabalhos cujos cenários de nuvem implementam serviços remotos. Neste contexto, a abstração de serviço pode simbolizar qualquer tipo de modelo de serviço ou composição de modelos, de forma que a sua implementação aproxima o ambiente de uma arquitetura do tipo SOA.

A atividade de escalonamento de tarefas permite o balanceamento da carga segundo diversas premissas, isto é, conforme o objetivo do escalonamento e o tipo dos elementos escalonados, porém, é comum observar que, em geral, essa atividade centra seus esforços em melhorar o desempenho geral e vazão do sistema (ZHU *et al.*, 2011). Dentre os trabalhos de escalonamento para sistemas de nuvem encontrados na literatura, é notório que podem ser divididos conforme as propostas de abordagem na atribuição de tarefas ao ambiente de nuvem.

Duas abordagens de escalonamento encontradas são as de tarefas dependentes (FARD *et al.*, 2011) ou não dependentes (DUTTA, 2011) (FANIYI; BAHSOON, 2015). Tarefas dependentes são aquelas precisam do resultado de outras tarefas para sua execução. Tarefas não dependentes são definidas pela inexistência de vínculo com quaisquer outra tarefa para sua execução.

Há também o caso de escalonamento com particionamento de tarefas em subtarefas, abordagem que torna a atividade de escalonamento ainda mais complexa e desafiadora. Nesse caso, após a divisão de uma tarefa em frações, pode-se distribuir as partes entre várias unidades

de processamento, acelerando a execução da tarefa por meio do ganho obtido com paralelismo (WILLS, 2014).

Outra abordagem encontrada na literatura de escalonamento para nuvem é a de grupo de tarefas, a qual permite o emprego de diversas estratégias para resolver os problemas que concernem as formas como o sistema deverá tratar os requisitos inerentes a um grupo de tarefas. Dentre as estratégias de execução dos grupos de tarefas, pode-se citar que tarefas semelhantes e com requisitos de QoS comuns podem ser organizadas (tanto no cliente ao submeter quanto no *broker* ao efetuar o processo de mediação) em lotes para serem despachadas para a mesma unidade de recurso computacional para serem executadas. A ideia dessa organização é a não retenção de tarefas na fila de entrada do *broker*, logo, a referida estratégia busca por possibilidades de aproveitar melhor os recursos de computação de que o provedor dispõe. Torna-se ainda mais interessante o agrupamento caso o escalonamento possa adequar os grupos de tarefas a um conjunto de regras para aquelas solicitações, de modo a otimizar ainda mais o desempenho, obedecendo aos requisitos de QoS previamente requisitados (POOLA *et al.*, 2014).

Outra vertente de escalonamento é a de fluxos de trabalho, a qual tem por objetivo aumentar a admissão de tarefas nos sistemas de nuvem (CHEN, W.; REY, 2012) (GENEZ; BITTENCOURT; MADEIRA, 2012). Nessa abordagem, é dada atenção especial aos requisitos de QoS de cada requisição, pois, o escoamento dos fluxos de trabalho para os recursos do sistema é uma tarefa não trivial na qual regras precisam ser consideradas. Relacionada a essa abordagem de escalonamento, num dos trabalhos encontrados na literatura, foi projetada uma camada de interposição para múltiplos fluxos de trabalho, a qual lida com fluxos de trabalho entrantes ao sistema, visando cumprir requisitos de QoS, aumentar o desempenho e melhorar a alocação de recursos de nuvem (MAHESHWARI *et al.*, 2014). Em outra instância de trabalho que lida com fluxos, os fluxos de trabalho foram tratados conforme o custo operacional, o qual se refere ao peso do processamento individual das tarefas e os seus custos de transmissão de dados (BARRETT; DUGGAN, 2011).

Com objetivo centrado nos ganhos em desempenho há o escalonamento de tarefas paralelas (HE *et al.*, 2012). Essa estratégia de escalonamento é frequentemente adotada em trabalhos cujo foco está nas aplicações científicas, nas quais tarefas podem ser divididas em subtarefas que serão executadas de forma paralela. As subtarefas geram atividade de computação intensa, pois, comunicam-se entre si para troca de dados e sincronização. Nesse

contexto de cenário e aplicação, a execução em paralelo tem o objetivo de acelerar o tempo de execução e de resposta ao usuário, dessa forma, focando aumento de desempenho.

Outra forma de escalonamento encontrado na literatura é o de recursos. Escalonar recursos é uma atividade ampla, pois, podem focar vários requisitos de QoS, infraestruturas, cenários, aplicações e tipos de recursos distintos. Máquinas virtuais (VMs) são recursos frequentemente encontrados nos trabalhos que lidam com modelo de serviço IaaS, porém, esse tipo de abordagem não se resume apenas ao escalonamento e organização das VMs. As máquinas virtuais são os elementos de base do sistema de virtualização de nuvem computacional e, dessa forma, a estratégia de vinculação das VMs com as tarefas é um dos fatores que pode prover melhorias em desempenho para o sistema.

Uma forma de implementar essa estratégia de escalonamento, é o desenvolvimento de multi-agentes paralelos em conjunto com implementação de um algoritmo genético modificado, os quais são utilizados para prover características dinâmicas e inteligentes ao escalonamento de VMs, provendo um balanceamento de carga melhor quando comparado com um algoritmo de escalonamento tradicional como o *Min-Min* (ZHU *et al.*, 2011).

O escalonamento de requisições de serviços para sistemas de nuvem baseiam-se, principalmente, em sistemáticas que consideram fluxos de trabalho e paralelismo dos serviços de nuvem.

Uma das formas de implementação de política de escalonamento de serviços, é o desenvolvimento de algoritmos, heurísticas e políticas de escalonamento de fluxos de trabalho de serviços de nuvem (SENNA *et al.*, 2011). Além disso, as novas heurísticas propostas buscam a melhor solução de custo/benefício em relação à negociação de requisitos de QoS e a gestão de SLA. Vários tipos de fluxos de trabalho projetados por meio de estruturas básicas permitem emular tipos diferentes de aplicação, tais como processos simples, *pipeline* de processos, processos com distribuição de dados, processos com agregação de dados, entre outros. Para o escalonamento de serviços, o modo como os serviços são distribuídos pela infraestrutura de nuvem tem impacto direto na execução dos fluxos de trabalho apresentados como carga ao sistema.

As estratégias de escalonamento de serviços são utilizadas para ambientes de nuvem híbrida, tornando o cenário experimental mais complexo (BITTENCOURT; MADEIRA, 2011). Neste caso, o algoritmo de escalonamento de fluxo de trabalho de serviços considera o custo e desempenho, logo, a estratégia utilizada baseia-se em executar o balanceamento de

carga entre um sistema de nuvem privada e um de nuvem pública observando os custos com solicitações de nuvem pública e ao mesmo tempo tentando manter os requisitos de QoS das requisições dos clientes.

De forma similar a outros sistemas computacionais distribuídos, a execução de serviços em nuvem também pode se valer dos benefícios do uso de técnicas de paralelismo. O escalonamento de tarefas dependentes com execução em paralelo (BABU; KRISHNA, 2014) é do tipo *NP-hard*, assim, além do fato de levarem em consideração a execução paralela das tarefas, também é observada a possibilidade de se executar tarefas paralelas dependentes em máquinas de *datacenters* distintos efetuando uma carga considerável de comunicações para a sincronização de dados entre as unidades de execução paralelas.

2.7.2 Alocação de Recursos em Nuvem Computacional

As políticas de escalonamento se relacionam com outro campo de pesquisas que são os trabalhos que possuem foco na forma como é realizada a alocação de recursos em sistemas em nuvem computacional. Os conceitos básicos sobre os tipos de recursos que podem ser instanciados em nuvens computacionais já foram discutidos e a base sobre escalonamento de recursos também já fora apresentada.

Os trabalhos de alocação de recursos em nuvens usam algoritmos adaptativos, oportunistas, dinâmicos, predição de carga, entre outros, para prover aproveitamento satisfatório dos recursos físicos ou virtualizados e, prioritariamente, manter a qualidade de serviço requisitada por usuários.

Os resultados experimentais das políticas de alocação de recursos podem prover análises diferentes, conforme o modo como se executa a alocação de recursos. Por exemplo, alguns pesquisadores focam na minimização de custos de aquisição de infraestrutura e na não violação de SLAs (GARG, SAURABH KUMAR; BUYYA, 2011). No referido trabalho, o modelo de serviço considerado foi o SaaS e são propostos dois algoritmos de alocação de recursos que receberam os nomes de *ProfminVmMaxAvaiSpace* e *ProfminVmMinAvaiSpace*. O primeiro algoritmo visa maximizar o lucro pela minimização do custo por meio do reuso de VMs que tenham o máximo de espaço disponível para alocar tarefas. O segundo algoritmo é o inverso, isto é, visa maximizar o lucro pela minimização de custo por meio de reuso de VMs que tenham um espaço mínimo para alocar tarefas.

Visando o trabalho com flutuações de carga de trabalho e provisionamento de recursos sob demanda com uso de algoritmos adaptativos, pesquisadores propuseram um framework para alocação de recursos baseado na plataforma de nuvem Eucalyptus (*framework* de nuvem para implementação dos modelos básicos de serviço IaaS, PaaS ou SaaS), integrado ao gerenciador de recursos Torque (trata-se de um sistema gestor de recursos virtualizados) e ao escalonador Maui (pode escalonar recursos para as requisições segundo vários algoritmos para escalonamento) (NGUYEN *et al.*, 2011). Os algoritmos adaptativos desenvolvidos levam em conta vários aspectos de informação de estado do sistema de nuvem, tais como número de tarefas em fila, tempo restante das tarefas, sobrecarga de tempo para provisionamento dos recursos, entre outras informações. Essas informações são utilizadas no processo decisório para disparar as ações adaptativas no sistema de modo a aumentar ou diminuir os recursos sob a demanda de solicitações de clientes.

Algoritmos de alocação *online* e dinâmica de recursos foram utilizados por autores em um trabalho que considerou um cenário com provedores de nuvem IaaS (COMPUT *et al.*, 2012). Nesse contexto, tarefas que são preemptíveis podem ser utilizadas para permitir a alocação de recursos com base em informações do estado atual da execução das tarefas, interrompendo-as por tempo limitado. Dois algoritmos que consideram a reserva antecipada de recursos para atender às requisições entrantes ao sistema foram desenvolvidos. O primeiro algoritmo desenvolvido foi o DCLS (*Dynamic Cloud List Scheduling*) e o segundo foi DCMMS (*Dynamic Cloud Min-Min Scheduling*). Por meio de uma avaliação de desempenho, ambos algoritmos desenvolvidos na pesquisa foram comparados a um terceiro já empregado em outros trabalhos, o FCFS (*First-Come-First-Serve*). Na avaliação realizada o algoritmo DCMMS superou o FCFS e o DCLS demonstrando que a técnica de atualização de informação *online* de servidores gerentes da nuvem, bem como a alocação de recursos antecipada podem prover ganhos em desempenho em provedores IaaS com infraestrutura de nuvem heterogênea. Um trabalho anterior do mesmo grupo de pesquisa investigou alocação adaptativa de recursos e fez uso de tarefas preemptíveis usando dois outros algoritmos o ALS (*Adaptive List Scheduling*) e o AMMS (*Adaptive Min-Min Scheduling*). Conforme resultado experimentais foi possível verificar, também neste artigo, o algoritmo que usou escalonamento com uso da técnica *min-min*, isto é, o AMMS, superou o ALS nos experimentos executados.

O mercado de provedores IaaS e seus usuários avaliam a alocação dos recursos segundo os custos a serem pagos. Essa abordagem se adequa aos padrões de mercado que comercializam recursos de infraestrutura como serviço sob demanda (YANG; YIN; LIU, 2011).

Num trabalho cujo desenvolvimento da alocação de recursos (mais especificamente escalonamento de recursos) foi o ponto de abordagem principal (YANG; YIN; LIU, 2011) e o algoritmo desenvolvido procurava, entre distintos provedores IaaS, aquele com o menor preço e a melhor qualidade de serviço. Um protocolo de escalonamento de recursos baseado em custos foi criado e segue um algoritmo confirmação de duas fases, com execução atômica do processo de escalonamento, isto é, ou todos os recursos reservados são efetivamente confirmados ou todos esses recursos são cancelados (YANG; YIN; LIU, 2011). A combinação do protocolo especializado com a política de escalonamento voltada à minimização de custos para o usuário do sistema de nuvem atingiu o objetivo de melhorias em qualidade de serviço proposto pelos autores, os quais usaram um *testbed* para apresentar os resultados experimentais e exprimir essa conclusão final.

O emprego da técnica de alocação de recursos de IaaS baseada em sistemas de leilão tem sido adotada por alguns pesquisadores. Essa metodologia de alocação de recursos pode contribuir para melhorar o desempenho geral do sistema de nuvem para ambas as partes envolvidas no processo de negócio (provedores e clientes) (CLOUD *et al.*, 2011). Nessa proposta, o melhor emprego dos recursos totais disponíveis é uma característica chave para se obter melhoria do desempenho global do sistema. A otimização usada no referido trabalho foi a denominada otimização social, provendo o efeito de *ganho-ganho*. Os resultados obtidos por meio de simulação revelaram que essa técnica pode melhorar quase cinco vezes o desempenho do sistema do que uma técnica que não considere a sistemática de leilões.

Em um outro trabalho de escalonamento de recursos, um modelo de balanceamento adaptativo para alocação de recursos baseado no algoritmo de colônia de formigas utilizou o recurso de monitoramento das VMs do ambiente de IaaS e, ao julgá-las sobrecarregadas, o sistema de alocação de recursos, por meio de uma implementação do algoritmo de colônia de formigas, provê alocação de recursos rápida e adequada para que os recursos livres de um nó sejam suficientes para atender aos requisitos da carga (LU; GU, 2011). Segundo o trabalho, uma característica essencial para um ambiente adaptativo é que o mesmo deve se programar automaticamente de modo a mudar as cargas de local e dinamicamente alocar os recursos necessários para resolver o problema de quantidades de “cargas incertas” chegando ao sistema. Os resultados apresentados no artigo por meio de simulação, corroboram com as ideias propostas pelos autores sobre alocação de recursos dinâmica e adaptativa, de modo que

atribuem ao monitoramento de certos parâmetros das VMs o desempenho e balanceamento de carga alcançado pelo sistema de nuvem do cenário considerado.

2.8 Qualidade de Serviço em Nuvem Computacional

Sistemas computacionais que atendem a múltiplos usuários podem implementar uma sistemática para considerar restrições de usuário que são expressas como parâmetros ou requisitos de qualidade de serviço (do inglês, *Quality of Service* ou QoS). A qualidade de serviço é a característica que mensura a efetividade de um sistema em prestar o seu serviço aos clientes. Dependendo do tipo de sistema abordado e de suas características, os parâmetros de qualidade de serviço podem ser diferentes. A QoS pode ser diretamente percebida e avaliada pelos usuários de um sistema computacional por meio de variáveis de resposta que ficam expostas, de modo que os usuários passam a ter um certo grau de sensibilidade sobre o desempenho geral do sistema computacional em questão.

Os sistemas de computação em nuvem também necessitam atender a critérios de qualidade de serviço. Nos modelos de negócio comercial, as empresas provedoras de serviço de nuvem estabelecem com cada um de seus usuários um contrato de nível de serviço ou SLA (*Service Level Agreement*). Esses SLAs definem um conjunto de requisitos de QoS, o quais são descritos, entre outras formas, por meio de valores mínimos e máximos de desempenho e custo que os serviços prestados pelo provedor devem seguir (WU, L.; GARG; BUYYA, 2011).

Os parâmetros ou requisitos de QoS podem ser diferentes de um usuário para outro (BUYYA; RANJAN; CALHEIROS, 2010), dessa forma, sistemas de nuvem precisam se ajustar dinamicamente obedecendo a muitas restrições distintas o que se torna uma tarefa desafiadora. Assim, considerando múltiplas restrições de muitos clientes conforme níveis de serviço acertados contratualmente, os provedores de nuvem podem continuar a garantir o cumprimento de SLAs, entregando o desempenho esperado pelos usuários.

A QoS de sistemas de nuvem possui duas vertentes bem definidas, a do cliente e a do provedor (WU, L.; GARG; BUYYA, 2011). Ao se considerar a QoS do lado cliente, é comum verificar que os parâmetros de QoS visam vantagem para esse ator do ecossistema de nuvem em específico, isto é, os parâmetros assumidos normalmente são: tempo de resposta, preço (ou custo), atraso máximo, confiabilidade, segurança, classificação do serviço em *ranking*, indicador de satisfação de usuários anteriores, entre vários outros (KLANÈNIK; JERMAN, 2010). *A priori*, os parâmetros de QoS centram-se em elementos de tempo e de custo que são os principais benefícios diretos que o usuário pressupõe que lhe sejam vantajosos. Além disso,

o usuário se interessa pelos parâmetros de qualidade que podem ser percebidos por ele durante o uso dos serviços de nuvem, fato que lhe dá maior sensação de que as restrições solicitadas estejam sendo cumpridas.

Analisando os aspectos relacionados aos provedores de nuvem, sabe-se que os mesmos possuem seus parâmetros de QoS internos, isto é, os requisitos de QoS dos provedores (principalmente os comerciais) centram-se nos ganhos, visando sempre aumentar a lucratividade dos serviços prestados, e na otimização do uso de recursos (economia), para que o sistema possa atender o máximo de clientes simultaneamente (WU, L.; GARG; BUYYA, 2011). Alguns parâmetros de QoS internos considerados pelos provedores são os que se relacionam com a infraestrutura do sistema, por exemplo, o tempo de instanciação de um serviço, tempo de inicialização de uma máquina virtual, tempo de fila de uma máquina virtual, ganho por hora com cada recurso, porcentagem de carga de trabalho em cada VM ligada no *datacenter*, vazão de tarefas executadas pela nuvem por minuto ou por hora, entre muitas outras possibilidades. O objetivo desse controle de QoS do lado provedor é manter, conforme citado, o controle sobre o centro de custos (simbolizado, principalmente, pelos equipamentos da infraestrutura, o software instalado para suporte às aplicações) em relação aos ganhos obtidos a partir da disponibilização do sistema de nuvem aos usuários.

Os provedores podem considerar parâmetros de QoS internos distintos de acordo com o tipo de carga de trabalho que recebem. Alguns dos tipos principais de carga de trabalho que são considerados na literatura são de fluxos de trabalho, requisições individuais ou requisições em grupos (que podem ser dependentes ou independentes) (WU, L.; GARG; BUYYA, 2011).

Os aspectos estudados sobre requisitos de QoS são importantes para caracterizar melhor os trabalhos científicos que se preocupam em reproduzir detalhadamente o ecossistema de sistemas de nuvem computacionais, pois, a QoS é um dos elementos por meio dos quais se pode aferir e entender qual o julgamento é feito do sistema computacional a partir dos serviços que são prestados pelo mesmo.

Existem parâmetros de QoS para o cliente e existem parâmetros de QoS para o provedor. Conforme explanado, para o cliente interessam economia de gastos e o melhor desempenho do serviço. Para o provedor interessam o maior ganho financeiro possível com a menor degradação de seus recursos de infraestrutura computacional. Logo, encontrar a sinergia sobre as duas vertentes de QoS (do cliente e do provedor) é um dos grandes desafios da computação em nuvem.

2.9 Simulação em Computação em Nuvem

No cenário atual de pesquisa em computação em nuvem, várias técnicas são empregadas pelos pesquisadores para validar suas proposições e teses. Entre as técnicas adotadas estão a prototipação, *benchmarks*, monitores, modelos analíticos e simulação (JAIN, R., 1991).

Observando-se os trabalhos em sistemas de nuvem recentes (CHUNG *et al.*, 2013) (SOTIRIADIS; BESSIS; ANTONOPOULOS, 2013a), é possível verificar que a simulação tem sido usada para a proposição e validação de novos modelos computacionais. Alguns pesquisadores da área de computação em nuvem justificam que a preferência de se utilizar a técnica de simulação ocorre porque o uso de um sistema real de nuvem para experimentação incorre em altos custos para se alocar uma grande quantidade de recursos de computação por períodos longos de tempo. Além disso, torna-se difícil executar a repetição de experimentos com o sistema nas mesmas condições, os experimentos tornam-se totalmente dependentes da infraestrutura do provedor (DEVI; SUJAN, 2014).

Segundo autores de um simulador de sistemas de computação em nuvem, a tarefa de modelagem de cenários distintos de políticas de provisionamento de nuvem, cargas de trabalho de aplicações e modelos de desempenho dos recursos computacionais é algo bastante desafiador, pois, grande é o nível de detalhamento que o trabalho precisa envolver para conseguir se aproximar de um sistema real (Calheiros, Ranjan, & Beloglazov, 2011).

No campo de pesquisa de computação em nuvem há muitas ferramentas de simulação, isto é, existem alguns simuladores mais usados e muitas ferramentas derivadas (ou extensões) destes, principalmente dos simuladores que possuem licença de *software* livre (SAKELLARI; LOUKAS, 2013) (DEVI; SUJAN, 2014). Alguns dos simuladores que frequentemente são utilizados nos trabalhos de pesquisa em computação em nuvem (JAIN, P., 2013) são: CloudSim (CALHEIROS *et al.*, 2009, 2011), CloudAnalyst (WICKREMASINGHE; CALHEIROS; BUYYA, 2010), GreenCloud (KLIASOVICH; BOUVRY, 2012), iCanCloud (VÁZQUEZ-POLETTI *et al.*, 2012), MDCSim (JAIN, P., 2013; LIM *et al.*, 2009), NetworkCloudSim (GARG, S. K.; BUYYA, 2011), EMUSim (CALHEIROS *et al.*, 2013), OpenCloud (GROSSMAN *et al.*, 2009), Open Cirrus (AVETISYAN *et al.*, 2010) entre outros. É importante destacar que o CloudAnalyst e NetworkCloudSim são extensões construídas sobre a plataforma CloudSim (CALHEIROS *et al.*, 2011), a qual possui ainda muitas outras extensões publicadas até o presente momento.

Os simuladores de computação em nuvem geralmente focam em algumas características essenciais as quais se deseja representar nos cenários simulados, assim, entre as características consideradas estão: provisionamento de recursos, segurança, provisionamento de recursos com base em consumo de energia (mais direcionados para pesquisa em *Green Cloud Computing*), entre outros (DEVI; SUJAN, 2014). Alguns dos simuladores são orientados a eventos discretos, como é o caso do CloudSim e suas extensões, enquanto outros permitem a simulação via *software* ou também em *hardware*, tal como OpenCloud e Open Cirrus, por meio do emprego de hipervisores e *middleware* para a criação de um ambiente de nuvem usando equipamentos restritos para teste e validação de modelos (ZHAO *et al.*, 2012) (SAKELLARI; LOUKAS, 2013).

Um *toolkit* que vem se destacando para modelagem e simulação de sistemas de nuvem é o CloudSim (Calheiros, Ranjan, & Beloglazov, 2011). Este simulador oferece a possibilidade de modelar o sistema (infraestrutura) e o comportamento de nuvem computacional, bem como admite elementos externos como cargas de trabalho e latências de rede, logo, acaba por conseguir conceber grande parte dos componentes de um ecossistema de nuvem.

Desenvolvido em linguagem de programação Java, o CloudSim disponibiliza, através de uma vasta API, classes que modelam os principais atores e componentes físicos e lógicos de sistemas de nuvem, tais como hosts, *datacenters*, máquinas virtuais, políticas de escalonamento de tarefas e de alocação de recursos, aplicações, tarefas e *brokers*. Na questão de modelagem de nuvens computacionais, o CloudSim admite formulação de agrupamentos de *datacenters* para formação de nuvens privadas, híbridas, públicas, comunitárias e federações.

O simulador de nuvem CloudSim tem sido utilizado por diversos centros de pesquisa, como por exemplo o HP Labs e muitos trabalhos científicos têm se empregado o CloudSim para a implementação de suas proposições para ambientes de computação em nuvem (WICKREMASINGHE; CALHEIROS; BUYYA, 2010) (BUYYA; RANJAN; CALHEIROS, 2010) (BELALEM; TAYEB; ZAOU, 2010) (SINDHU; MUKHERJEE, 2011).

A partir da implementação do simulador CloudSim, seus autores, em conjunto com outros pesquisadores interessados na área de simulação em computação em nuvem, criaram novas ferramentas visuais a partir da versão inicial do CloudSim, como é o caso do CloudAnalyst (WICKREMASINGHE; CALHEIROS; BUYYA, 2010), o qual permite simular aplicações de nuvem de grande escala a partir de várias opções de configuração de implantação. É possível verificar, com auxílio do CloudAnalyst, a distribuição de aplicações e de serviços

em um ambiente de infraestrutura de nuvem, realizando otimizações no desempenho das aplicações, por meio de uso de *brokers* de serviço numa ferramenta cujo ambiente é totalmente visual ao invés da interface de caracteres de texto do CloudSim original (WICKREMASINGHE; CALHEIROS; BUYYA, 2010).

Pelas características oferecidas por simuladores como o CloudSim é que a simulação tem sido largamente empregada em trabalhos de nuvem, isto é, com um sistema de nuvem simulado pode-se testar e validar algoritmos de escalonamento, infraestruturas diversas, composições de tarefas e de serviços das mais variadas formas. A simulação viabiliza a validação das teses apresentadas pelos pesquisadores sem a necessidade da montagem de um protótipo de sistema de nuvem que, para alcançar um mínimo de realismo, requisita custos consideráveis para sua construção ou locação e demanda um considerável tempo de pesquisa e preparação do ambiente de testes.

A simulação pode ser empregada como metodologia principal ou secundária de trabalhos científicos de computação em nuvem. Tal técnica oferece vários benefícios desde a facilidade de implementar os componentes de um ecossistema de nuvem até a possibilidade de testar algoritmos e políticas de escalonamento de nuvem sem a necessidade dos equipamentos físicos e das plataformas de virtualização e de *software*.

2.10 Considerações finais

No presente capítulo foram abordados tópicos importantes para o entendimento geral de sistemas de computação em nuvem. Assim, foram discutidos: a arquitetura de nuvem computacional a partir do modelo de referência do NIST, os modelos de implementação de nuvem, os modelos básicos de serviço, os tipos de estrutura adotados, escalonamento e alocação de recursos, conceito e considerações sobre parâmetros de qualidade de serviços e a importância da metodologia de simulação para os trabalhos envolvidos em computação em nuvem.

Dado que os sistemas de nuvem computacional estão focados, em todos os seus modelos de negócio, no provimento de serviços segundo um modelo, isto é, IaaS, PaaS ou SaaS, pode-se afirmar que esse tipo de sistema computacional tem o objetivo de implementar serviços remotos para seus clientes em diversos níveis de abstração distintos.

Alguns trabalhos já estão modelando os serviços de nuvem como serviços Web, logo, o modelamento dos serviços de nuvem considera uma arquitetura SOA. Os serviços Web são um padrão estabelecido e absorvido pelo mercado e academia para a elaboração de sistemas de computação fracamente acoplados e padronizados com relação à comunicação de dados.

O próximo capítulo apresenta uma visão geral de arquitetura orientada a serviços e um detalhamento sobre o desenvolvimento de serviços Web.

---

3. Arquitetura Orientada a Serviços

3.1 Considerações iniciais

“Arquitetura orientada a serviços é um estilo arquitetônico (framework) empregado no desenvolvimento de aplicações corporativas. Tal padrão prega a componentização dos serviços em unidades de funcionalidade de negócio. O uso dessa estratégia baseada em serviços tem várias vantagens, como auto-descrição, independência de plataforma, integração independente de linguagem, reusabilidade de componentes, desenvolvimento ágil de aplicações, integração com sistemas legados.” (DHARA, 2015)

Alguns trabalhos publicados têm destacado a importância dos sistemas distribuídos baseados em SOA (KARNOUSKOS *et al.*, 2012) e entre os tipos de implementação de SOA os mais conhecidos são os serviços Web. As empresas estão aderindo ao uso de serviços Web pela facilidade de interoperação com sistemas legados, pela modularidade oferecida para as alterações nas regras de negócio e pela forma como os serviços tornam-se unidades de execução de partes do negócio de modo independente de plataforma. Algumas abordagens apresentam princípios e práticas de projeto relacionados com a criação de ambientes baseados em SOA (PSIUK; BUJOK; ZIELIŃSKI, 2012), as empresas modernas que implementam esses ambientes criam um Barramento de Serviços Empresariais (do inglês, *Enterprise Service Bus* ou ESB). No ESB é disponibilizada uma biblioteca de serviços Web que compõe todos os processos ou regras de negócio internos ou externos que seus sistemas necessitam, criando um ambiente homogêneo para a governança de suas aplicações e infraestrutura. Um dos principais diferenciais oferecidos pela SOA é a possibilidade de modularizar grandes aplicações como um conjunto de vários serviços Web, os quais interoperam com qualquer plataforma de programação que siga os protocolos padronizados pela arquitetura. Qualquer empresa, aplicação, infraestrutura ou mesmo sistema distribuído, que siga as diretrizes essenciais descritas nos padrões SOA, terá alcançado o estado da arte em termos da organização estratégica de Computação e de Tecnologia da Informação (TI).

Com objetivo de prover uma visão geral a respeito de arquitetura orientada a serviços e apresentar também um conteúdo essencial a respeito da implementação de serviços Web, este capítulo apresenta fundamentos básicos de SOA e serviços Web, seus protocolos de funcionamento, modelo de serviço e qualidade de serviço em serviços Web.

3.2 Serviços Web

“Serviços Web são componentes auto-contidos, auto-descritos e reutilizáveis. Podem ser descobertos, compostos, e invocados via Internet. Suas funcionalidades geralmente estão associadas com as regras de negócio de um sistema.”(REZK; FOUFOU, 2015)

Como SOA vem se tornando um novo paradigma para os ambientes corporativos e também para as pesquisas recentes em computação, as unidades de regras de negócio dos sistemas passam a ser entendidos e implementados como serviços Web, porque ficam disponibilizados via rede para uma *Intranet*, *Extranet* ou para toda a Internet. Para o contexto de implementação dos serviços Web são considerados padrões abertos que regem a arquitetura de serviços Web, a qual é uma implementação de SOA. Esses padrões abertos se formalizam como um conjunto de recursos de comunicação distribuído, podendo-se citar entre os principais componentes de desenvolvimento a Linguagem de Descrição de Serviços Web (WSDL), o Protocolo de Acesso a Objetos Simples (SOAP) e o registro de Descrição Universal, Descoberta e Integração (UDDI).

Para o correto funcionamento de um ambiente SOA baseado em serviços Web, há uma ordem de eventos para que o procedimento de serviço seja executado com sucesso. Pode-se definir, de modo simples, a seguinte ordem de eventos: publicação, descoberta e consumo do serviço Web.

Na Figura 3.1 é possível observar como se dá a intercomunicação entre os três principais componentes da arquitetura de serviços Web (Cliente, Servidor e Serviço de Diretório UDDI). A execução dos fluxos para publicar serviço, gerar de apêndice de descrição WSDL, consultar serviço Web e o consumo do serviço via protocolo SOAP representam o mecanismo de funcionamento desse modelo de arquitetura orientada a serviços (TANENBAUM; STEEN, 2006).

Figura 3.1: Dinâmica de comunicação do ambiente de serviços Web. (TANENBAUM; STEEN, 2006)

Seguindo sequência da dinâmica de comunicação (Figura 3.1) do cliente com o servidor segundo o modelo de serviço Web (fase de descoberta), os fluxos de comunicação são iniciados por uma aplicação cliente. O primeiro passo consiste em consultar o serviço desejado via Internet. Para tanto, a aplicação cliente envia uma mensagem a um serviço de diretório UDDI contendo os parâmetros de busca por serviços. O diretório UDDI consulta sua base de dados e retorna ao cliente um ou mais resultados de acordo com os parâmetros de busca. O cliente recebe uma mensagem WSDL contendo a descrição completa do serviço Web e pode então escolher qual o serviço Web de sua preferência. Com os metadados WSDL recebidos pelo cliente, está finalizada a descoberta do serviço. Ao final dessa comunicação o cliente já possui todas as informações que precisa para consumir o serviço Web.

A última etapa de comunicação é a do consumo do serviço Web, na qual o cliente envia uma chamada direta ao serviço Web em conjunto de dados referentes aos parâmetros de entrada para o serviço. O provedor de serviços, recepciona a mensagem do cliente formatada com protocolo de comunicação escolhido. O próximo passo é o encaminhamento da requisição para sua infraestrutura de modo a atender o cliente, executando o serviço Web solicitado. O retorno ou resposta resultante da execução do serviço Web é enviado ao cliente na sequência, também em formato de pacotes SOAP, com os valores resultantes da prestação do serviço.

3.2.1 Protocolo de Comunicação SOAP

Entre os elementos que podem ser empregados para a implementação de serviços Web está o protocolo de comunicação de troca de mensagens. Um dos protocolos possíveis de se utilizar é o SOAP (*Simple Object Access Protocol*). Tal protocolo representa as regras de comunicação direta utilizada entre cliente e provedor durante o processo de troca de mensagens referentes aos dados e informações trocadas na operação de chamada ao serviço Web. As requisições de serviço, como são mais conhecidas, são mensagens formatadas com o protocolo de comunicação que funcionam como chamadas a procedimento remoto (do inglês, *Remote Procedure Call* ou **RPC**) executadas pela Internet (TEWARI, 2012). O protocolo SOAP implementa um protocolo de transporte de alto nível (camada de aplicação) baseado no Protocolo de Transferência de Hipertexto (do inglês, *Hyper Text Transfer Protocol* ou HTTP), Protocolo de Transferência de Hipertexto Seguro (do inglês, *Hyper Text Transfer Protocol Secure* ou HTTPS) e Protocolo Simples para Transferência de Correio (do inglês, *Simple Mail Transfer Protocol* ou SMTP).

O protocolo SOAP possui uma estrutura de comunicação baseada na transmissão e recepção de dados entre cliente e provedor com base em *tags* baseadas em XML (do inglês, *eXtensible Markup Language*) (TANENBAUM; STEEN, 2006). O protocolo SOAP pode ser utilizado como padrão de comunicação para ambientes de serviços Web e seu objetivo é permitir comunicação entre os pares comunicantes (cliente e provedor) sem que eles tenham conhecimento dos detalhes de implementação um do outro. Essa última característica do SOAP é que torna os ambientes distribuídos baseados em serviços Web interoperáveis. As mensagens SOAP usam meta-*tags* XML, assim, a sintaxe XML serve para descrever a mensagem e o conteúdo é colocado em meio aos *tags* customizados de cada documento

3.2.2 Protocolo de Descoberta de Serviço WSDL

Os serviços Web possuem um modelo de serviço no qual incluem-se algumas etapas que precedem o consumo do serviço, conforme já discutido. Entre as referidas etapas está a de publicação do serviço Web (TEWARI, 2012), na qual é feito o uso do protocolo WSDL para o registro da descrição de um serviço em um diretório UDDI.

O WSDL é um protocolo baseado em XML que descreve os métodos, parâmetros de entrada e de saída, tipos de dados e protocolo padrão de comunicação considerados por serviço. Entre os *tags* XML descritos num documento WSDL podem-se notar as mensagens que o protocolo do serviço aceitará (requisição, resposta e tipo de porta), a descrição de vinculação

do serviço com os protocolos necessários à sua execução é informada na *tag binding* e, por fim, na *tag service* é informado o nome, a descrição do serviço e sua URL (*Uniform Resource Locator*) de acesso para que o mesmo possa ser indexado num diretório UDDI.

3.2.3 Serviço de Diretório UDDI

O UDDI funciona como um repositório de informações sobre provedores e seus respectivos serviços Web disponíveis. No modelo de serviços Web padrão, o cliente consulta o diretório UDDI quando necessita efetuar a descoberta de serviços Web para determinada funcionalidade de sua aplicação, por exemplo, consulta a preços de hotéis. Logo, pode-se conceituar UDDI (TEWARI, 2012) como um banco de registros de provedores e de seus serviços Web disponibilizados.

O serviço de diretório UDDI pode ser mais bem detalhado se forem levadas em consideração as seguintes características desse serviço (POINT, 2016):

- Serve como um registro distribuído de serviços Web;
- É independente de plataforma;
- Comunica-se com vários tipos de protocolos de comunicação distribuída, tais como SOAP, CORBA (do inglês, *Common Object Request Broker Architecture*), Java RMI (do inglês, *Remote Method Invocation*) e outros;
- Descreve os serviços em sua base de dados através de WSDL;
- É considerado um dos três componentes fundamentais do modelo de serviço Web juntamente com o protocolo SOAP e a linguagem WSDL;
- É um padrão aberto da indústria para o estabelecimento de intercâmbio de negócios pela Internet.

O serviço de diretório UDDI pode ser dividido em duas partes. A primeira composta pelos registros de todos os serviços Web descritos em linguagem WSDL através de *metadados* baseados em XML. A segunda composta por um conjunto de definições de tipos de portas de acesso para manipulação e busca nos registros do banco de dados UDDI (POINT, 2016).

Os tipos de informações que podem ser registrados em um diretório UDDI podem ser distintos. Embora o principal elemento a ser procurado pelas aplicações cliente sejam os serviços Web, devem-se considerar outros tipos de informações a respeito do provedor, sua descrição e outras informações de base que necessitam ser armazenadas no repositório UDDI. Esses elementos se subdividem em três categorias: páginas brancas, páginas amarelas e páginas

verdes. Nas páginas brancas registram-se informações básicas sobre a empresa e o seu tipo de negócio, portanto, dados de endereçamento e contato fazem parte da informação que compõe esse registro. As páginas amarelas possuem um tipo de informação um pouco mais técnica, descrevendo os tipos de tecnologias que a empresa dispõe para executar processos de negócios com os seus parceiros e clientes. As páginas amarelas podem conter ainda, códigos referentes a padrões industriais de identificação, codificação de produtos e serviços, entre outras possibilidades. A última categoria é a que realmente interessa aos sistemas que querem interoperar com a empresa ou provedor, tratam-se das páginas verdes, as quais contêm as informações técnicas sobre um serviço Web, permitindo a um cliente se conectar com o serviço Web logo que o mesmo tiver sido descoberto. Informações constantes de páginas verdes são as diferentes interfaces de conexão para com o serviço, URLs de acesso e todas as demais informações de descoberta e dados similares que são requisitos para que se possa enviar uma requisição ao serviço Web (POINT, 2016).

3.2.4 Composição de Serviços

O conceito de composição de serviços tem como base de sua conceituação a questão dos serviços que, para poderem responder aos seus requisitos funcionais, necessitam da consulta a vários outros serviços Web antes de prover uma resposta (MALLAYYA; RAMACHANDRAN; VISWANATHAN, 2015). Assim, o processo de implementação de um serviço composto por vários outros serviços Web pode ser chamado de composição de serviço. Alguns autores cujos trabalhos também definem o conceito de combinação de serviços Web, descrevem-na como uma interface de serviço cujas operações são combinadas com as de outros serviços Web para fornecer nova funcionalidade (COULOURIS; DOLLIMORE; KINDBERG, 2012). O exemplo clássico de composição é o de serviço de agente de viagens, o qual consulta vários outros serviços para cumprir o seu papel como o serviço de reserva de voo, aluguel de carro e reserva de hotel. Os serviços da composição são utilizados para tentar retornar os melhores valores dos serviços de viagem e turismo conforme os parâmetros e restrições que foram solicitados por um cliente (COULOURIS; DOLLIMORE; KINDBERG, 2012). A composição é baseada em protocolos de coordenação que asseguram a execução das etapas do serviço composto, de acordo com uma determinada ordem e no devido tempo para que o mesmo possa ser executado com sucesso (TANENBAUM; STEEN, 2006).

A implementação de composição de serviços não é uma atividade trivial porque muitas vezes inclui vários serviços de provedores distintos que devem colaborar para o objetivo comum de devolver uma resposta ao cliente conforme o esperado de um serviço Web

individual. Para abordar tal complexidade de problema, alguns protocolos de confirmação podem ser utilizados, tais como o protocolo confirmação de duas ou de três fases (*commit protocol*). Desse modo, os processos envolvidos numa composição estabelecem entre si os papéis de coordenador e subordinados. Desta forma, há sempre um elemento executando o protocolo de coordenação e confirmação e outros elementos subordinados executando partes do processamento, provendo suas respostas e confirmações para as partes finalizadas de um serviço composto (TANENBAUM; STEEN, 2006).

3.3 Qualidade de Serviço

Prover serviços Web para os clientes e atender a grande demanda de execução solicitada por uma rede de grandes proporções, tal como a Internet, é um grande desafio. Além disso, os provedores precisam se preocupar com o fato de que cada cliente prioriza ou valoriza aspectos diferentes do processo de execução do serviço para que ele seja satisfatório. Conforme já discutido anteriormente no capítulo sobre computação em nuvem, a efetividade de um serviço prestado de forma adequada ao cliente é obtida a partir de atributos ou parâmetros de QoS. Nas arquiteturas orientadas a serviços não é diferente, pois, cada cliente, por mais que venha a receber os mesmos conteúdos de retorno de um serviço, pode possuir particularidades de requisição que o diferenciam dos demais solicitantes.

Seja por requisitos de temporização como por requisitos não funcionais que são enviados como parâmetros aos serviços Web, essas solicitações são consideradas como expressões de requisitos de QoS (TANENBAUM; STEEN, 2006). Os requisitos de QoS para serviços Web podem ainda ser divididos em duas categorias: requisitos de QoS dependentes de usuário e requisitos de QoS independentes de usuário (ZHENG; ZHANG; LYU, 2014).

Os requisitos dependentes de usuário variam de acordo com os interesses particulares de cada cliente. Nesse caso, os requisitos mais comuns considerados são tempo de resposta, probabilidade de falha, vazão e outros. Parâmetros de QoS dependentes de usuário podem ser influenciados por suas conexões de Internet ou até por ambientes de usuário heterogêneos que fazem com que os indicadores de QoS venham a variar muito de usuário para usuário (ZHENG; ZHANG; LYU, 2014).

Os requisitos de QoS independentes de usuário são mais voltados para a comercialização dos serviços ou adequação dos mesmos ao mercado de serviços Web. Alguns exemplos desses requisitos são o preço, popularidade (ou *ranking*) e outros que podem auxiliar

na classificação de um serviço Web em meio aos seus concorrentes similares. Muitos trabalhos têm se dirigido a abordagens distintas de QoS em serviços Web, tais como os que lidam com seleção de serviço, composição de serviço ótima, serviço tolerante a falhas, recomendação de serviço, entre outras. A maioria dos trabalhos leva em consideração atributos de QoS independentes de usuário, pois, os dependentes de usuário representam um, dado que o princípio de localidade do cliente faz toda a diferença nos experimentos de avaliação de serviços Web(ZHENG; ZHANG; LYU, 2014).

3.4 Considerações finais

Neste capítulo foram apresentados o conceito de arquitetura orientada a serviços e uma breve introdução à trabalhos recentes que pesquisam SOA com abordagem em serviços Web. A seguir, proveu-se a definição de serviços Web como implementação de SOA e todos os seus componentes funcionais básicos foram apresentados, isto é, o protocolo de comunicação SOAP, o protocolo de publicação e descoberta de serviço WSDL e o serviço de diretório UDDI. Além disso, tópicos adicionais sobre serviços foram também discutidos, tais como a composição de serviços e a qualidade de serviços.

Os serviços Web são um dos padrões de mercado para criação de interoperabilidade de aplicações baseadas em protocolos de Internet e na padronização de interfaces entre as partes comunicantes. Como os acessos aos serviços Web são, em geral, realizados via Internet, atualmente esses serviços vêm sendo implantados em *datacenters* de nuvens computacionais, de modo que os requisitos de QoS ficam ainda mais relevantes, sejam eles dependentes ou não de seus usuários. A integração de computação em nuvem e serviços Web gera inúmeras possibilidades de expansão em pesquisas científicas para validar as novas possibilidades de implementação desse tipo de ambiente computacional.

O próximo capítulo trata de um assunto mais específico e próximo da hipótese e implementação desta tese, isto é, os *Cloud Brokers*. Será provido o conceito fundamental desse componente dos ambientes de computação em nuvem e uma discussão sobre o relacionamento da comunicação entre *brokers* e os demais elementos que compõem o ecossistema de nuvem. Além disso, são apresentados os principais tipos de implementação de *cloud brokers*.

---

4. Cloud Brokers

“O Broker de Nuvem atua como um intermediador entre o clientes e provedores. Seu objetivo é auxiliar os clientes com a complexidade da oferta de serviços de nuvem, podendo também criar serviços em nuvem com valor agregado”. (BUMPUS, 2013)

Tanto os ambientes de computação em nuvem quanto os ambientes de SOA baseados em serviços Web podem possuir *brokers*, principalmente pelo fato de que ambos os ecossistemas são baseados na provisão de serviços aos clientes e, dada a grande quantidade de provedores disponível, torna-se interessante que um mediador auxilie na vinculação das tarefas dos clientes com os serviços dos provedores.

A missão de um *broker* é a de receber as solicitações de clientes com requisitos específicos, selecionar um ou mais provedores que melhor se adaptem às necessidades do cliente, combinando os requisitos específicos com várias ofertas de provedores distintos e, se o processo de seleção obtiver sucesso, escalonar a tarefa do cliente ao serviço remoto (CHANDRASEKAR, 2014).

Brokers geralmente possuem comunicação com alguns dos elementos de sistemas de nuvem, como o coordenador de nuvem e, em alguns casos, possuem acesso direto às informações de estado de VMs ou de serviços do provedor em questão. Dessa forma, se faz necessário todo um tramite de comunicação de dados constante entre o *broker* e os provedores por ele representados. Isto é, dada a natureza dinâmica do ambiente, é preciso atualizar a base de informações do intermediador a todo momento, de modo a ter a maior confiabilidade possível a respeito dos dados consultados sobre serviços e recursos, antes de escaloná-los para uma requisição cliente.

Na próxima seção, uma explanação mais detalhada da comunicação de dados envolvendo um *cloud broker* e outros componentes de um ecossistema de nuvem é apresentada.

4.1 Comunicação em Brokers de Nuvem Computacional

Entre o *broker* e os provedores de nuvem ocorrem fluxos de comunicação e de negociação quando da execução do processo de intermediação. Essa comunicação pode ser representada por um conjunto de mensagens que vão desde a consulta dos serviços e *datacenters* disponíveis até a obtenção do resultado da tarefa e seu encaminhamento até o cliente.

Usando como base a Figura 4.1, a qual foi traduzida e adaptada de um trabalho que apresenta uma metodologia de gerenciamento eficiente para *cloud brokers* (RAJ, 2011), são apresentadas as comunicações entre o *cloud broker* e outros componentes internos dos *datacenters* de provedores de nuvem.

Figura 4.1: Diagrama de sequência de comunicação em sistemas de nuvem. Adaptado de (RAJ, 2011)

No diagrama de sequência da Figura 4.1 é possível verificar uma sequência de mensagens que simbolizam a comunicação entre componentes do ambiente computacional de nuvem. Esses componentes necessitam intercambiar dados e informações e, neste contexto, é possível verificar e evidenciar o papel importante de negociação do *cloud broker*. Inicialmente, os *datacenters* se registram em um serviço de diretório denominado Registro CIS (*Cloud Information Service*). Em seguida, o *broker* inicia suas consultas ao Coordenador de Nuvem para saber da existência dos *datacenters* disponíveis. Através de uma pesquisa no registro CIS o Coordenador recebe a resposta dos *datacenters* disponíveis e a repassa ao *broker* que consulta as características de configuração dos *datacenters* enviando-lhes uma mensagem. Por meio de suas regras e algoritmos de seleção, o *broker* irá depurar o resultado e criar uma lista dos *datacenters* com os serviços que satisfazem os requisitos das tarefas de seus clientes. Depois disso, o *broker* envia uma mensagem contendo a tarefa a ser executada para o Coordenador de Nuvem que deverá escalonar os recursos de acordo com as restrições enviadas na mensagem

do *broker*. A tarefa é escalonada, executada e quando finalizada, o Coordenador recebe uma mensagem de tarefa completada. A seguir o Coordenador despacha o resultado de volta ao *broker* e envia uma mensagem solicitando a *desalocação dos recursos*.

O processo de comunicação discutido (Figura 4.1) propõe uma sistemática de gerenciamento envolvendo o *cloud broker*, logo, pressupõe que o mesmo atue como membro externo aos provedores de nuvem, se comportando como espécie de agenciador de serviços em benefício do cliente e de seus requisitos de QoS (LIU, F. *et al.*, 2012) (BUMPUS, 2013). Essa abordagem também é considerada no modelo de referência de computação em nuvem do NIST e em diversos outros trabalhos disponíveis na literatura (RAJ, 2011) (FAZIO *et al.*, 2011) (CHANDRASEKAR, 2014).

4.2 Modos de Operação de Cloud Broker

Os modos de operação para *cloud brokers* são aplicáveis a vários cenários de computação em nuvem para a definição do comportamento padrão do elemento mediador no escalonamento de tarefas para sistemas de nuvem (BARKER; VARGHESE; THAI, 2015).

A atividade de mediação de realizada por um *cloud broker*, conforme definido pelo modelo de referência em computação em nuvem do NIST, pode seguir as seguintes abordagens para implementação: intermediação, agregação ou arbitragem (BUMPUS, 2013).

O modo de operação de intermediação é caracterizado pela atividade do *cloud broker* melhorar capacidades específicas de serviços de nuvem de modo a fornecer serviços de valor agregado aos clientes (BUMPUS, 2013).

Na agregação, o *cloud broker* pode combinar e integrar serviços de nuvem implementando um ou mais novos serviços. Além disso, conforme especificado pelo modelo de referência do NIST, o *cloud broker* deve prover os dados e a integração de serviços garantindo a segurança da informação entre os clientes e os provedores (BUMPUS, 2013).

A atividade realizada no modo de operação de arbitragem é bastante similar com o que ocorre na atividade de agregação, porém, os serviços que são combinados ou integrados nunca são fixos. Dessa forma, na arbitragem o *cloud broker* tem a possibilidade de efetuar escolhas flexíveis entre serviços de vários provedores de nuvem conforme o interesse dos requisitos solicitados pelos clientes (BUMPUS, 2013).

4.3 Trabalhos relacionados com *Cloud Brokers*

Na literatura é possível verificar um considerável número de trabalhos cujo enfoque principal está na proposição de novas arquiteturas para *cloud brokers*, fato que evidencia que a atividade de *brokerage* está ganhando destaque e encontra-se em fase de emancipação no campo de pesquisa de computação em nuvem (BARKER; VARGHESE; THAI, 2015).

Várias propostas de implementação de *cloud brokers* com abordagens distintas podem ser verificadas na literatura (AMATO; DI MARTINO; VENTICINQUE, 2013) (SAMREEN; BLAIR; ROWE, 2014) (KHANNA; JAIN, 2015). É importante citar que não há uma classificação para os trabalhos envolvendo *cloud brokers*, porém, conforme a abordagem de cada trabalho, é possível averiguar com qual tipo de elemento de nuvem o *broker* trabalha (por exemplo, manipulação de tarefas, serviços ou recursos) e qual o seu modo de operação (intermediação, agregação ou arbitragem).

Dado o crescimento no número de trabalhos sobre *cloud brokers* na comunidade científica, é possível afirmar que esses componentes intermediadores estão sendo vistos pelos pesquisadores e profissionais de sistemas de nuvem como facilitadores do acesso aos serviços para os clientes (FINGAR, 2012) (CHANDRASEKAR, 2014). O que se espera é que com essa crescente tendência, os clientes vinculem-se cada vez mais aos *brokers* e, desse modo, optem por utilizar serviços de nuvem por meio de intermediadores.

Dada a atualidade do assunto *cloud brokers* no cenário de pesquisa em computação em nuvem, dentre as publicações recentes encontram-se trabalhos de revisão literária sobre *cloud brokers*, os quais apresentam os conceitos básicos, vantagens e desvantagens, modos de operação e a importância dos *brokers* no auxílio para que os clientes encontrem mais e melhores serviços de nuvem (CHANDRASEKAR, 2014) (FANIYI; BAHSOON, 2015).

O uso de *brokers* como responsáveis pelo gerenciamento de eficiente de sistemas de nuvem é a abordagem de um trabalho no qual um gerenciamento da comunicação de infraestrutura eficiente, o qual leva em conta o custo e o tempo de execução para a entrega de serviços, é proposto (HWANG; KIM, 2012). Para tanto, os pesquisadores implementaram um módulo de gerenciamento de comunicação que foi vinculado ao *cloud broker*. Neste caso, o intermediador atua buscando uma conexão com um custo mínimo entre os clientes e provedores. Os resultados experimentais obtidos apresentaram melhoras nos valores obtidos pelos clientes entre custo e desempenho dos serviços solicitados.

Por meio da manipulação de requisitos de QoS e várias camadas de abstração em um ambiente de nuvem, pesquisadores implementaram uma arquitetura de *cloud broker* de recursos, o qual atua paralelamente em modo de operação de intermediação, agregação ou arbitragem de acordo com as políticas e estratégias de múltiplos provedores de nuvem (FAZIO *et al.*, 2011). Para a colaboração na execução de composição de tarefas entre provedores, os autores usam como estratégia o emprego de memória transacional para manter um *buffer* de informações temporárias até que o resultado final das operações de computação de um grupo de tarefas seja alcançado. Neste trabalho os autores afirmam ter alcançado resultados que representam melhorias no atendimento dos elementos de QoS solicitados por meio da arquitetura construída e das estratégias implementadas.

Numa abordagem diferenciada em relação à manipulação de contratos SLA, um trabalho que propõe uma arquitetura de *cloud broker* (GENEZ; BITTENCOURT; MADEIRA, 2012), a qual considera contratos SLA em 2 níveis, empregou a técnica de programação linear inteira (do inglês, *Integer Linear Programming* ou ILP) para abordar o problema de escalonamento de fluxo de trabalho de clientes em nível SaaS e PaaS. Nesse contexto de SLA de vários níveis, o primeiro nível de SLA é o do cliente e o segundo nível de SLA é o do provedor. Conforme os resultados experimentais, os autores afirmam, conforme a ILP criada com base em duas heurísticas propostas, que foi possível obter soluções de baixo custo para requisições de cliente com *deadlines* curtos (GENEZ; BITTENCOURT; MADEIRA, 2012).

A divisão de tarefas entre duas ou mais nuvens é também uma temática relacionada com os trabalhos envolvendo *cloud brokers*. Para resolver o problema desse tipo de cenário de pesquisa, autores propuseram uma arquitetura de *cloud broker* que provê suporte para a divisão exata de tarefas entre duas nuvens. A colaboração ocorre por meio de tecnologias de auxiliam a manipulação do fluxo de dados das aplicações e na comunicação entre as nuvens. Os resultados obtidos nos experimentos realizados demonstram que é possível obter resultados profícuos da divisão de tarefas entre duas ou mais infraestruturas de nuvem por meio de técnicas de manipulação do fluxo de dados e de comunicação (WU, D. *et al.*, 2013).

Implementações do processo de intermediação em nuvens computacionais considerando o uso de um *broker* com informações de contexto para auxílio na mediação entre provedores e clientes foram evidenciadas entre os trabalhos relacionados com *cloud brokers* (EVOY; SCHULZE; GARCIA, 2011) (LALEH *et al.*, 2015). A questão do que se considera como informação de contexto é um elemento particular de cada trabalho. Em um dos trabalhos

relacionados com essa temática, o estado de cada nó do sistema é a informação de contexto (EVOY; SCHULZE; GARCIA, 2011). O uso da estratégia de *caching* de dados de contexto, a qual armazena temporariamente informações de contexto dinâmicas, é uma abordagem implementada, por meio da qual autores revelaram ganhos em desempenho no ambiente computacional de nuvem segundo a estratégia de intermediação desenvolvida nos cenários experimentais considerados (KIANI *et al.*, 2011) (MALLICK; KUSHWAHA, 2013).

Uma arquitetura de *cloud broker* para a seleção de um provedor de nuvem a partir de múltiplas instâncias de provedores foi implementada por um trabalho (ACHAR; THILAGAM, 2014), no qual o *cloud broker* projetado mensura elementos de QoS de cada provedor e ordena-os conforme os requisitos da requisição do cliente. Para a diferenciação dos provedores de nuvem foi desenvolvido o *Service Measurement Index* (SMI), um índice relativo calculado para prover informação de diferenciação entre os serviços dos diferentes provedores. A técnica de seleção adequada de provedor denominada TOPSIS (*Technique for Order Preference by Similarity to Ideal Solution*) é baseada no estabelecimento de um *ranking* para a seleção de um provedor de nuvem adequado. Os resultados experimentais do trabalho foram obtidos a partir de experimentação em simulação com CloudSim. Foi realizado um conjunto de experimentos considerando 6 provedores e a conclusão dos autores foi a de que a aplicação do conjunto de técnicas escolhidas permitiu selecionar de modo eficiente os provedores de nuvem baseado nos requisitos de clientes.

A seleção de serviço em ambientes de nuvem com múltiplos provedores (NITHYA; JAYAPRATHA, 2014) é também a abordagem de autores que propõem uma solução por meio de uma arquitetura de *cloud broker* com múltiplos agentes. No referido trabalho, o algoritmo *Jumper Firefly* foi usado na implementação para reduzir o tempo de execução *make span time* (tempo de resposta), por meio de uma tabela de *status* que grava os comportamentos anteriores. A validação de todas as proposições feitas no trabalho foi realizada com auxílio do simulador CloudSim. Nos resultados experimentais e, segundo afirmações dos autores, foi possível constatar que o mecanismo *Jumper Firefly* é mais efetivo do que o algoritmo padrão *Firefly* e outras heurísticas que foram avaliadas.

Com objetivo de obter a execução rápida e efetiva de tarefas enviadas por usuários para um ambiente de *Cloud Computing* (RAJ; SETIA, 2012), autores de um uma arquitetura de *cloud broker* propõem um framework de comunicação entre o *broker* e as máquinas virtuais (VMs), buscando resultados de custo e execução ótimos. A referida arquitetura foi nomeada *Broker Virtual Machine Communication Framework* (BVCF). O ambiente de experimentação

foi construído com auxílio do simulador CloudSim e sua API, criando políticas de escalonamento de VMs baseadas em custo. No contexto da programação do ambiente simulado também foram considerados escalonamento de *cloudlets* e retransmissão de *cloudlets*, sendo que a análise da execução das tarefas foi realizada por meio das políticas Round Robin e FCFS (*First Come First Serve*). Conforme os resultados obtidos na experimentação e análise realizada pelos autores do trabalho, os fatores de custo e tempo de execução de uma tarefa são sempre os componentes primordiais entre as restrições de qualidade de serviço impostas por requisições de clientes.

Outro foco de pesquisas está na implementação de ambientes de federações de nuvens e *inter-clouds* (AAZAM, 2014), nesses cenários os serviços de valor agregado são valorizados pelos clientes e, dentre os elementos considerados como primordiais no gerenciamento, estão a política de precificação ou bilhetagem e a estratégia de alocação de recursos. Autores de um trabalho com o referido foco desenvolveram, com auxílio do CloudSim *Toolkit* versão 3.0.3, uma arquitetura de *cloud broker* para Federações de Nuvem, o qual atua com o processo de intermediação, interoperabilidade e negociação das requisições de serviço. Segundo os autores, conforme os resultados experimentais, pode-se concluir que o modelo de alocação de recursos baseado em QoS e sistemática reembolso na cobrança pelos serviços obteve êxito e demonstrou a aplicabilidade e necessidade da observação da degradação de QoS em ambientes complexos de *inter-cloud*.

Atender a cenários de simulação complexos com contextos de *inter-clouds*, conforme citado anteriormente, não é tarefa trivial e exige novas estratégias para o escalonamento de tarefas. Com foco nestes cenários, pesquisadores implementaram um novo modelo de escalonamento para ambientes de computação em nuvem denominado ICMS (*Inter-Cloud Meta-Scheduling*) (SOTIRIADIS; BESSIS; ANTONOPOULOS, 2013b). Para tanto, os autores da pesquisa criaram uma extensão para o CloudSim que foi nomeada de SimIC (*Inter-Cloud*). Na arquitetura proposta, o processo de intermediação de requisições (*cloudlets*) é feito por múltiplos *cloud meta-brokers* que executam o gerenciamento dinâmico e de tempo real das cargas de trabalho recebidas usando um processo decisório padrão para os escalonamentos efetuados. As métricas usadas para as análises foram tempo de execução e RTT (*Round Trip Time*) e como fatores de modificação dos ambientes simulados foram utilizadas diferentes submissões de usuários e requisitos computacionais. A partir dos resultados experimentais comparativos entre os valores retornados para o CloudSim *Toolkit* original e o SimIC, foi

possível verificar e concluir que houve ganhos consideráveis nos novos algoritmos implementados pelo módulo ICMS, principalmente nos resultados comparativos da métrica de tempo de execução.

4.4 Considerações finais

Neste capítulo foram apresentados conceitos fundamentais sobre *brokers*, com enfoque direcionado para trabalhos de computação em nuvem. Uma explanação em perspectiva sobre como se dá a participação do *broker* no ambiente de computação em nuvem foi dada com base em um modelo básico de comunicação entre um *cloud broker* e os demais elementos componentes do ecossistema de nuvem.

O processo de negociação executado por *cloud brokers* segundo os modos básicos de operação e a interação do *broker* com os demais componentes existentes num ambiente de nuvem computacional foram discutidas conforme o diagrama de sequência da Figura 4.1.

Trabalhos cuja temática destaca a implementação de *cloud brokers* foram discutidos com objetivo de apresentar os tipos de pesquisas que vêm sendo realizadas com essa temática e a importância que os *cloud brokers* vêm ganhando no campo de pesquisa de computação em nuvem. Para facilitar a análise das propostas que foram apresentadas no presente capítulo, foi elaborada a Tabela 4-1 contendo o resumo de informações das principais implementações de *cloud broker* e, ao final da mesma, são citadas as características da arquitetura QBroker.

Tabela 4-1: Resumo das propostas de arquiteturas de Cloud Broker.

Autores	Solução ou Técnica Proposta	Atendimento Requisitos QoS	Suporte SOA	Diferenciação. Serviços	Alocação. Din. Serviços	Ambiente de Nuvem Considerado	Modo Operação / Algoritmo Escalonamento
(RAJ, 2011)	<i>Efficient Broker Cloud Management (BCM)</i>	V	X	X	X	Nuvem Privada	<i>Optimum Route Cost Finder (ORCF)</i>
(FAZIO <i>et al.</i> , 2011)	<i>Cloud@Home</i>	V	X	V	X	Múltiplos Provedores	<i>Resource & QoS Manager Algorithm</i>
(CLAMP; CARTLIDGE, 2013)	<i>autonomous adaptive thresholding (AAT)</i>	V	X	V	X	Não informado	<i>R&C'S Brokerage Algorithm</i>
(AMATO; DI MARTINO; VENTICINQUE, 2013)	<i>Distributed Cloud Broker Architecture</i>	V	V	V	X	Múltiplos Provedores	<i>CFP Brokering</i>
(SAMREEN; BLAIR; ROWE, 2014)	<i>Adaptive Decision Making Broker (ADMB)</i>	V	V	V	X	Múltiplos Provedores	<i>Machine Learning for Optimised Decision Making</i>
(NITHYA; JAYAPRATHA, 2014)	<i>Multi Agent Brokering Approach</i>	V	X	X	X	Múltiplos Provedores	<i>Jumper FireFly Algorithm</i>
(HWANG; KIM, 2012)	<i>List and First-Fit / Deadline-aware Tasks Packing</i>	X	X	X	X	Múltiplos Provedores	<i>LFF-based VM Provisioning / DTP-based VM Provisioning</i>
(PARK; AN; YEOM, 2015)	<i>Virtual Cloud Bank (VCB)</i>	V	V	V	X	Múltiplos Provedores	<i>VCB Operation Algorithm</i>
(AAZAM, 2014)	<i>Resource Reservation and Refunding</i>	V	V	V	X	Federações de Nuvem	<i>Resource reservation and refunding algorithm</i>
(SOTIRIADIS; ANTONOPOULOS, 2013b), BESSIS;	<i>InterCloud Meta-Scheduling (ICMS) Framework</i>	V	X	V	X	Múltiplos Provedores	<i>ICMS Algorithm Set</i>
(ACHAR; THILAGAM, 2014)	<i>Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)</i>	V	X	V	X	Múltiplos Provedores	TOPSIS
(PARDO & SANTANA, 2016)	Arquitetura QBroker	V	V	V	V	Privada Pública Híbrida	Intermediação de Serviços com e sem QoS

É possível verificar, a partir da análise dos dados que constam da Tabela 4-1, que quase todos os projetos de arquiteturas de *Cloud Brokers* possuem tratativas de QoS com um enfoque para um requisito em específico ou com enfoque multiobjetivo. Os trabalhos mapeados que consideram explicitamente o ambiente baseado em SOA são mais escassos, embora quase todos considerem conjuntos de serviços de nuvem. A alocação dinâmica de serviços é uma característica mais rara, presente em apenas um item da tabela referente ao próprio QBroker. Quanto aos tipos de ambiente de nuvem considerados, na maioria dos trabalhos correlatos prevalece a opção de múltiplos provedores. No quesito modos de operação ou algoritmos de escalonamento é onde está a maior diversidade, pois, os trabalhos se diferem bastante nos algoritmos, arquiteturas, *frameworks* e demais soluções propostas como implementação.

Com relação a todos os itens da avaliação que constam da Tabela 4-1, pode-se observar que a arquitetura QBroker se destaca como a mais completa nos quesitos considerados. Desta forma, cabe citar que a arquitetura QBroker pode ser usada para um número de cenários maior. Essa caracterização dos principais detalhes inerentes ao projeto da arquitetura QBroker como componente de intermediação e QoS, auxiliam na percepção dos valores particulares que fazem desta arquitetura uma instância única no segmento de pesquisas relacionadas com escalonamento de tarefas e atividade de *brokerage*.

Concluindo, este capítulo sobre *cloud brokers* teve o objetivo de prover uma visão geral desse importante componente de ambientes de computação em nuvem.

O próximo capítulo apresenta a arquitetura CloudSim-BEQoS, a qual está relacionada com a arquitetura de *cloud broker* criada especificamente para esta tese. Os componentes da arquitetura CloudSim-BEQoS são apresentados e discutidos. Além disso, uma explicação detalhada da arquitetura QBroker é provida juntamente com sua interação com os demais componentes da arquitetura de nuvem CloudSim-BEQoS enfatizando, desta forma, as possibilidades de análises e validações possíveis para ambientes de computação em nuvem híbrida.

5. Arquitetura CloudSim-BEQoS e QBroker

5.1 Considerações iniciais

Neste capítulo, será apresentada a arquitetura de computação em nuvem denominada CloudSim-BEQoS, desenvolvida no LaSDPC, laboratório vinculado ao ICMC-USP de São Carlos. Tal arquitetura, modela um ambiente de computacional em nuvem no qual foram propostas muitas resoluções para desafios atuais dessa área de pesquisa. Além disso, será também apresentada e discutida a implementação do QBroker, um *cloud broker* que é um dos componentes da arquitetura CloudSim-BEQoS e cujas funcionalidades foram alvo do projeto de pesquisa vinculado a esta tese.

A sigla BEQoS (*Bursty, Energy and Quality of Service*) possui três conceitos embutidos, os quais se referem a cada um dos projetos de doutorado participantes da construção da arquitetura.

O primeiro conceito (*Bursty*) está atrelado a tese de doutorado defendida por Adriana Molina Centurion (CENTURION, 2015), a qual teve foco em avaliar o impacto das rajadas em serviços de sistemas de nuvem.

O segundo conceito (*Energy*) está relacionado com o projeto de doutorado de Paulo Sérgio Franco Eustáquio, o qual possui foco no consumo eficiente de energia com diferenciação de serviços.

O terceiro conceito presente na sigla (*Quality of Service*) faz referência ao projeto de pesquisa desta tese, no qual o foco está no provimento de uma arquitetura de *Cloud Broker* com modo de operação de intermediação para o provimento de qualidade de serviços no ambiente de nuvem computacional. Os três projetos tinham o fator comum de considerarem o ambiente de computação em nuvem híbrida para a execução de seus testes, análises e validações. A caracterização de cargas de trabalho adotada na classe Cliente foi implementada para uso comum, pois, a presença da camada cliente do ambiente de nuvem era requisito de todos os três projetos de doutorado.

É importante citar que a arquitetura CloudSim-BEQoS também foi usada como plataforma para o projeto de doutorado defendido por Bruno Guazzelli Batista (BATISTA, 2015), o qual se dedicou à investigação de modelos de negócio para sistema de computação em nuvem baseados em parâmetros de QoS relacionados a desempenho e segurança.

5.2 Arquitetura CloudSim-BEQoS

A arquitetura CloudSim-BEQoS é uma extensão criada a partir dos componentes pré-existentes da API do *CloudSim Simulation Toolkit* (BELOGLAZOV *et al.*, 2016), o qual trata-se de um projeto desenvolvido e mantido pelos pesquisadores do *CLOUDS Laboratory* vinculado a Universidade de Melbourne, Austrália, iniciado em meados de 2009.

A nova arquitetura CloudSim-BEQoS possui um conjunto de novas classes que foram criadas a partir da extensão das classes originais do simulador por meio do conceito de herança da linguagem orientada a objetos Java, linguagem na qual o simulador CloudSim foi construído.

É importante esclarecer que a CloudSim-BEQoS foi concebida a partir do trabalho em equipe de três projetos de pesquisa de doutorado realizados por alunos do grupo de pesquisa SDPC (Sistemas Distribuídos e Programação Concorrente) do ICMC-USP de São Carlos, sendo que cada um dos projetos contribuiu com uma parte da implementação realizada para a construção dos componentes da arquitetura.

5.3 Camadas da Arquitetura CloudSim-BEQoS

A arquitetura CloudSim-BEQoS foi idealizada, conforme já discutido, para possibilitar a simulação de ambientes de computação em nuvem com maior riqueza de detalhes. Quando de sua concepção, a arquitetura foi idealizada como um conjunto de camadas. As camadas consideradas na arquitetura BEQoS, conforme é possível verificar na Figura 5.1, foram: Camada Cliente, Camada de Gerenciamento e Camada de Infraestrutura.

A *Camada Cliente* é representada por entidades Cliente, isto é, por instâncias da classe *Client*, as quais simulam o comportamento de usuário de nuvem por meio do comportamento de modos de submissão de requisições (vetor de requisições, tempo real e grupo de requisições) distintos e de caracterizações de cargas de trabalho (caracterizadas pelos processos de chegada e demanda de serviço).

A *Camada de Gerenciamento* é composta por uma instância da entidade QBroker, a qual representa o componente de arquitetura *Cloud Broker* e efetua a atividade de mediação de requisições (por meio dos modos de operação de intermediação, agregação e arbitragem) e

negociação entre clientes e provedores de nuvem. Nessa camada encontram-se ainda os componentes da arquitetura referente aos monitores, os quais mantêm o *cloud broker* abastecido sobre informações de estado a respeito de rajadas, consumo de energia, consumo de recursos e o estado de filas das VMs.

A *Camada de Infraestrutura* é composta por instâncias de *Datacenters*, os quais são constituídos por conjuntos de equipamentos (*Hosts*), camada de virtualização (instâncias de VMs) e conjuntos de serviços (instâncias de serviços sobre as VMs). A composição dessa camada, apesar de baseada na API original do CloudSim, foi adaptada para prover um ambiente baseado em SOA, permitindo a instanciação e diferenciação de serviços.

Como é possível perceber, conforme a visão em camadas apresentada pela Figura 5.1, as camadas da arquitetura possuem fluxos (conforme é possível verificar pelas setas existentes no diagrama) de comunicação entre si. Essa comunicação de informações se dá por meio do envio de mensagens associadas aos eventos de simulação, conforme estrutura padronizada pelo simulador CloudSim.

Figura 5.1: Visão em Camadas da Arquitetura CloudSim-BEQoS. (Autoria Própria)

Para demonstrar como é a sequência de comunicação envolvendo todas as três camadas, na Figura 5.2 é apresentado um diagrama de sequência contendo as principais entidades de simulação instanciadas em tempo de simulação e a sequência de mensagens trocadas para representar o envio de uma requisição e o recebimento de sua resposta.

De acordo com o diagrama de sequência da Figura 5.2 é possível verificar a ordem numerada das mensagens trocadas entre a camada de clientes virtuais, representada pelo objeto *Cliente* no diagrama e os demais componentes da Arquitetura CloudSim-BEQoS. A sequência de mensagens prevê as seguintes operações:

1. Inicialmente o *Cliente* dispara a mensagem *envia requisição*, a qual é recebida pelo *cloud broker* da arquitetura, representado pelo objeto *QBroker*.
2. O *QBroker*, por sua vez, ao receber uma requisição de serviço, envia a mensagem *notifica chegada de requisição* ao objeto *Monitor de Requisições*, o qual registra a requisição, atualiza a contagem de requisições recebidas e devolve uma mensagem de *status de sucesso*.
3. *QBroker* envia ao objeto *Monitor de Recursos* uma outra mensagem de *consulta a carga atual do sistema*.
4. Uma mensagem de consulta ao objeto *Datacenter* com intuito de *verificar a porcentagem de utilização de VMs privadas* é disparada, resultando numa mensagem de retorno com a informação solicitada. Em seguida essa informação é repassada do *Monitor de Recursos* de volta ao *QBroker* o qual irá prosseguir com seu processo de mediação.
5. Uma vez recebida a informação da porcentagem de uso das VMs privadas, o *broker* envia uma nova mensagem ao *Monitor de Energia*.
6. O *Monitor de Energia* realiza nova consulta a entidade *Datacenter*, solicitando informação do consumo energético dos *datacenters* em *Watts*. A resposta da solicitação do *Monitor de Energia* é devolvida e repassada ao *broker*.
7. Enfim, de posse de todas as informações de estado do sistema de nuvem as quais precisa para efetuar o processo de negociação, conforme as configurações do ambiente simulado, o *broker* inicia a comunicação de mensagens com o *MetaCloudletScheduler* afim de executar a regra de negócio de mediação conforme o modo de operação que estiver configurado no mesmo. Dessa forma, o *broker* envia uma mensagem ao

MetaCloudletScheduler *solicitando a seleção de VMs conforme o ID de serviço* que foi solicitado na requisição. Quando está de posse dessa lista (a qual pode conter itens ou não) o *QBroker* procede com a verificação dos parâmetros de QoS frente aos recursos disponíveis. Se for necessário e houver possibilidade, o *broker* pode alocar novos serviços em VMs ativas ou em *Standby*. Caso não exista condições de atender ao pedido do cliente, o *broker* envia uma resposta ao cliente com o *status* da tarefa (*cloudlet*) configurado como não atendida (NOT_SCHEDULED).

8. Após todas as verificações, se o *broker* encontra recursos em condições de atender a requisição de serviço do cliente cumprindo com os requisitos de QoS solicitados, uma mensagem de *submissão e alocação de tarefa* é disparada contra o componente *Datacenter*, o qual deverá recepcionar a tarefa e executá-la na VM para a qual o *broker* determinou sua execução. Depois de algum tempo de execução, a resposta da tarefa pronta é devolvida ao *broker* que por sua vez a repassa ao cliente virtual em questão. O cliente virtual, por sua vez, recepciona a resposta esperada por sua aplicação, espera um *quantum* de tempo referente ao *think time*.
9. Após o *think time*, o cliente virtual envia uma nova requisição, reiniciando novo procedimento conforme já discutido.

É importante destacar, para o caso da comunicação de mensagens entre as entidades e objetos de simulação presentes na Figura 5.2, que nem sempre todos os componentes apresentados estão instanciados no ambiente de simulação. Alguns componentes opcionais são usados sob demanda em determinados cenários de simulação, como é o caso do *Monitor de Energia* que é empregado apenas nos experimentos de simulação que consideram consumo energético como parâmetro de QoS para o processo de negociação entre cliente e provedores.

Figura 5.2: Diagrama de sequência com comunicação entre objetos da arquitetura CloudSim-BEQoS. (Autoria Própria)

A função do *QBroker* como agente de intermediação e QoS para os ambientes de nuvem simulados por meio da Arquitetura CloudSim-BEQoS já foi apresentada conceitualmente nos capítulos conceituais anteriores de Computação em Nuvem e *Cloud Brokers*, porém, para prover uma explicação mais detalhada de sua funcionalidade, modo de operação e relação do mesmo com os demais componentes do sistema, na próxima seção será apresentada a Arquitetura QBroker.

5.4 Características Arquiteturais

Para o melhor entendimento de como a arquitetura CloudSim-BEQoS foi concebida, esta seção apresenta uma discussão sobre as características chave da arquitetura, os novos componentes desenvolvidos para o ambiente de simulação, uma visão da arquitetura explanada em camadas e uma visão geral de como se dá a comunicação entre os clientes virtuais e o ambiente de nuvem por meio de um diagrama de sequência UML (*Unified Modeling Language*).

Para cumprir com todos os requisitos para os quais a arquitetura CloudSim-BEQoS foi idealizada, uma série de características fundamentais fazem parte do projeto do ambiente de simulação. Essas características são as seguintes:

- Representar o lado cliente do ecossistema de computação em nuvem e implementar nesse componente os principais elementos que representem o seu comportamento do modo mais realístico. Deve ser possível, por exemplo, modelar diferentes tipos de cargas de trabalho a serem enviadas, por meio de modos de submissão distintos, modelar a sistemática de *think times* para tornar o comportamento entre as submissões mais real. Caracterizar a ocorrência de rajadas no processo de chegada das requisições ou na demanda de serviço também deve ser possível;
- Prover um método para a mediação entre as entidades cliente e os provedores de nuvem, representado por uma entidade que seja um *Cloud Broker*. Essa entidade deve fornecer qualidade de serviço ao ambiente de computação, por meio de modos de operação que possuam estratégias de escalonamento de tarefas que considerem serviços remotos (caracterizando um ambiente SOA) alocados em diferentes provedores de modo a possibilitar a gestão de serviços de nuvem em ambientes mais complexos, usando como cenário experimental padrão o modelo de implantação de nuvem híbrida;

- Oferecer recursos para tratativas de consumo eficiente de energia, por meio da implementação de uma política de escalonamento de tarefas para sistemas de nuvem, a qual considere um ambiente no qual exista a possibilidade da diferenciação de serviços e tarefas considerando a prioridade como parâmetro de QoS;
- Efetuar uma avaliação de desempenho da arquitetura com base em uma metodologia de avaliação de sistemas computacionais já reconhecida pela academia (JAIN, R., 1991).

Para o cumprimento dos requisitos supracitados, foram desenvolvidas várias classes de objetos, as quais representam novos componentes e funcionalidades do ambiente de simulação computacional de nuvem baseado no CloudSim. Os códigos de programa implementados possibilitaram que os requisitos traçados para a arquitetura BEQoS fossem alcançados.

Na implementação do projeto, desenvolvido em linguagem Java, para a arquitetura CloudSim-BEQoS, foram desenvolvidas as seguintes novas classes de objeto: *Client*, *QBroker*, *QBrokerPower*, *MetaCloudletScheduler*, *PowerMonitor*, *ResourceMonitor*, *Utilities*, *VMConfiguration*, *DatacenterConfiguration*, *ServiceDemandTable*, *Parameters*, *CloudletSchedulerSpaceSharedUSP*, *PowerModelSpecPowerIntelXeon2670*, *PowerModelSpecPowerIntelXeon5650*, *MAPIAT*, *MAPIATSinc* e *MAPSD*. O total de linhas de código implementadas somando-se todas as classes de objeto citadas é de 8437.

As classes de objeto implementadas na arquitetura BEQoS são apresentadas, juntamente com seu relacionamento com classes da API do simulador CloudSim, no diagrama que consta da Figura 5.3.

É importante destacar que no diagrama da Figura 5.3 não constam todas as classes de objeto que compõem a API do CloudSim *Simulation Toolkit*, pois, o objetivo foi apresentar as novas classes da arquitetura BEQoS e suas associações diretas (ou relacionamentos) com classes da API do simulador.

É possível perceber (conforme Figura 5.3) que o diagrama de classes UML utiliza caixas retangulares para representar as classes. No diagrama resumido apresentam-se apenas os nomes das classes e seus campos de instância e métodos são ocultados.

Figura 5.3: Diagrama de classes com componentes da arquitetura CloudSim-BEQoS e associações com componentes da API padrão do CloudSim. (Autoria própria)

Há também uma diferenciação de cores utilizada para a identificação e separação das novas classes de objeto projetadas (destacadas pela cor verde), das classes de objeto da API do simulador que foram alteradas (destacadas em cor azul) e das classes de objeto originais da API do CloudSim (destacadas em cor amarela) que estão relacionadas com componentes da arquitetura BEQoS.

5.4.1 Detalhamento das classes de objeto da arquitetura CloudSim-BEQoS

Dando prosseguimento a discussão sobre a implementação, nesta seção serão explanadas com mais detalhes as classes de objeto da arquitetura CloudSim-BEQoS.

Por meio da análise do diagrama de classes da Figura 5.3, é possível notar a classe *SimEntity*, pertencente a API original do CloudSim no topo da hierarquia de classes de entidade. Essa classe é a base para a construção de entidades de simulação. A classe *DatacenterBroker* (também parte do *Toolkit* original do CloudSim), por exemplo, tem uma relação de herança para com a classe *SimEntity*. A classe *DatacenterBroker* representa a entidade *Cloud Broker* no ambiente de simulação CloudSim. Além disso, a classe *DatacenterBroker* provê um

mecanismo de intermediação básico, o qual consiste num algoritmo Round Robin para efetuar a distribuição de tarefas para as VMs dos *datacenters* de nuvem.

A classe *QBroker* tem relação de herança com a classe *DatacenterBroker* e possui todos os métodos referentes aos eventos de simulação para realizar a tarefa de intermediação, provendo as assinaturas dos métodos referentes aos modos básicos de operação (intermediação, agregação e arbitragem). Os métodos foram projetados de modo que o programador usuário da classe possa realizar o *overloading* (sobrecarga) dos métodos caso seja necessário customizar o código de programa para suas necessidades. É importante destacar a relação da classe *QBroker* com a classe *MetaCloudletScheduler*, pois, todos os métodos que representam algum tipo de ação para realizar a política de escalonamento, tais como selecionar VMs que possuem um dado serviço, verificar o tamanho de fila de uma VM, obter a VM com maior fila, menor fila ou fila média, entre outros, são implementados na classe *MetaCloudletScheduler*, a qual serve como apoio ao *QBroker* e permite a característica de abertura (*Openess*) para a arquitetura *QBroker*, pois seus métodos podem também ser sobrescritos ou novos métodos podem ser adicionados. É importante destacar sobre a classe *MetaCloudletScheduler*, que a mesma possui acesso direto a lista de VMs ou PowerVMs (especialização da classe VM para simulações que consideram consumo de energia) instanciadas durante o tempo de simulação (conforme relacionamento apresentado na Figura 5.3). Consequentemente, essa classe também possui acesso aos *CloudletSchedulers* (componentes que possuem a política de escalonamento e execução de tarefas interna de cada VM) associados a cada VM ou aos *Hosts* onde cada VM está hospedada. Por meio desses acessos exclusivos, é possível implementar na classe *MetaCloudletScheduler* métodos que permitam a obtenção de informação de estado em qualquer nível hierárquico da infraestrutura de recursos dos provedores de nuvem de um cenário.

A classe *QBrokerPower* tem relação de herança com a classe *QBroker* e possui código de programa incremental para o tratamento de requisições conforme classes de prioridade, visando escalonamento de tarefas para consumo eficiente de energia.

A classe de objetos *Client* possui relação de herança com a classe *SimEntity* e detém os métodos de eventos para realizar as funcionalidades de envio e recebimento de requisições de serviço para o ambiente de nuvem. Além disso, a classe *Client* também possui métodos referentes a representação do *think time*, o intervalo de tempo existente entre o recebimento de uma resposta pelo cliente e a submissão de uma nova requisição. Métodos que executam a

funcionalidade de geração de demanda (tamanho da tarefa) e geração dos identificadores de serviço a serem solicitados na nuvem também são providos com essa entidade, a qual consegue representar várias das ações reais de um cliente enviando requisições durante o tempo de simulação. A entidade cliente conta ainda com três modos de submissão de requisições distintos:

- **Vetor de Requisições:** Neste modo de submissão, um vetor requisições (*cloudlets*) é gerado no início da simulação, contendo a caracterização completa da carga de trabalho. Dessa forma, a quantia de requisições enviadas pelos clientes virtuais é sempre fixa durante o tempo de simulação.
- **Tempo Real:** Trata-se de um modo de submissão no qual os clientes enviam requisições durante o tempo de simulação ou de modo *online*, porém, sem perspectiva de quantidade. As entidades referentes aos clientes virtuais ficam continuamente em *looping* no processo de envio e recebimento de requisições de serviço de nuvem. Nesse tipo de configuração para os clientes é necessário finalizar a simulação por meio de uma configuração de tempo de simulação pré-determinada.
- **Grupos de Requisições:** no modo de submissão de grupos de requisições, as entidades cliente são configuradas com um valor padrão que determina a quantidade de requisições por grupo e, a partir de então, são geradas as composições de requisições de serviço que são enviadas ao *cloud broker*, o qual receberá os grupos de requisições e efetuará as tratativas de mediação conforme o modo de operação determinado.

Ainda sobre a classe de objetos *Client*, é possível perceber, observando o diagrama de classes da Figura 5.3, que as classes *MAPIAT*, *MAPDS* e *MAPIATDSSINC* são utilizadas pela classe *Client* em seu código de programa. Essas classes que começam pelo nome MAP (do inglês, *Markovian Arrival Process*) relacionam-se com o processo de chegada Markoviano, o qual foi utilizado nos métodos dessas classes. O detalhamento do funcionamento do código de programa referente à essas classes é provido no trabalho de doutorado de CENTURION(2015).

No diagrama de classes da Figura 5.3 é possível perceber a existência de entidades referentes aos monitores existentes no ambiente de simulação da arquitetura CloudSim-BEQoS. Esses componentes monitores trabalham associados ao *QBroker* provendo informações de estado dos recursos dos provedores de nuvem, com objetivo de auxiliar o processo de mediação

de tarefas. Todas as classes de objeto que representam os monitores são estendidas da classe *SimEntity*, portanto, tais componentes apresentam-se ao ambiente de simulação como entidades. Uma das classes referentes aos monitores é nomeada de *WorkloadMonitor* e que efetua o monitoramento da chegada de requisições ao *cloud broker* e também monitoram os *slots* de tempo em que se verifica a taxa de chegada de requisições. A classe *ResourceMonitor* implementa o monitoramento dos recursos computacionais de um *datacenter* de nuvem privada e por meio da coleta informações de estado calcula e provê o valor do indicador de desempenho de porcentagem de uso de recursos (aferido em % de MIPS utilizados das VMs). A classe *PowerMonitor* possui a funcionalidade de monitoramento da porcentagem de uso de energia (verificado em unidade de *watts*) dos recursos de um *datacenter*. Esse indicador de desempenho é importante para ambientes simulados nos quais o objetivo seja a implementação de políticas de escalonamento de tarefas considerando consumo eficiente de energia ou ainda outros aspectos de Computação Verde.

A classe *PowerModel*, pertencente a API original do CloudSim, é a superclasse de duas classes especializadas que foram criadas com os nomes de *PowerIntelXeon5650* e *PowerIntelXeon2670*, as quais modelam o comportamento de consumo energético dos processadores Intel Xeon modelos 5650¹ e 2670², conforme valores publicados no *website* da SPEC obtidos a partir de execuções do *benchmark* SPECPower_SSJ2008. Essas classes são utilizadas em simulações que utilizam os algoritmos de escalonamento de tarefas para consumo eficiente de energia.

A classe *CloudletScheduler*, conforme já mencionado, representa a atividade de escalonamento e execução de tarefas de cada uma das VMs. Toda VM criada no ambiente de simulação precisa ter uma instância de uma *CloudletScheduler*. A API original do CloudSim ainda oferece outras classes especializadas dessa classe que são:

- *CloudletSchedulerSpaceShared* – emula política de escalonamento de tarefas com base em execução de tarefas alocando a CPU como posse exclusiva para uma dada tarefa por vez até que a mesma chegue ao final de sua execução;

---

¹ Resultados do *benchmark* SPECPower_SSJ2008 disponíveis em: [\[LINK\]](#)

² Resultados do *benchmark* SPECPower_SSJ2008 disponíveis em: [\[LINK\]](#)

- *CloudletSchedulerTimeShared* – emula política de escalonamento de tarefas com base na técnica de tempo compartilhado (*time sharing*);
- *CloudletSchedulerDynamicWorkload* - emula política de escalonamento de tarefas com base na premissa de cargas de trabalho dinâmicas tratando cada tarefa (*cloudlet*) em execução como um serviço online.

Com base na implementação da classe *CloudletSchedulerSpaceShared* foi criada uma especialização da mesma nomeada como *CloudletSchedulerSpaceSharedUSP*, a qual implementa uma política de escalonamento de tarefas baseada no agendamento de um tempo de início da tarefa, conforme valores do relógio da simulação (*clock tick*), de modo a permitir o controle e garantia do momento quando a tarefa será iniciada e, por consequência, terminada. Esse recurso permite usar tratativas de “agendamento” de tarefas como parte do mecanismo de provimento de QoS.

A classe VM original da API do CloudSim foi modificada para a inclusão de campos de instância e métodos para acrescentar a abstração de serviços. Assim, uma coleção de identificadores de serviço (variável *services* do tipo *List*) foi adicionada como variável de classe. Os métodos *getters* e *setters* para acessibilidade a essa classe também foram confeccionados. Uma variável de classe chamada *numMaxServices* do tipo inteira (*int*) foi criada, juntamente com seus métodos *getter* e *setter* para permitir definir o número máximo de serviços que podem ser instanciados em cada VM criada. Essa implementação permite aproximar o ambiente de simulação das características observadas numa arquitetura orientada a serviços.

A classe *HostDynamicWorkload* original da API do CloudSim foi adaptada para as necessidades da arquitetura BEQoS por meio de ajustes que foram feitos no método *updateVmsProcessing()* e pela adição dos métodos *getUtilizationOfCpuSimulacaoUSP()* e *getPreviousUtilizationOfCpuSimulacaoUSP()* os quais permitiram a leitura do percentual de uso de recursos de CPU dos *Hosts* dos *datacenters*. Esse indicador é fundamental para algumas verificações que são realizadas pelo *MetaCloudletScheduler* em seus métodos que efetuam a verificação do estado atual da carga de trabalho executada pelos recursos de nuvem dos provedores considerados.

A classe *SANStorage* foi utilizada pela arquitetura BEQoS para representar as transações de leitura, escrita e exclusão de arquivos de dados em sistemas de armazenagem de arquivos. Já a classe *NetworkTopology* foi utilizada pela arquitetura BEQoS para a

representação das latências de comunicação de dados em rede entre as entidades cliente e *broker* e entre *broker* e provedores de nuvem.

Algumas classes utilitárias para evitar replicação excessiva de declarações de variáveis nos blocos de código de programas de simulação foram criadas com os nomes de *Utilities*, *Parameters*, *VMConfiguration*, *DatacenterConfiguration* e *ServiceDemandTable*. A classe *Utilities* possui uma série de métodos auxiliares para a criação de coleções de objetos como listas de VMs, de *Hosts* e *Datacenters*, além de muitos métodos de registros de dados em arquivos de saída (os quais são utilizados para análises após as execuções de experimentos de simulação) e métodos para a impressão em terminal de texto dos resultados da execução das tarefas no simulador. A classe *Parameters* possui uma série de variáveis estáticas referentes a componentes de configuração geral que são usados nos programas de simulação, tais como listas de VMs para nuvem privada e pública, listas de tarefas (*cloudlets*), vetores de configurações para as características de máquinas virtuais privadas e públicas, definição de valores padrão das configurações de *Hosts* para nuvem privada e pública e, por fim, vetores contendo modelos de consumo energético que são utilizados em simulações que consideram consumo eficiente de energia. A classe *VMConfiguration* possui variáveis de instância públicas que são utilizadas para hospedar os valores padrão de configurações de VM e passa-los ao método de construção de listas de VMs da classe *Utilities*. A classe *DatacenterConfiguration* possui variáveis de instância públicas que são utilizadas para armazenar os valores padrão de configurações de Host e passá-los ao método de construção de listas de *Hosts* da classe *Utilities*. A classe *ServiceDemandTable* trata-se de um objeto do tipo mapa (*Map*) o qual hospeda todos os identificadores (IDs) de serviço considerados no ambiente de simulação e um valor padrão utilizado para a geração de demanda em MI (sigla referente a unidade de Milhões de Instruções, utilizada pelo simulador CloudSim na quantificação de comprimento de tarefa) por meio do uso de uma função exponencial com média baseada nesses valores padrão.

5.5 Arquitetura do QBroker (QoS Broker)

A arquitetura de *cloud broker* proposta nesta tese, foi nomeada como QBroker (QoS Broker), e deve permitir uma melhor adequação na avaliação do atendimento dos requisitos de QoS dos clientes. Essa premissa é factível por meio do mapeamento dos serviços de nuvem associados com as VMs de cada provedor que participar do ambiente. No caso específico desta tese, considera-se uma nuvem privada e um provedor de nuvem pública, denotando ambiente

de nuvem híbrida. O *cloud broker*, ao efetuar o seu papel, irá efetuar consultas ao mapa de serviços e recursos, buscando pelos que atendem aos parâmetros solicitados para atender ao cliente.

A arquitetura QBroker é composta por uma camada de abstração que irá mapear a lista de serviços de nuvem instanciados a diferentes recursos virtuais (máquinas virtuais, por exemplo) de diferentes provedores de nuvem. Outro fator componente da proposta do *broker* é que o mesmo considera requisitos de QoS das requisições enviadas por clientes, de modo que a intermediação possa buscar a adequação dos requisitos informados nos recursos disponíveis buscando alcançar o desempenho esperado nos resultados finais obtidos.

O *broker* de serviços, como é proposto nesta tese, pode alcançar melhorias em desempenho ao lidar com serviços que podem se encontrar em distintos provedores de nuvem. Para o cenário a ser estudado na implementação da arquitetura, os serviços de nuvem serão expostos aos usuários e ao *broker* como abstrações de serviços Web, de modo a representar a convergência desses dois paradigmas (Nuvem+SOA).

Para desempenhar a sua função, um *cloud broker* deve escalonar as requisições atribuindo-as aos serviços ou recursos de nuvem. Para tanto, se faz necessário que o mesmo possua algoritmos de escalonamento que forneçam a sinergia entre os requisitos de QoS do cliente e os requisitos de QoS de provedor, executando as tarefas respeitando as restrições e, ao mesmo tempo, efetuando balanceamento de carga no sistema de nuvem.

O escalonamento de tarefas a ser executado na arquitetura QBroker pode manipular requisições de serviços individualmente e os algoritmos de escalonamento considerados são implementações dos modos de operação do modelo de referência de nuvem do NIST. Tendo em vista que o *broker* em questão deverá também conciliar a questão da estrutura privada e da estrutura pública de nuvem, os algoritmos de escalonamento consideram tratativas para a manipulação de QoS de modo que os requisitos de qualidade serviço deverão ser seguidos ou a tarefa deverá ser refutada.

Em arquiteturas de *broker* propostas em trabalhos anteriores, o serviço de mediação entre os clientes e provedores era com base nos recursos, isto é, nas máquinas virtuais (FAZIO *et al.*, 2011). Focando apenas em recursos de infraestrutura não é possível atingir uma granularidade de serviço mais alta, por exemplo, para provedores que possuem modelo de serviço PaaS ou SaaS. Um *broker* de recursos que apenas aloque máquinas virtuais capacitadas para um cliente pode não ser uma solução eficiente para a representação de provedores PaaS ou

SaaS. Assim, a combinação da abstração de serviços associada com recursos de máquinas virtuais, produz resultados mais significativos e próximos aos ambientes reais de computação em nuvem baseados em SOA.

5.5.1 Camadas da Arquitetura QBroker

O QBroker é um componente que atua na camada de gerenciamento de serviços da Arquitetura CloudSim-BEQoS. Todavia, o QBroker possui uma arquitetura de camadas e também segue as definições de modos de operação apresentadas na arquitetura de referência em computação em nuvem do NIST (BOHN *et al.*, 2011).

Um diagrama de componentes da arquitetura QBroker é apresentado na Figura 5.4. É possível verificar, na camada de aplicação, várias instâncias de clientes enviando requisições de tarefas para o *broker*, o qual recebe as requisições e faz as consultas ao repositório constituído pelo mapa de serviços e de recursos. Com a resposta do repositório o *broker* decide pelo cliente e escalona o serviço que melhor atender aos requisitos de QoS solicitados. Dependendo do provedor selecionado, a requisição será encaminhada para o sistema de nuvem privada ou pública. Ao realizar o escalonamento da tarefa para um recurso, o QBroker leva em consideração que aos atendimentos feitos na nuvem pública incorrem custos extras para a organização, enquanto que os atendimentos na nuvem privada, para o cenário considerado, não incorrem em custos. Dessa forma, é ideal que as requisições só sejam encaminhadas a nuvem pública em casos de saturação da capacidade da nuvem privada.

Os clientes (Camada de Aplicação da Figura 5.4) enviam as requisições a qualquer momento durante o tempo de simulação, isto é, o processo de chegada das requisições é não determinístico. Além disso, cada cliente pode solicitar diferentes serviços a cada requisição. Há também, por parte do cliente, o recurso de gerar tarefas com diferentes tamanhos para a demanda (propriedade *length* de cada tarefa ou *cloudlet*). O Cliente foi desenvolvido como sendo uma classe de objeto separada, de forma que a modelagem do cenário ficou diferente do padrão do CloudSim, contudo, a caracterização de componentes aproximou-se mais de um cenário real de computação em nuvem.

A arquitetura QBroker foi projetada para dar suporte aos três modos de operação que constam no modelo de referência do NIST. Na versão corrente da implementação, o modo de operação que foi implementado e que se encontra em estado funcional é o de intermediação. O algoritmo de intermediação consulta, inicialmente, o mapeamento de recursos, que contém a

relação de provedores e VMs, no qual consultando individualmente cada VM, pode-se obter a lista de serviços ativos instanciados em cada uma. A seguir, é efetuado o processo de negociação, isto é, a verificação dos requisitos de QoS impostos na requisição para, então, efetuar o despacho da requisição ao recurso ou a negação de serviço caso não exista possibilidade de cumprir os requisitos de QoS.

Quando as requisições de clientes chegam ao *broker* são colocadas em uma fila única por ordem de chegada. Essa ação é o padrão de funcionamento de *um Cloud Broker* e foi denominado de etapa de pré-escalonamento.

Figura 5.4: Arquitetura em Camadas do QBroker. (Autoria Própria)

Na segunda etapa, isto é, no processo de escalonamento principal, as requisições são negociadas conforme o modo de operação que estiver sido configurado no QBroker. Suponha-se que o modo de operação configurado seja o de intermediação, isso significa que o broker irá trabalhar com requisições individuais. Será feita a chamada ao método de intermediação, o qual realiza várias atividades sequenciais de negociação para tentar escalonar a tarefa (conforme já apresentado no diagrama de sequência da Figura 5.2) para um recurso (VM) que possua o serviço (no caso, consideram-se os serviços como abstrações de *Web Services*) requisitado.

Uma outra possibilidade para a segunda etapa seria o *broker* trabalhar com grupos de tarefas, isso significa que para cada conjunto de tarefas é realizada uma chamada ao método de escalonamento de grupos (agregação ou arbitragem) e então o *broker* realiza várias atividades de negociação na tentativa de que as requisições dentro do grupo sejam atribuídas para os recursos de nuvem que possuam os serviços solicitados em cada requisição. Caso o *broker* verifique que todas as requisições de um grupo de tarefas podem ser atendidas por um recurso do ambiente de nuvem, o grupo de tarefas é encaminhado aos recursos. Contudo, caso o *broker* verifique que uma das tarefas não poderá ser executada no grupo de tarefas, todo o grupo é rejeitado. É importante destacar também, no caso dos grupos de tarefas, que cada requisição poderá executar numa VM distinta, por conta da distribuição dos serviços. A fase na qual as tarefas são executadas e o resultado de seu processamento é enviado de volta ao *broker* para subsequente envio da resposta ao cliente é considerada etapa de pós-escalonamento.

Com objetivo de clarificar a sistemática de escalonamento representada pelo algoritmo de intermediação proposto neste trabalho, um diagrama de atividades UML foi desenvolvido e é apresentado na Figura 5.5. O algoritmo proposto considera algumas configurações possíveis, dentre elas é possível destacar: a utilização ou não de atributos de QoS, utilização de ambientes de nuvem estáticos ou dinâmicos e utilização de modelos de implantação de ambiente de nuvem, podendo ser privada, pública ou híbrida.

Tanto nos casos em que atributos de QoS são considerados quanto nos casos em que esses atributos não são considerados, deve ser definido um tempo máximo de execução para cada requisição a ser executada. O que diferencia os dois casos é que para execução com QoS, se não for encontrada uma VM que consiga executar a requisição de serviço no tempo estipulado, ela será descartada. No caso sem QoS, se nenhuma VM conseguir executar a requisição no tempo máximo de execução estipulado, ela será alocada na VM que possuir o serviço requisitado e com menor fila.

Nos ambientes de nuvem estáticos, todas as VMs que fazem parte do cenário têm os serviços alocados antes do início da simulação. Nos ambientes dinâmicos, parte das VMs podem ter os serviços alocados no início da simulação e as demais VMs terão os serviços alocados em tempo de simulação, quando o serviço for requisitado e não tiver disponível em uma VM atendendo ao requisito de tempo de serviço máximo (pode-se alocar a instância de serviço máquina virtual ativa em *standby*).

Os tipos de ambiente em nuvem que podem ser considerados são nuvem privada, pública ou híbrida. Cabe ressaltar que a modelagem da infraestrutura de nuvem pode ser definida conforme necessário, contendo quantas nuvens públicas ou privadas se queira e para cada uma das nuvens pode-se definir qual a configuração das máquinas físicas (*Hosts*) e virtuais.

Conforme o diagrama de atividades do algoritmo de intermediação do QBroker (Figura 5.5) é possível verificar já no ponto de início o algoritmo recebe como entrada uma requisição (*cloudlet*) com um identificador de serviço (*service id*) solicitado pelo cliente. A primeira atividade é solicitar ao *MetaCloudletScheduler* a seleção de todas as VMs disponíveis e ativas que possuem o serviço requisitado instanciado. A seleção mencionada é obtida como um objeto lista (*List*). Se a lista estiver vazia, isto é, se não existem VMs com o serviço requisitado, o fluxo de programa segue para uma sequência de atividades de tentativa de alocação de serviço. Se a lista estiver preenchida com instâncias de VM com o serviço solicitado, o fluxo de programa segue para uma sequência de atividades de escalonamento nas VMs que constam da lista.

No caso de a lista retornar preenchida, a primeira atividade executada é a tentativa de escalonar a tarefa para uma VM da nuvem privada com tempo de fila zero. Caso a primeira atividade não obtenha sucesso, a segunda atividade é executada na tentativa de escalonar a tarefa para uma VM da nuvem pública com tempo de fila zero. Se não houverem VMs com tempo de fila zero nem na nuvem privada e nem na nuvem pública, então a próxima atividade efetuará a tentativa de escalonar na VM com a menor fila conforme as VMs da lista retornada. Durante todas as atividades de escalonamento, é importante frisar que o algoritmo de intermediação sempre verifica se o requisito de QoS de tempo máximo de execução da tarefa é atendido.

Figura 5.5: Diagrama de Atividades do Algoritmo de Intermediação do QBroker. (Autoria Própria)

Se todas as atividades citadas anteriormente não puderem escalonar a tarefa para uma VM com o serviço requerido conforme suas restrições de QoS quanto ao tempo máximo de execução, a sequência do fluxo (seta em cor azul no fluxo do diagrama da Figura 5.5) será um conjunto de atividades que irão tentar alocar uma nova instância de serviço na infraestrutura disponível e atender ao parâmetro de QoS. A primeira atividade tentará efetuar a alocação de uma nova instância do serviço requisitado em uma VM ativa (isto é, que já possua outros serviços instanciados e esteja sendo utilizada) da nuvem privada. Se a primeira tentativa não obtiver sucesso no escalonamento, a segunda atividade tentará alocar uma nova instância do serviço em uma VM ativa da nuvem pública. Em seguida, no caso da impossibilidade de

escalonamento, será executada a atividade de tentar alocar a nova instância de serviço em VMs privadas em *standby* (VMs que não estão sendo utilizadas e que não possuem nenhum serviço instanciado) e, a seguir, se não for possível o escalonamento, será feita tentativa nas VMs públicas em *standby*. Caso todas as atividades mencionadas não consigam escalar com sucesso a tarefa para um recurso de nuvem (VM) com o serviço requerido cumprindo o parâmetro de QoS de tempo máximo de execução, então a tarefa será descartada.

É necessário mencionar que em cada requisição de serviço de nuvem (*cloudlet*) há um parâmetro booleano chamado *nonQoS* que permite que uma tarefa, ao invés de ser descartada, seja atribuída à VM que possuir o serviço requerido com a menor fila. Normalmente, em muitas situações, isso significa o descumprimento do parâmetro de QoS de tempo máximo de execução, dessa forma, essa opção de configuração para os cenários experimentais ficou definida como *algoritmo de intermediação sem QoS*.

É interessante mencionar que é possível acrescentar novos parâmetros de QoS para que o QBroker possa considera-los no processo de intermediação, porém, são necessárias algumas modificações na classe *Cloudlet* e na classe *QBroker*. Na classe *Cloudlet* será necessária a inserção dos parâmetros de QoS no formato de uma lista de nomes de parâmetros com seus respectivos valores, dessa forma, o *broker* pode localizar por meio da lista todos parâmetros enviados na requisição. Na classe *QBroker* será necessária a sobrescrita do método de intermediação de modo que se acrescente os comandos de programa para as regras de negócio inerentes aos novos parâmetros de QoS.

5.6 Considerações Finais

Neste capítulo foi apresentada e discutida a arquitetura CloudSim-BEQoS e suas características. Além disso, foram apresentadas as principais classes que compõem a arquitetura e uma explanação geral sobre seus objetivos, atributos e funcionalidades. Um diagrama da arquitetura CloudSim-BEQoS em camadas foi apresentado e explicado de modo a evidenciar as inovações propostas e implementadas.

Uma explanação mais direcionada à arquitetura QBroker, a qual se faz objeto principal desta tese, foi realizada, apresentando componentes estruturais do componente QBroker, modos de operação, relacionamento do mesmo com as informações de estado do ambiente computacional distribuído e negociação com provedores considerando ambiente de nuvem híbrida. Um diagrama em camadas a respeito do QBroker também foi apresentado com intuito

de esclarecer as interações entre o *cloud broker* durante a execução do serviço de mediação entre clientes e provedores de nuvem.

Foi apresentada uma discussão sobre os códigos-fonte de programa da classe QBroker e seus principais atributos de classe, constantes e métodos de instância implementados.

Finalizada a etapa de discussão da arquitetura de computação em nuvem CloudSim-BEQoS e a arquitetura QBroker, será apresentada no próximo capítulo uma avaliação de desempenho direcionada ao componente QBroker, considerando fatores para provimento de QoS para ambientes de nuvem cujo contexto seja híbrido.

6. Avaliação de desempenho da arquitetura QBroker

6.1 Considerações Iniciais

Este capítulo apresenta a avaliação de desempenho de três cenários com objetivo de validar a arquitetura QBroker e de demonstrar como a arquitetura proposta pode ser utilizada para avaliar diferentes características de uma nuvem híbrida. Nos três cenários será considerado um ambiente contendo um provedor de nuvem privada e um provedor de nuvem pública.

O intuito de tal modelagem de ambiente computacional foi a de simular uma situação comum nos dias atuais para o uso de sistemas computação em nuvem: o emprego de uma infraestrutura de nuvem privada pertencente a uma corporação e cujo uso não incorre em custos de locação de recursos sob demanda e uma infraestrutura de nuvem pública pertencente a um provedor de serviços em nuvem no âmbito da Internet.

Para permitir a demonstração das características de implementação do QBroker foram elaborados três planejamentos de experimentos (três cenários) que dão destaque para as características principais deste *cloud broker*, tais como, provimento de qualidade de serviços, diferenciação de serviços e alocação dinâmica de serviços. Os resultados experimentais foram obtidos usando o *CloudSim Toolkit* e componentes da arquitetura CloudSim-BEQoS.

Uma discussão a respeito dos resultados obtidos nas simulações, bem como as considerações a respeito das metas alcançadas pela implementação do QBroker são também apresentadas.

6.2 Descrição do Ambiente de Testes

Todos os experimentos apresentados no capítulo corrente são baseados na mesma modelagem de ambiente de testes criada com uso do simulador de computação em nuvem CloudSim (CALHEIROS *et al.*, 2011) na versão 3.0.3.

Na Figura 6.1 é apresentado o diagrama do ambiente de testes utilizado em todos os cenários experimentais.

Figura 6.1: Ambiente de testes modelado para os experimentos de simulação.

Conforme é possível verificar por meio dos elementos da Figura 6.1, o ambiente de testes é constituído por uma quantia fixa de 150 instâncias de clientes virtuais, os quais foram configurados com modo de submissão de tempo real de simulação. O fluxo de comunicação dos clientes com o ambiente de nuvem considera o envio de requisições e a recepção das respostas. A quantidade de 150 clientes foi definida com base na execução de experimentos prévios nos quais essa quantia se mostrou suficiente para gerar carga de trabalho para o ambiente de nuvem híbrida utilizando da capacidade da infraestrutura disponível.

A comunicação direta dos clientes é realizada com o componente QBroker, o qual efetua as atividades de negociação para com os recursos computacionais dos provedores de nuvem.

Conforme mencionado no anteriormente, o ambiente de nuvem híbrida oferece recursos de computação em nuvem por meio de uma nuvem privada e um provedor de nuvem

pública. Nesse contexto de ambiente (ainda conforme Figura 6.1), o fluxo de comunicação entre a entidade QBroker e os *datacenters* se dá por meio do escalonamento das requisições para o provedor cuja negociação obtiver sucesso (conforme verificação de requisitos de qualidade de serviço) e da obtenção da resposta da execução das tarefas nos provedores de nuvem. O QBroker, como elemento de intermediação e QoS, pode escalonar tarefas tanto para a nuvem privada quanto para a nuvem pública, dependendo da carga a que a nuvem privada está submetida e dos parâmetros de QoS.

É importante destacar, no diagrama da Figura 6.1, o fluxo de comunicação do QBroker com os monitores instanciados no ambiente de testes, no caso, o monitor de requisições e o monitor de carga de trabalho. Tais componentes são colaboradores do processo de intermediação, pois, sempre que o *cloud broker* necessita de informações acerca do estado dos recursos computacionais, os monitores são os principais provedores dessas informações. Os monitores, por sua vez, possuem fluxos de comunicação com a infraestrutura de *datacenters* de modo que possam efetuar suas atividades monitoramento, como por exemplo, efetuar a contagem de tarefas que chegam ao *cloud broker*, contagem de tarefas processadas e não processadas, verificação da porcentagem de carga de trabalho dos *datacenters*, entre outras possibilidades.

Antes de prosseguir para a discussão dos elementos de configuração do ambiente de nuvem modelado para os experimentos da tese, é importante destacar que os valores considerados para a caracterização dos *datacenters*, VMs e clientes foram criados com objetivo de tornar um ambiente de nuvem com capacidade de processamento e carga de trabalho suficientes para evidenciar as características de intermediação propostas. As configurações foram definidas com execuções de experimentos prévias ao planejamento experimental de forma que o montante de carga de trabalho fosse suficiente para estimular toda a infraestrutura disponível no ambiente simulado. Portanto, os valores considerados não são baseados em infraestruturas de *datacenters* reais e nem em aplicações reais. A estratégia do uso de valores fixos foi para a melhor percepção do comportamento dos fatores levados em consideração nas avaliações de desempenho. A questão da escalabilidade do *cloud broker* proposto também não foi considerado, utilizando-se, para todos os ambientes de simulação, apenas uma única instância do mesmo.

É oportuno informar que todos os experimentos foram repetidos 10 vezes. Cada repetição teve a duração total de 9000 segundos (ou instantes de simulação baseados no *clock*

tick do CloudSim). Os valores obtidos na execução de experimentos possuem um intervalo de confiança de 95% de acordo com a tabela *T-student*.

6.2.1 Configurações dos *Datacenters*

Para maior detalhamento da composição e quantificação dos componentes de infraestrutura do ambiente de testes, são apresentadas informações de configuração dos *datacenters* conforme Tabela 6-1 e Tabela 6-2.

Tabela 6-1: Configurações de *Host* da nuvem privada.

<i>Datacenter</i> de Nuvem Privada - Configurações	
MIPS por Núcleo:	10.000
Núcleos por <i>Host</i> :	4
RAM:	8.000 Mb
Largura de Banda:	1.000 Mbps
<i>Storage</i> :	500.000 Mb
Sistema Operacional:	Linux
<i>Hypervisor</i> :	Xen
Número total de <i>Hosts</i> :	5

Na Tabela 6-1 é possível verificar que o *datacenter* de nuvem privada foi instanciado com um número total de 5 *Hosts*. Cada um dos *Hosts* é equipado com um processador com 4 núcleos de processamento (elementos de processamento). Cada núcleo possui a capacidade de execução de 10.000 MIPS (Milhões de Instruções por Segundo). Outras configurações de capacidade do *Host* são memória dinâmica (RAM) de 8.000 megabytes, largura de banda da interface de rede de 1.000 Megabit/segundo e espaço em disco rígido (*storage*) de 500 gigabytes. Com relação à instalação de *software* de base para o uso dos *Hosts* no ambiente simulado, considerou-se o sistema operacional Linux e o hipervisor considerado foi o Xen. É importante esclarecer que essa configuração mencionada *software* de base não gera nenhuma influência prática durante a simulação, mas serve como elemento descritor para estipular uma configuração próxima ao mundo real.

Tabela 6-2: Configurações de *Host* da nuvem pública.

<i>Datacenter</i> de Nuvem Pública - Configurações	
MIPS por Núcleo:	20.000
Núcleos por <i>Host</i> :	8
RAM:	32.000 Mb
Largura de Banda:	10.000 Mbps
<i>Storage</i> :	1.000.000 Mb
Sistema Operacional:	Linux
<i>Hypervisor</i> :	Xen
Número total de <i>Hosts</i> :	2

Conforme as informações da Tabela 6-2, é possível observar as configurações dos *Hosts* pertencentes ao *datacenter* de nuvem pública. O número total de equipamentos de computação locados na nuvem pública foi de até 2 *Hosts*. Cada *Host* foi configurado com um

processador com 8 núcleos de processamento. Cada núcleo possui a capacidade de execução de até 20.000 MIPS.

Além disso, outras configurações complementares dos *Hosts* de nuvem pública são: memória dinâmica (RAM) de 32.000 megabytes, largura de banda de rede de 10.000 Megabit/segundo e espaço de armazenamento em disco de aproximadamente 1 Terabyte. Da mesma forma como foi discutido para a Tabela 6-1, as configurações de *software* de base para as aplicações ficaram definidos o sistema operacional Linux e o Xen como hipervisor.

Embora a quantidade de máquinas da nuvem privada tenha um número menor em relação à nuvem pública, é possível verificar, comparando os valores das tabelas Tabela 6-1 e Tabela 6-2 que os *Hosts* da nuvem pública possuem características de configuração superiores, destacando o valor da capacidade de processamento dos núcleos de processamento, o qual é o dobro do valor na nuvem privada.

6.2.2 Configurações das Máquinas Virtuais

As configurações das VMs utilizadas no ambiente simulado seguem um conjunto de valores que se aproximam das características de seus *Hosts* para dar maior aproveitamento, principalmente, aos núcleos de processamento alocados para as aplicações e tarefas.

As configurações de VMs das nuvens privada e pública são apresentadas conforme Tabela 6-3 e Tabela 6-4.

Tabela 6-3: Configurações de VMs do *datacenter* da nuvem privada.

<i>Datacenter Privado – Configurações de VMs</i>	
MIPS por Núcleo:	10.000
Número de Núcleos:	1
RAM:	2.000 Mb
Largura de Banda:	100 Mbps
Tamanho de imagem:	125.000 Mb
VMM:	Xen
Número total de VMs:	20

Um número total de 20 VMs foi criado no *datacenter* de nuvem privada (conforme dados da Tabela 6-3), sendo cada VM configurada para o uso de 1 núcleo de processamento. As demais configurações de VM são: 2 gigabytes de memória dinâmica, 100 Megabit/s de largura de banda de rede, 120 gigabytes de espaço em disco virtual e a supervisão do gerenciamento das VMs é feita pelo hipervisor Xen.

Tabela 6-4: Configurações de VMs do *datacenter* da nuvem pública.

Datacenter Público – Configurações de VM	
MIPS por Núcleo:	20.000
Número de Núcleos:	1
RAM:	4.000 Mb
Largura de Banda:	1.000 Mbps
Tamanho de imagem:	250.000 Mb
VMM:	Xen
Número total de VMs:	10

No cenário de simulação implementado, foi criado um número total de 10 VMs homogêneas no *datacenter* de nuvem pública (conforme Tabela 6-4), sendo que cada VM foi configurada com 1 núcleo de processamento de 20.000 MIPS, memória dinâmica de 4 gigabytes, largura de banda de rede 1 gigabit/s e tamanho de imagem de disco virtual de 250 gigabytes. Todas as VMs são gerenciadas pela plataforma do hipervisor Xen.

6.2.3 Configuração da Demanda de Serviços e dos Clientes

Outra questão envolvendo o cenário de computação em nuvem simulado são as configurações da camada cliente que foram implementadas considerando uma sistemática de geração de demanda de serviço e uma quantidade de clientes fixa. Os valores considerados não se reportam ou caracterizam nenhum tipo de aplicação real, porém, permitem a percepção de demandas de requisição com pesos diferentes, tornando demanda de serviço heterogênea.

Com relação à geração de demanda de serviço, foi implementada uma tabela de identificadores de serviços e suas demandas em MI (Milhões de Instruções). A demanda de serviço é atribuída para cada *cloudlet* com base no ID de serviço solicitado conforme uma distribuição exponencial específica para cada serviço. A distribuição exponencial considerada tem o valor médio de 70.000 MI. O número total de serviços possíveis considerados no cenário foram 5 serviços. É importante lembrar que a demanda em MI é aplicada ao campo *length* de cada *cloudlet*, o que especifica o comprimento de cada tarefa. Os valores de demanda de serviços considerados nos experimentos constam da Tabela 6-5.

Tabela 6-5: Configuração da demanda de serviços.

ID Serviço	Demanda (MI)
S1	30.000
S2	50.000
S3	70.000
S4	90.000
S5	110.000

A quantidade de entidades cliente foi fixada em 150 unidades para todos os cenários. A configuração do modo de submissão das entidades clientes em tempo de simulação, foi o do envio de requisições em tempo real (REAL_TIME_REQUESTS), o que significa que as requisições serão enviadas pelos clientes durante o transcorrer do relógio lógico do CloudSim,

criando um processo de chegada mais realístico e fidedigno ao mundo real. A geração dos IDs (identificadores) de serviço a serem inseridos dentro de cada requisição também foi feita de modo randômico considerando apenas 5 serviços.

Ainda referente à configuração das entidades clientes, é importante destacar que o atributo de QoS considerado em cada *cloudlet* foi o tempo máximo de execução (*maxExecutionTime*). Para a obtenção do valor deste campo de restrição de QoS foi elaborado um método na classe Cliente de modo a garantir que a geração dos tempos máximos de execução seja proporcional ao tamanho da *cloudlet*. Para o cálculo gerador do tempo máximo de execução (*maxExecutionTime*) o procedimento é o seguinte: são somados, o cálculo de tempo de execução da *cloudlet* conforme a capacidade em MIPS da VM mais lenta do sistema (no caso dos experimentos desta tese, esse valor é de 10.000 MIPS), a latência de rede para enviar a requisição e receber a resposta (valor obtido por meio do método *getDelay* da classe *NetworkTopology*) e o tempo médio de fila do ambiente (obtido a partir da execução experimentos de usados como *benchmark* considerando a configuração de ambiente de nuvem privada).

Para tornar a demanda de serviço heterogênea, o método *getRandomServiceIdWithDistribution* da classe Cliente gera amostras aleatórias dos IDs de serviço usando como controle uma distribuição em porcentagem. A distribuição considerada pode ser observada pelos dados que constam da Tabela 6-6.

Tabela 6-6: Distribuição da geração aleatória de serviços para as requisições das entidades cliente.

ID Serviço	Distribuição (%)
S1	5,0
S2	15,0
S3	60,0
S4	19,0
S5	1,0

Com base nas configurações apresentadas, foi possível obter significativos resultados de simulação, influenciando as variáveis de resposta consideradas nos experimentos.

6.2.4 Descrição das variáveis de resposta

Para a obtenção de informações de retorno nos conjuntos de experimentos, foram selecionadas algumas variáveis de resposta que são descritas a seguir:

- **Tempo de Resposta:** aferida em segundos (ou instantes de simulação) considerando a quantia de tempo dispendida em enviar uma requisição do cliente até a VM de um provedor e o seu retorno de volta ao cliente.

- **Porcentagem de Requisições Processadas:** considera as requisições que foram processadas com *status* de sucesso.
- **Porcentagem de Requisições Não Atendidas:** considera as requisições que não puderam ser atendidas pelo *cloud broker* por não atender ao requisito de QoS de tempo máximo de execução (*maxExecutionTime*).
- **Número de Requisições Processadas:** considera o número absoluto de requisições que foram processadas com *status* de sucesso.
- **Número de Requisições Não Atendidas:** considera o número absoluto das requisições que não foram atendidas pelo *cloud broker* por não atender ao requisito de QoS de tempo máximo de execução (*maxExecutionTime*).

As variáveis de resposta da avaliação foram selecionadas com o objetivo de demonstrar aspectos importantes diretamente relacionados com o desempenho do sistema de nuvem computacional.

Assim, o tempo de resposta é uma variável muito utilizada, pois, trata-se de uma informação que pode ser aferida diretamente pelo usuário e acaba sendo o principal componente de aferição de desempenho para o mesmo.

As variáveis de porcentagem ajudam a verificar o grau de eficiência que o *cloud broker* está obtendo em lidar com a carga de trabalho imposta ao sistema de nuvem. A porcentagem de requisições processadas demonstra o efetivo sucesso em escalonar e executar tarefas no ambiente de nuvem.

A porcentagem de requisições não atendidas permite a análise do percentual de rejeição da carga de trabalho enviada durante o tempo de simulação.

Os número absoluto de requisições processadas e não processadas são variáveis que permitem a análise mais detalhada da eficiência operacional do modo de operação do *cloud broker* por meio da averiguação da quantidade de tarefas que foram ou não atendidas durante o tempo de execução dos experimentos.

6.3 Análise dos resultados experimentais

No projeto experimental foram elaborados três conjuntos de experimentos (ou cenários) com objetivo de apresentar e evidenciar as características implementadas no processo de intermediação executado pelo QBroker.

6.3.1 Cenário 1: Demonstração de QoS

O primeiro cenário discutido é o de demonstração de QoS. A Tabela 6-7 apresenta o resumo do planejamento de experimentos criado para o cenário em questão.

Para esse cenário, foram considerados dois fatores: algoritmo de escalonamento e tipo de nuvem.

Tabela 6-7: Planejamento de experimentos do cenário de evidenciação de QoS.

Experimento	Algoritmo de Escalonamento	Tipo de Nuvem	Número De VMs	Número de Serviços Alocados
A	Round Robin	Privada	PRV=20	-
B	Round Robin	Híbrida	PRV=20+PUB=10	-
C	Intermediação com QoS	Privada	PRV=20	5
D	Intermediação com QoS	Híbrida	PRV=20+PUB=10	5
E	Intermediação sem QoS	Privada	PRV=20	5
F	Intermediação sem QoS	Híbrida	PRV=20+PUB=10	5

Como é possível perceber observando a Tabela 6-7, para o fator algoritmo de escalonamento foram considerados três níveis, isto é, o algoritmo Round Robin foi comparado com experimentos usando o algoritmo de intermediação do QBroker com e sem tratativa de QoS.

Nos experimentos com intermediação foram alocados os 5 serviços considerados no ambiente em todas as VMs de modo a tornar justa a comparação com o Round Robin, o qual não possui a política de seleção de serviços.

Para o fator tipo de nuvem foram considerados dois níveis: nuvem privada e nuvem híbrida. Para a nuvem privada foi considerada a configuração especificada na Tabela 6-3 e para a nuvem pública a configuração da Tabela 6-4.

Esses experimentos foram realizados por meio de um planejamento fatorial completo considerando os seguintes fatores: algoritmo de escalonamento e tipo de nuvem. Os resultados referentes à variável de tempo médio de resposta constam na Figura 6.2.

Os resultados obtidos para a variável de tempo médio de resposta (Figura 6.2) permitem perceber que, nos cenários de nuvem privada (experimentos A, C e E), o algoritmo de intermediação do QBroker se mostrou eficiente, pois, no experimento A, com Round Robin, o tempo de resposta foi de 49,08 segundos enquanto no experimento E, com intermediação sem QoS, obteve um desempenho melhor com tempo médio de 45,13 segundos (cerca de 8,05% mais rápido). Comparando-se ainda o experimento A com o experimento C, isto é, considerando a intermediação no com QoS, o desempenho foi ainda melhor perante os dois outros

experimentos, obtendo o valor de 18,12 segundos (cerca de 63,08% mais rápido em relação à variável de resposta do que o experimento A e 59,85% mais rápido do que o experimento E).

Figura 6.2: Gráfico de tempo médio de resposta do cenário de demonstração de QoS.

Ainda nos resultados com tempo de resposta médio (Figura 6.2), considerando-se os experimentos com cenários de nuvem híbrida (experimentos B, D e F), o algoritmo de intermediação do QBroker também apresentou eficiência e ganhos em desempenho. O experimento B, o qual considerou o uso do algoritmo Round Robin teve o tempo médio de resposta de 30,86 segundos, enquanto o experimento F, considerando intermediação sem QoS, obteve 19,32 segundos de resultado, o que significa um desempenho melhor (cerca de 37,40% mais rápido). Na análise do experimento D, com intermediação com QoS, o valor médio obtido foi melhor do que os outros dois experimentos, resultando na marca de 17,57 segundos no tempo de resposta (cerca de 43,07% mais rápido do que o experimento B e 9,08% mais rápido em relação ao experimento F).

Os resultados obtidos com os algoritmos de intermediação do QBroker apresentam um resultado melhor em desempenho porque a política de escalonamento de tarefas considera a seleção dos recursos (VMs) conforme o serviço de nuvem requisitado e conforme a validação do parâmetro de QoS de tempo máximo de execução (*maxExecutionTime*). Assim, com várias atividades internas do algoritmo de intermediação pode-se efetuar um melhor atendimento aos clientes, o que resultou em tamanhos médios de fila menores e descarte das requisições que não podiam ser atendidas no *deadline* solicitado. Logo, tais resultados corroboram com a premissa deste trabalho de que a nova arquitetura do QBroker oferece ganhos em desempenho para uma das principais variáveis de resposta observada pelos usuários finais de sistemas de computação em nuvem, isto é, o tempo de resposta das requisições de serviço.

Também é possível visualizar diferenças de como os algoritmos de escalonamento de tarefas se comportaram nos cenários de simulação de acordo com as variáveis de porcentagem de requisições processadas e porcentagem de requisições não atendidas. Conforme os resultados apresentados pelas variáveis de resposta referentes às porcentagens de requisições processadas e não atendidas (Figura 6.3 e Figura 6.4) destacam-se os experimentos C e D, os quais consideraram cenários com nuvem privada e híbrida respectivamente, com algoritmo de intermediação com QoS, pois foram os únicos cenários restritivos na questão da não aceitação de requisições devido à violação do parâmetro de QoS de tempo máximo de execução (*maxExecutionTime*).

Figura 6.3: Gráfico de porcentagem média de requisições processadas do cenário de demonstração de QoS.

Figura 6.4: Gráfico de porcentagem média de requisições não atendidas do cenário de demonstração de QoS.

No experimento C (conforme Figura 6.3 e Figura 6.4), tendo em vista que o algoritmo de intermediação com QoS do QBroker só encaminha a um datacenter as requisições cujo *deadline* solicitado no tempo máximo de execução (*maxExecutionTime*) informado como atributo de QoS em cada requisição (*cloudlet*), o QBroker processou 21,09% das requisições enviadas pelos clientes e rejeitou os outros 78,92%. Assim, para o provimento de QoS foi necessário considerável número em descarte de tarefas. No experimento D, usando da mesma justificativa quanto ao método de intermediação com QoS, o QBroker processou 79,72% das requisições e rejeitou os outros 20,28%. Nos demais experimentos não houve registro de rejeições de requisições, vindo a variável de resposta de porcentagem de requisições processadas a obter o valor constante de 100%.

Outras variáveis de resposta consideradas nos experimentos e que serão discutidas em conjunto serão: o tempo médio de fila, número de requisições processadas e número de requisições não atendidas.

Figura 6.5: Gráfico de Tempo Médio de Fila do cenário de demonstração de QoS.

Conforme as informações obtidas nos experimentos e analisando-se a partir da variável de tempo médio de fila (conforme Figura 6.5), é possível notar que as filas médias nos experimentos com Round Robin foram consideravelmente maiores do que nos experimentos com intermediação com e sem QoS. Isso ocorre porque o algoritmo Round Robin, por sua característica de alternância cíclica na atribuição de tarefas aos recursos de nuvem sem considerar QoS e nem SOA, acaba por alocar as tarefas de modo revezado entre as VMs, fato

que causa filas maiores e uma baixa eficiência também na variável de número absoluto de requisições processadas (conforme Figura 6.6).

Ainda com relação aos resultados da Figura 6.5, é possível notar que umas das razões pelo desempenho otimizado na variável de tempo médio de resposta (Figura 6.2) possui relação direta com os tempos médios de fila menores que foram alcançados por meio dos algoritmos de intermediação com e sem QoS.

O algoritmo de intermediação com QoS (experimentos C e D) permitiram ainda alcançar maior desempenho na variável de número absoluto de requisições processadas (Figura 6.6) em relação ao algoritmo Round Robin. Os experimentos C e D processaram, respectivamente, 2006 e 15.769 requisições à mais. O mesmo comportamento ocorreu no algoritmo de intermediação sem QoS, no qual foi possível processar, nos experimentos E e F, respectivamente, 1870 e 15.937 requisições à mais. Esses registros revelam que, neste primeiro cenário, a arquitetura QBroker obteve êxito no processo de mediação ao impactar de maneira positiva nas variáveis de resposta consideradas para os experimentos.

Figura 6.6: Gráfico do número médio de requisições processadas do cenário de demonstração de QoS.

Com relação à variável de número absoluto de requisições não atendidas (Figura 6.7) foi possível notar, assim como no gráfico de porcentagem de requisições não atendidas, que apenas nos experimentos com intermediação com QoS é que houve descarte de requisições. Os valores de descarte obtidos nos experimentos C e D, isto é, respectivamente 97.122 e 13.057 representam tarefas cujo *deadline* representado pelo atributo de QoS de tempo máximo de

execução não pode ser cumprido. Apesar da grande quantia de descartes verifica-se que o número total de requisições processadas ainda foi superior ao algoritmo Round Robin.

Figura 6.7: Gráfico de número médio de requisições não atendidas do cenário de demonstração de QoS.

Um estudo realizado em paralelo a observação das variáveis de resposta já mencionadas, visa avaliar em qual infraestrutura de nuvem as tarefas (*cloudlets*) foram executadas por meio dos registros de sua execução no relatório de saída do simulador CloudSim. A Figura 6.8 contém um gráfico no qual é apresentada tal informação.

Analisando o gráfico da Figura 6.8 é possível verificar que os experimentos A, C e E executaram 100% de suas requisições na nuvem privada, isso ocorre porque, no caso específico desses três experimentos, o cenário considerado foi o de nuvem privada apenas.

Nos resultados dos experimentos B, D e F (conforme apresentado na Figura 6.8), o cenário considerado é o de nuvem híbrida, portanto, é possível verificar a divisão da alocação de requisições em ambas as estruturas de nuvem privada e pública. Analisando as barras do experimento D, é possível observar que o algoritmo de intermediação com QoS, considerando cenário de nuvem híbrida, sempre provê prioridade em escalonar tarefas primeiro na nuvem privada e a seguir, após esgotadas as possibilidades, executa o escalonamento na nuvem pública. Deste modo, embora a diferença de valores não seja tão significativa, esta é a razão para o resultado de mais requisições serem executadas na nuvem privada do que na nuvem pública. Já no experimento F, o uso de intermediação sem QoS apresenta um padrão de fluxo de execução bastante similar ao da intermediação com QoS, no entanto, ao invés de descartar os pedidos quando da não conformidade com o parâmetro de tempo máximo de execução

(*maxExecutionTime*), a tarefa é escalonada para uma VM ativa com o serviço solicitado com a menor fila existente. Isso faz com que as VMs públicas sejam usadas com mais frequência, porque as VMs públicas têm o dobro do poder de processamento em MIPS (conforme as configurações do ambiente de testes são 20.000 MIPS), o que torna as chances das VMs públicas terem um tamanho de fila menor maiores.

É importante destacar, no caso do experimento B com Round Robin, (ainda conforme Figura 6.8) que tal algoritmo de escalonamento de tarefas usa todos os recursos de distribuir as tarefas usando alternância cíclica. Já a explicação para maior alocação de tarefas em máquinas virtuais privadas é um comportamento causado por conta da modelagem do cenário de nuvem híbrida, no qual considerou-se a nuvem privada possui o dobro da quantidade de máquinas virtuais (20 VMs) em comparação com a infraestrutura locada na nuvem pública (10 VMs). Essa organização causa a distribuição em porcentagem apresentada no gráfico da Figura 6.8, o qual permite verificar uma proporção de 2 partes para a nuvem privada (66,57%) e uma parte para nuvem pública (33,43%) na distribuição das tarefas escalonadas.

Figura 6.8: Distribuição das requisições em porcentagem por tipo de provedor de nuvem do cenário de demonstração de QoS.

Outro estudo conduzido em paralelo aos experimentos com as variáveis de resposta do plano experimental foi a observação da porcentagem de consumo de recursos de nuvem dos experimentos. A Figura 6.9 contém os gráficos de acompanhamento do consumo de recursos de nuvem durante o tempo de simulação dos experimentos A e B. Nesse contexto, se faz necessário salientar que a porcentagem de consumo de recursos, no caso de experimentos com

cenário de nuvem híbrida, refere-se aos recursos de nuvem privada e também aos recursos de nuvem pública.

Figura 6.9: Porcentagem de uso de recursos de nuvem dos experimentos A e B do cenário de demonstração de QoS.

Conforme é possível verificar pela análise de gráficos da Figura 6.9, os experimentos A e B, os quais consideram o algoritmo Round Robin para o escalonamento de tarefas, obtiveram resultados que assinalam que a infraestrutura recursos de nuvem, embora tenha sido utilizada numa sistemática de alternância cíclica, acabou por não aproveitar todo o desempenho dos recursos disponíveis. Dessa forma, é possível concluir que tanto no cenário de apenas nuvem privada (experimento A) como no cenário de nuvem híbrida (experimento B) as VMs foram subutilizadas, fato que impactou no número de requisições processadas, o qual foi de cerca de 23.952 *cloudlets* para o experimento A e de 35.609 *cloudlets* para o experimento B.

Os resultados para a porcentagem de consumo de recursos de nuvem para os experimentos C e D são apresentados nos gráficos da Figura 6.10.

Figura 6.10: Porcentagem de uso de recursos de nuvem dos experimentos C e D do cenário de demonstração de

Os gráficos referentes ao consumo de recursos de nuvem dos experimentos C e D (Figura 6.10) revelam um comportamento bastante distinto comparando-os aos experimentos A e B (Figura 6.9). Nota-se que no intervalo dos 9.000 segundos de simulação, tempo total adotado para todos os experimentos, os experimentos C e D utilizaram 100,00% dos recursos disponíveis até o último instante de simulação. Esse comportamento refletiu em ganhos no

processamento de requisições de clientes. No experimento C foram processadas 25.766 *cloudlets* e no experimento D foram processadas 51.190 *cloudlets*. Usando os experimentos A e B como parâmetro de comparação, o experimento C, que é similar em configurações de fatores ao experimento A, obteve um ganho de 7,05% no processamento de requisições. Da mesma forma, comparando-se o experimento D com o experimento A, o experimento D obteve um ganho ainda mais significativo, obtendo um ganho de 30,44% em requisições processadas. É válido lembrar que os experimentos C e D usam do modo de operação de intermediação de serviços com QoS do QBroker.

Já os resultados do consumo de recursos de nuvem dos experimentos E e F (Figura 6.11) revelam um comportamento similar aos gráficos apresentados pelos experimentos C e D, logo, também consumiram 100,00% dos recursos disponíveis durante todo o tempo de simulação. A justificativa para essa ocorrência se dá por conta do algoritmo de escalonamento de tarefas, pois, embora os experimentos E e F se utilizem do modo de operação de intermediação de serviços sem QoS do QBroker, a estrutura principal de atividades do modo de operação segue sendo a mesma. A diferença é que nos cenários de intermediação sem QoS não há o descarte de tarefas, de modo que o número de requisições de clientes processadas foi superior a todos os experimentos do cenário atual.

Figura 6.11: Porcentagem de uso de recursos de nuvem dos experimentos E e F do cenário de demonstração de QoS.

No que diz respeito ao processamento de requisições (ainda segundo a Figura 6.11), o experimento E apresentou ganho de 7,28% em relação ao experimento A e de 0,22% em relação ao experimento C. Considerando o processamento de requisições experimento do F, o qual considera nuvem híbrida, apresentou ganho de 30,80% em relação ao experimento B e ganho de 0,52% em relação ao experimento D. É interessante verificar que mesmo o modo de operação do QBroker trabalhando em caráter de não priorizar qualidade de serviço é possível constatar

sensíveis ganhos em desempenho e aproveitamento da infraestrutura comparando-o ao algoritmo Round Robin.

Neste cenário foram apresentados resultados cujo objetivo foi a evidenciação da característica de provimento de qualidade de serviço implementada no QBroker. Por meio dos resultados experimentais foi discutida a avaliação do desempenho das variáveis de resposta consideradas para todos os planejamentos experimentais, a distribuição de tarefas executadas e o consumo de recursos de nuvem. Cabe ressaltar que é possível ainda verificar em avaliações futuras a influência de cada um dos fatores considerados no presente cenário.

6.3.2 Cenário 2: Alocação Estática de Serviços

No segundo cenário de experimentos, buscou-se evidenciar os efeitos obtidos conforme a alocação estática de serviços por meio da variação da quantidade de serviços alocados nas máquinas virtuais. A característica de diferenciar serviços por identificadores aproxima a arquitetura QBroker dos *cloud brokers* compatíveis com arquiteturas orientadas a serviços (SOA). Na Tabela 6-8 é apresentado o planejamento de experimentos do cenário de experimentos.

Tabela 6-8: Planejamento de experimentos do cenário de alocação estática de serviços.

Experimento	Algoritmo de Escalonamento de Tarefas	Tipo de Nuvem	Número de VMs	Número de Serviços Alocados
C	Intermediação com QoS	Privada	PRV=20	5
D	Intermediação com QoS	Híbrida	PRV=20+PUB=10	5
E	Intermediação sem QoS	Privada	PRV=20	5
F	Intermediação sem QoS	Híbrida	PRV=20+PUB=10	5
C'	Intermediação com QoS	Privada	PRV=20	2
D'	Intermediação com QoS	Híbrida	PRV=20+PUB=10	2
E'	Intermediação sem QoS	Privada	PRV=20	2
F'	Intermediação sem QoS	Híbrida	PRV=20+PUB=10	2

De acordo com o plano de experimentos (Tabela 6-8), é possível verificar que a configuração de experimentos é uma variação do planejamento de experimentos inicialmente feito no cenário de demonstração de QoS. Para esse cenário foram considerados 3 fatores. O fator algoritmo de escalonamento considera dois dos níveis também considerados no cenário 1 (intermediação com e sem QoS). O segundo fator denominado tipo de nuvem considera também os dois níveis descritos no cenário 1 e o terceiro fator considera o número de serviços alocados nas VMs, com níveis 5 e 2. Os experimentos C', D', E' e F' possuem as mesmas características de cenário do que, respectivamente, os experimentos C, D, E e F, contudo, o número de serviços alocado nas VMs é diferente. Nos experimentos C, D, E e F são alocados 5 serviços em todas as VMs instanciadas (da mesma forma considerada no cenário 1), enquanto nos experimentos C', D', E' e F', a quantidade de serviços alocados nas VMs é de 2. Para o cenário 2 também foi

considerado um planejamento do tipo fatorial completo. É oportuno lembrar que em todos os cenários onde se utilizou o algoritmo de intermediação do QBroker, os serviços existentes usam identificadores numerados de 1 a 5.

As informações dos resultados do cenário de alocação estática de serviços com relação ao tempo médio de resposta são apresentadas na Figura 6.12. É possível verificar, por meio da Tabela 6-8, que o número de serviços para cada VM nesse cenário é pré-definido no início da simulação, logo, não há ocorrência de tentativa de alocação de novos serviços. No caso específico dos experimentos C', D', E' e F', os serviços instanciados em cada VM usam um método de distribuição uniforme para os 5 serviços considerados.

Conforme a Figura 6.12, para este cenário de alocação estática de serviços, é possível notar que os experimentos C e D possuem valores bem próximos, embora não equivalentes estatisticamente. Comparando os experimentos C e C' percebe-se que C' obteve um tempo médio de resposta mais eficiente com 15,13 segundos. A mesma situação ocorre com os experimentos D e D', caso no qual o experimento D' obteve melhor desempenho no tempo de resposta, cujo valor foi de 14,02 segundos.

Figura 6.12: Gráfico de tempo médio de resposta do cenário de alocação estática de serviços.

A justificativa para os valores apresentados na Figura 6.12 é justamente a questão da distribuição dos serviços, pois, no caso dos experimentos C e D, todos os serviços possíveis

estão instanciados em todas as VMs do cenário, de modo que, embora possa oferecer mais possibilidades de escalonamento para as requisições, acabam por obter um tamanho de fila médio maior e, nessa situação, ocorre decréscimo no desempenho assinalado na variável de tempo de resposta e também há uma porcentagem de descarte de requisições maior.

Os experimentos com apenas dois serviços e com uso de intermediação com QoS (conforme Figura 6.12), isto é, C' e D', embora tornem o cenário inicialmente mais restritivo quanto às opções de escalonamento das requisições para as VMs, geram um tamanho médio de fila diferenciado de uma VM para outra, pois, os serviços cuja função exponencial de demanda são maiores não são instanciados em todas as VMs, ocasionando, para esta situação em específico, um desempenho consideravelmente melhor no tempo de resposta.

Ainda com relação às informações dos experimentos na Figura 6.12, nos experimentos E, F, E' e F', os resultados tem outro posicionamento, isto é, muitas VMs possuem um tempo médio de fila menor do que nos experimentos C e D. Como nos experimentos E e F tem-se todos os serviços instanciados em todas as VMs do cenário e a disponibilidade acaba, portanto, sendo mais ampla, e, conforme já explanado, considerando-se que a última atividade da intermediação sem QoS é escalonar a requisição para a VM que possuir o serviço requerido instanciado com a menor fila. Nestes casos, os cenários com mais serviços oferecem mais oportunidades de escalonamento, o que faz com que os valores dos tempos médios dos experimentos E e F, isto é, 45,13 segundos e 19,32 segundos respectivamente, apresentem um melhor desempenho do que os experimentos E' e F' que possuem 2 serviços instanciados em todas as VM do cenário.

Os gráficos de desempenho de porcentagem de requisições processadas e porcentagem de requisições não atendidas do cenário de experimentos atual são apresentados na Figura 6.13 e Figura 6.14, respectivamente.

Corroborando com a argumentação apresentada para a variável de tempo médio de resposta, é possível observar (Figura 6.13 e Figura 6.14) que nos experimentos C e D obteve-se um valor menor de termos requisições processadas e, por sua vez, maior porcentagem de requisições não atendidas, conforme argumentação anterior sobre as considerações que envolvem a variável de tempo de resposta.

Figura 6.13: Gráfico de porcentagem média de requisições processadas do cenário de alocação estática de serviços.

Figura 6.14: Gráfico de porcentagem média de requisições não atendidas do cenário de alocação estática de serviços.

Ainda conforme a Figura 6.13 e Figura 6.14, nos demais experimentos (E, F, E' e F') que não consideram o parâmetro de QoS *maxExecutionTime* sempre se obtém 100,0% das requisições processadas, de modo que não há descarte de requisições.

Com relação às variáveis de resposta de tempo médio de fila (Figura 6.15), número de requisições processadas (Figura 6.16) e número de requisições não atendidas (Figura 6.17), um primeiro ponto a se notar, é que no cenário com 5 serviços, devido à maior disponibilidade de serviços, as VMs recebem mais requisições dos serviços de maior demanda, isto é, serviços S3, S4 e S5, fato que causa um impacto direto no tempo médio de fila. No cenário mais restritivo, nem todas as máquinas possuem os serviços de maior demanda, o que contribui para tempos médios de fila menores.

Nos experimentos de intermediação sem QoS a situação apresentada é contrária à da afirmativa anterior, isto é, a maior disponibilidade de serviços causa um melhor rateio das requisições entre as máquinas (experimentos E e F), visto que o parâmetro de QoS não precisa ser obrigatoriamente considerado.

Figura 6.15: Gráfico de tempo médio de fila do cenário de alocação estática de serviços.

Assim, o cenário restritivo leva desvantagem (experimentos E' e F'), no qual longas filas referentes às máquinas que possuem os serviços de maior demanda se formam.

A partir dos valores apresentados na Figura 6.15 é possível notar que as filas dos experimentos E' e F' chegando a ser, respectivamente, 70,12% e 99,10% maiores do que as dos experimentos E e F. A partir dessas constatações torna-se mais evidente o porque dos resultados obtidos e já explanados para a variável de tempo médio de resposta (Figura 6.12).

Com relação aos gráficos de resultado apresentados nas variáveis de número de requisições processadas e número de requisições não atendidas (Figura 6.16 e Figura 6.17), ao se comparar os experimentos que consideram intermediação com QoS (C, D, C' e D'), é possível notar que nos experimentos com cenário considerando todos os serviços em todas as máquinas (C e D), o número absoluto de requisições não atendidas é menor, totalizando 2.346 requisições a menos no experimento C e 40.050 requisições a menos no experimento D.

Pode-se afirmar, com base nas informações anteriormente mencionadas, que, havendo menor descarte, o sistema de nuvem trabalhou mais, fato que também pode ser constatado por meio da análise do gráfico de tempo médio fila o qual revela que o tamanho médio das filas dos experimentos C e D também foram maiores comparando com os valores obtidos nos experimentos C' e D'. O fato das filas das VMs estarem mais cheias também ocasionou impacto no tempo médio de execução das requisições.

É possível notar também que no experimento C houve um aumento de 25,00% no tempo de fila e aumento de 12,77% no tempo médio de execução das requisições em relação aos valores do experimento C'. Da mesma forma, no experimento D, houve um aumento de 40,93% no tempo de fila e aumento de 1,75% no tempo médio de execução em comparação aos respectivos valores do experimento D'.

Usando como premissa as informações supracitadas, é possível concluir que o escalonamento de tarefas por meio do modo de operação de intermediação com QoS considerando o cenário com menor número de serviços (2 serviços por VM) causou melhor desempenho em todas as variáveis de resposta consideradas, pois, o cenário mais restritivo leva a um maior valor de descarte, a uma utilização de recursos de nuvem menos intensa e, conseqüentemente, a tempos de resposta melhores.

Ainda considerando as informações da Figura 6.16 e Figura 6.17, na análise dos experimentos considerando intermediação sem QoS (E, F, E' e F'), é possível verificar uma situação oposta à discussão dos resultados de intermediação com QoS.

Figura 6.16: Gráfico de número médio de requisições processadas do cenário de alocação estática de serviços.

Partindo dos valores de resultado obtidos para o número de requisições processadas, nota-se que o cenário considerando todos os cinco serviços em todas as VMs (experimentos E e F), por oferecer maior disponibilidade de serviços, processou quantidades de requisições consideravelmente maiores do que os experimentos com cenário restrito a dois serviços por VM. Assim, observa-se no experimento E um aumento de 51,47% no número de requisições processadas, comparando-o ao experimento E'. Da mesma forma, nota-se no experimento F um aumento no valor de 51,26% no número de requisições processadas em comparação ao experimento F'. Considerando o tempo médio de fila, os experimentos E e F obtiveram também valores com desempenho melhores em relação aos experimentos E' e F'. Assim, os experimentos E e F obtiveram um tempo médio de fila na ordem de 58,78% e 50,23% menores em comparação aos experimentos E' e F'. É possível notar também que o tempo médio de fila do experimento F foi 3,70% menor. É importante lembrar que nos experimentos com intermediação sem QoS não há descarte de requisições. A partir do exposto, é possível concluir que os tempos de resposta melhores foram obtidos pelos experimentos E e F foi devido a maior oferta de serviços nas VMs, o que ocasionou impacto direto no tempo médio de fila, variável de resposta que possui uma relação direta com o aumento de desempenho.

Figura 6.17: Gráfico de número médio de requisições não atendidas do cenário de alocação estática de serviços.

O parecer final sobre o cenário de experimental 2 é de que a alocação estática de serviços implementada pela arquitetura QBroker permitiu diferentes perspectivas de desempenho e QoS conforme as configurações consideradas pelos cenários de simulação. Foi possível diagnosticar que nem sempre a maior disponibilidade de unidades de serviços em cada recurso de nuvem (VM) significa a obtenção de maior desempenho na variável de tempo de resposta. Da mesma forma, foi possível demonstrar que a adoção de intermediação sem QoS apresentou resultados inversamente proporcionais aos experimentos com QoS, demonstrando que, para os cenários de aplicação nos quais não há descarte de requisições, a maior disponibilidade de serviço acaba por trazer um desempenho consideravelmente melhor. Todas essas prerrogativas podem ser consideradas verdades tanto nos experimentos em ambiente computacional considerando apenas nuvem privada quanto nos experimentos considerando nuvem híbrida.

Neste cenário experimental buscou-se evidenciar a alocação estática de serviços e os comportamentos decorrentes dessa característica implementada na arquitetura QBroker nas variáveis de resposta consideradas pelo planejamento experimental.

É oportuno mencionar que os gráficos referentes a alocação de tarefas por tipo de nuvem e os gráficos de porcentagem de consumo de recursos de nuvem dos experimentos C', D', E' e F' serão discutidos em conjunto com os gráficos dessa mesma temática no cenário

experimental de número 3. Assim como no primeiro cenário experimental, cabe ressaltar da possibilidade da execução da influência dos fatores considerados no presente cenário.

6.3.3 Cenário 3: Alocação Dinâmica de Serviços

No terceiro cenário de experimentos o objetivo principal foi o de demonstrar a alocação dinâmica de serviços em tempo de simulação realizada pela arquitetura QBroker. É interessante informar que os custos envolvidos com o processo de alocação de instâncias de serviços não são computados em nenhum dos cenários experimentais desta tese.

A Tabela 6-9 possui as informações do planejamento de experimentos criado para o cenário experimental de alocação dinâmica de serviços.

Tabela 6-9: Planejamento de experimentos do cenário de alocação dinâmica de serviços.

Experimento	Algoritmo de Escalonamento de Tarefas	Tipo de Nuvem	Número de VMs	Número de Serviços Alocados
C'	Intermediação com QoS	Privada	Estática PRV=20	2
D'	Intermediação com QoS	Híbrida	Estática PRV=20+PUB=10	2
E'	Intermediação sem QoS	Privada	Estática PRV=20	2
F'	Intermediação sem QoS	Híbrida	Estática PRV=20+PUB=10	2
C''	Intermediação com QoS	Privada	Dinâmica PRV=(10 + 10 Stdbby)	2
D''	Intermediação com QoS	Híbrida	Dinâmica PRV=(10 + 10 Stdbby) + PUB=(10 Stdbby)	2
E''	Intermediação sem QoS	Privada	Dinâmica PRV=(10 + 10 Stdbby)	2
F''	Intermediação sem QoS	Híbrida	Dinâmica PRV=(10 + 10 Stdbby) + PUB=(10 Stdbby)	2

De acordo com o plano de experimentos (Tabela 6-9), é possível notar que foi feita uma junção dos experimentos com número fixo de serviços (C', D', E' e F') com outros quatro experimentos que executam alocação dinâmica de serviços. Desta forma, nesse cenário são considerados três fatores. Os fatores sobre algoritmos de escalonamento e tipo de nuvem consideram os mesmos níveis do cenário 2. O terceiro fator considera a existência ou não de alocação dinâmica de serviços tanto na nuvem privada quanto na nuvem pública. No caso da nuvem privada com alocação dinâmica, 5 VMs possuem 2 serviços instanciados enquanto as outras 15 VMs permanecem em estado *Standby*. Já nos cenários com nuvem híbrida com alocação dinâmica, a nuvem privada é inicializada com a mesma configuração já mencionada e a nuvem pública é inicializada com todas as VMs em estado de *Standby*. Esse experimento foi realizado considerando um planejamento fatorial completo.

Os resultados referentes à variável de tempo médio de resposta do cenário atual são apresentados na Figura 6.18.

A partir da análise do gráfico da Figura 6.18, é possível observar que os experimentos que consideram algoritmo de intermediação com QoS (C', D', C'' e D'') apresentam uma diferença em desempenho. Nota-se que nos experimentos com alocação dinâmica de serviços o tempo de resposta foi maior. O tempo de resposta do experimento C', o qual considerou alocação estática de serviços e nuvem privada, foi 14,66% mais rápido do que o C'', com alocação dinâmica de serviços. Uma situação similar ocorre entre os experimentos que possuíam nuvem híbrida nos cenários, isto é, o experimento D', com alocação estática de serviços, obteve tempo de resposta 17,67% mais rápido do que o experimento D'', o qual usou alocação dinâmica de serviços. Tal resultado era esperado, pois, quando do início da execução dos experimentos em simulação, os experimentos C'' e D'' tem apenas 5 VMs à disposição para o escalonamento de tarefas, logo, a alocação dinâmica de serviços só é efetuada quando há real necessidade devido ao não cumprimento do parâmetro de QoS de tempo máximo de execução.

Figura 6.18: Gráfico de tempo médio de resposta do cenário de alocação dinâmica de serviços.

Ainda referente à Figura 6.18, os experimentos que foram configurados com intermediação sem QoS (E', F', E'' e F'') tiveram um resultado diferente, pois, neste caso particular, os experimentos com alocação dinâmica de serviços tiveram notável diferença, apresentando um desempenho melhor. Para o entendimento do motivo do melhor desempenho nos

experimentos E'' e F'' é necessário reforçar que os experimentos E' e F' iniciam com 2 serviços por VM e com alocação estática sendo feita por meio de uma distribuição uniforme. Devido à referida distribuição realizada na alocação estática de serviços, os experimentos E' e F' acabam por ter um desempenho inferior para a variável de tempo de resposta, pois, o arranjo dos serviços causa a consequência de tempos médios de fila maiores. Já os experimentos E'' e F'', os quais possuem apenas 5 VMs que são inicialmente inicializadas com serviços usando distribuição uniforme, possuem a possibilidade de efetuarem uma estratégia de alocação de serviço dinâmica e sob demanda, logo levam significativa vantagem, dado que os serviços são alocados em função da necessidade real e conforme os serviços são solicitados nas requisições.

As informações de resultado das variáveis de porcentagem de requisições processadas e porcentagem de requisições não atendidas são apresentadas na Figura 6.19 e Figura 6.20.

Figura 6.19: Gráfico de porcentagem média de requisições processadas do cenário de alocação dinâmica de serviços.

Os experimentos nos quais foi empregado o algoritmo de intermediação sem QoS (E', F', E'' e F'') apresentam comportamento similar (conforme Figura 6.19), isto é, a variável de porcentagem de requisições processadas nesses experimentos foi de 100,0% e não houve descarte.

Figura 6.20: Gráfico de porcentagem média de requisições não atendidas do cenário de alocação dinâmica de serviços.

Já nos experimentos com intermediação com QoS, no caso dos experimentos considerando nuvem privada C' e C'' (ainda conforme a Figura 6.19 e Figura 6.20), apresentam valores porcentagens de requisições processadas e não atendidas próximos, revelando um comportamento parecido no ambiente restrito de recursos da nuvem privada. Quanto aos experimentos D' e D'', os quais consideram nuvem híbrida, o experimento D'' obteve um resultado melhor, pois, acabou processando uma porcentagem maior de requisições, isto se dá, conforme já explanado, por conta da alocação dinâmica de serviços em tempo de execução o que se revelou um recurso do QBroker que torna o atendimento aos clientes mais profícuo e causa um efeito “adaptativo” à demanda quando se observa os registros da alocação de serviços realizada durante a execução do experimento.

Com relação aos resultados referentes às variáveis de resposta de tempo médio de fila, requisições processadas e requisições não atendidas, é possível verificar que nos resultados apresentados pelo gráfico de tempo médio de resposta, foi possível perceber que, considerando intermediação com QoS, os experimentos com alocação estática de serviços (C' e D') obtiveram um resultado melhor diante dos experimentos com alocação dinâmica de serviços (C'' e D''). Já nos experimentos considerando intermediação sem QoS, os experimentos com alocação estática de serviços (E' e F') obtiveram um resultado inferior aos experimentos com alocação dinâmica de serviços (E'' e F''). Para os experimentos de intermediação com QoS (C', D', C'' e D''), partindo da análise do número de requisições não atendidas, é perceptível que os experimentos C' e D' tiveram maior número de descartes. Seguindo as proposições já citadas

no cenário experimental 2, sabe-se que, quando o descarte é maior, o sistema de nuvem não está trabalhando o tempo todo em capacidade máxima de processamento, logo, as filas das VMs tornam-se menores, fator que leva o tempo de resposta das requisições a um melhor desempenho. É possível comprovar essa relação de fatores observando os valores do número de requisições não atendidas, tempo de fila e tempo de execução das requisições. No caso dos experimentos C' e D', foi possível verificar que os experimentos obtiveram, respectivamente, os seguintes valores para as variáveis de resposta complementares: 2,50% e 179,48% de aumento no número de requisições não atendidas, 10,02% e 2,65% de diminuição no tempo médio de execução das requisições e 18,21% e 25,16% de redução no tempo médio de fila, em comparação com os experimentos C'' e D''.

Figura 6.21: Gráfico de tempo médio de fila do cenário de alocação dinâmica de serviços.

Para os experimentos com intermediação sem QoS (E', F', E'' e F''), como não há valor para a variável de resposta de número de requisições não atendidas, parte-se da análise dos valores do número de requisições processadas, o qual revela que os experimentos com alocação dinâmica, isto é, E'' e F'', obtiveram um desempenho melhor processando um número maior de tarefas. Assim, no caso dos experimentos E' e F', os valores obtidos nas variáveis de resposta complementares foram, respectivamente: 32,91% e 33,90% de redução no número de requisições processadas, 0,14% e 1,04% para o tempo médio de execução das requisições e 66,36% e 98,80% de aumento para o tempo médio de fila, quando comparados com os experimentos E'' e F''.

Figura 6.22: Gráfico de número médio de requisições processadas do cenário de alocação dinâmica de serviços.

Figura 6.23: Gráfico de número médio de requisições não atendidas do cenário de alocação dinâmica de serviços.

Nos experimentos com intermediação sem QoS, pode-se notar uma inversão do comportamento na variável de resposta de tempo médio de resposta, pois, como a alocação dinâmica de serviços se dá em função da demanda, isto é, conforme as solicitações dos clientes, foi notável o ganho de desempenho nos experimentos E'' e F''. O fato do algoritmo de intermediação sem QoS não efetuar descartes permitiu que as requisições de serviço, ao serem atribuídas para a VM com a menor fila possível para o serviço requisitado, obtivessem um

melhor resultado do que nos experimentos que já iniciam com todas as VMs contendo 2 serviços por meio de uma distribuição normal.

Todavia, embora não tenham obtido os melhores resultados, é possível notar que os experimentos C'' e D'' processaram número de requisições significativamente maior do que os experimentos C' e D', quase equiparando-se aos valores dos experimentos E'' e F''. Esse fato denota que a alocação dinâmica de serviços, embora ofereça um tempo de resposta um pouco maior do que a estática, conseguiu atender a um número muito maior de clientes e usou os recursos de nuvem de forma muito mais profícua, o que, conseqüentemente, representa melhor QoS do lado do provedor de nuvem.

No que diz respeito a alocação das requisições de acordo com os *datacenters* instanciados nos cenários, é apresentado na Figura 6.24 um gráfico de porcentagem das requisições executadas no *datacenter* para o qual foram escalonadas.

De acordo com as informações contidas no gráfico da Figura 6.24, os experimentos que consideram apenas nuvem privada (C', E', C'' e E'') possuem apenas uma barra mostrando que todas as requisições foram executadas no único *datacenter* disponível que é o de nuvem privada. Já em experimentos que consideraram na configuração do ambiente computacional uma nuvem híbrida (D', F', D'' e F'') têm diferentes distribuições das execuções das requisições de clientes.

Em experimentos com nuvem híbrida e intermediação com QoS (D' e D'') é possível ver que a maioria das requisições são sempre enviadas para a nuvem privada (Figura 6.24), isso ocorre por conta do processo de negociação implementado no QBroker, o qual prioriza sempre a atribuição de tarefas primeiro na infraestrutura privada e, se não for possível, então, em seguida, a infraestrutura de nuvem pública é escolhida. Esta abordagem é interessante e importante porque a alocação de serviços em uma nuvem privada com infraestrutura de propriedade da organização implica em economia. Por outro lado, o uso de recursos de nuvem pública sempre incorre em custos extras. Nos experimentos utilizando intermediação sem QoS (F' e F'') foram obtidos dois resultados diferentes: no experimento F' o comportamento segue o padrão dos experimentos D' e D'', destinando a maior parte das tarefas para a infraestrutura de nuvem privada; o experimento F'' teve um comportamento diferente, pois neste caso a distribuição foi quase igual entre a nuvem privada e nuvem pública.

Figura 6.24: Distribuição em Porcentagem das requisições por tipo de provedor de nuvem.

É possível notar (Figura 6.24) no caso do experimento F'', o qual considerou a alocação dinâmica de serviço e um máximo de 2 serviços por VM, que o algoritmo de intermediação sem QoS balanceou a carga de trabalho entre os dois datacenters e também levou a um uso mais intenso do número total de recursos.

Neste cenário também são apresentados resultados referentes a porcentagem de consumo de recursos de nuvem dos experimentos. A Figura 6.25 possui gráficos de resultados do uso de recursos referente aos experimentos C', D', E' e F'.

Conforme informações da Tabela 6-9, nos experimentos C', D', E' e F' (Figura 6.25) foi utilizado um número fixo de 2 serviços de nuvem por VM. A partir da análise dos gráficos de porcentagem de uso de recursos é possível notar que nos experimentos C' e D', os quais consideraram o algoritmo de intermediação com QoS, o uso de recursos ficou entre aproximados 80% e 100% considerando o cenário com apenas a nuvem privada (experimento C') e ficou entre 80% e 100% no cenário de nuvem híbrida (experimento D'), com alguns momentos de depressão no uso de recursos ficando entre 60 e 79,9%. É possível perceber também que no cenário de nuvem híbrida o uso de recursos foi mais brando, podendo-se afirmar que o sistema ficou com parte da infraestrutura disponível durante todo o tempo de simulação.

Figura 6.25: Porcentagem do uso de recursos de nuvem dos experimentos C', D', E' e F'.

A análise dos resultados dos experimentos E' e F', os quais consideraram o algoritmo de intermediação sem QoS também revela um uso de recursos de nuvem entre 40% e 90% no cenário de nuvem privada (experimento E'), com alguns picos acima dos 90% em alguns momentos e uma média de 66,30%. No cenário de nuvem híbrida (experimento F') os valores da porcentagem do uso de recursos flutuam entre 39,0% e 95,0% com alguns picos que chegam a atingir 100% do uso dos recursos de nuvem durante o tempo de simulação.

O diagnóstico para os resultados apresentados pela simulação são que o arranjo fixo de serviços por meio de uma distribuição uniforme causa uma subutilização da infraestrutura de nuvem, dado que a distribuição por igual de serviços não considera a forma como a carga de trabalho de requisições de serviço chega para o ambiente de nuvem.

A Figura 6.26 possui os gráficos de resultados do uso de recursos referente aos experimentos C'', D'', E'' e F''. De imediato é possível notar uma sensível diferença em relação ao uso de recursos de nuvem dos experimentos C', D', E' e F' (conforme Figura 6.25).

Nos experimentos C'' e D'' (Figura 6.26), os quais foram executados considerando o algoritmo de intermediação sem QoS, é possível notar que no experimento C'', por conta do uso do cenário com nuvem privada, o algoritmo de escalonamento pode garantir o uso de 100,0% dos recursos da infraestrutura de nuvem durante todo o tempo de simulação.

Já no experimento D'', considerando cenário de nuvem híbrida, o uso de recursos se manteve entre 92,0% e 100,0%, mantendo-se boa parte do tempo, próximo ao uso máximo de recursos.

Ainda referente ao resultado do experimento D'' é importante justificar que o motivo pelo qual existem momentos cujo uso de recursos foi inferior a 100% com uso de intermediação com QoS é porque a carga de trabalho imposta pelos clientes não foi suficiente para usar 100,0% dos recursos durante o tempo de simulação.

Nos experimentos E'' e F'', os quais consideraram o algoritmo de intermediação sem QoS, há notória similaridade nos resultados, ficando os valores referentes à porcentagem do uso de recursos de nuvem entre 92,0% e 100,0% em ambos os experimentos.

A média de uso de recursos no experimento E'' foi de 97,91% e no experimento F'' a média foi de 99,05%. Embora não ocorra descarte nos experimentos de intermediação sem QoS, a alocação dinâmica de serviços acabou ocasionando um pouco de ociosidade na infraestrutura, pois, acabou por acomodar a carga imposta sobre o ambiente de computação em nuvem com sobra de recursos.

Figura 6.26: Porcentagem do uso de recursos de nuvem dos experimentos C'', D'', E'' e F''.

Como análise final dos resultados do uso de recursos, comparando os valores obtidos nas figuras Figura 6.25 e Figura 6.26 destaca-se a vantagem obtida pela alocação dinâmica de serviços, a qual foi executada utilizada nos experimentos C'', D'', E'' e F''. Essa confrontação

de resultados firma ainda mais o argumento de que efetuar a gestão dinâmica de serviço alocando-os conforme o componente intermediador *cloud broker* percebe a necessidade de demanda é mais interessante do que a escolha de qualquer política para a distribuição dos serviços pela infraestrutura sem que se conheça a carga de trabalho a ser acomodada.

Assim, por meio do exposto, é possível prover um parecer final sobre o cenário experimental 3, o qual demonstra que o procedimento de alocação dinâmica de serviços que foi implementado pela arquitetura QBroker pode beneficiar e influenciar positivamente o desempenho da variável de tempo de resposta e das demais variáveis de resposta nos cenários de simulação com alocação estática de serviços e intermediação com QoS e nos cenários com alocação de dinâmica de serviços e intermediação sem QoS.

Além do exposto na afirmativa anterior, os resultados do presente cenário evidenciam que o recurso de alocação dinâmica de serviços, presente no modo de operação de intermediação de serviços do QBroker, constitui um diferencial importante para o escalonamento de tarefas para sistemas de computação em nuvem, pois, como foi possível notar em alguns dos experimentos, a alocação dinâmica acaba por causar um efeito de “adaptação” da disponibilidade de serviços conforme a carga de trabalho imposta ao ambiente computacional de nuvem resultando em ganhos de desempenho e melhor balanceamento de carga.

6.4 Considerações Finais

Neste capítulo foi apresentada a metodologia aplicada para a avaliação de desempenho da arquitetura QBroker. O ambiente de testes modelado com simulador CloudSim foi apresentado em detalhes e o planejamento de experimental dividido em três cenários também foi explanado. A análise dos resultados experimentais dos cenários cujos enfoques foram a evidenciação de qualidade de serviço, de diferenciação de serviços e de alocação de serviço foi discutida.

A partir dos resultados alcançados nos experimentos é possível ressaltar que com o QBroker e seu modo de operação de intermediação foi possível oferecer qualidade de serviço em todos os cenários de simulação considerados, evidenciando sua característica de tratamento de requisitos de QoS. Outro elemento de destaque foi a possibilidade de trabalhar com coleções de diferente número de serviços em cada VM dos provedores de nuvem, assim, também foi possível comprovar que há influência no desempenho das variáveis de resposta por causa do número de serviços em cada unidade de máquina virtual. Como terceiro item que cabe destacar,

a alocação dinâmica de serviços foi uma característica importante para o desempenho do QBroker. Ao ser comparada com experimentos de alocação estática, o procedimento de alocação dinâmica causou um impacto positivo nas variáveis de resposta, tendo em vista que as instâncias de serviços foram sendo realizadas sob demanda e de acordo com o perfil da carga de trabalho. Assim, essas características implementadas no QBroker corroboram com algumas dentre as principais características alvo de implementações de *cloud brokers* que são o balanceamento de carga, o aumento da qualidade dos serviços e o melhor aproveitamento no uso dos recursos de nuvem disponíveis.

Conforme argumentado, foi possível, por meio da avaliação de desempenho da arquitetura QBroker destacar os ganhos obtidos no ambiente de nuvem simulado, por meio de suas características arquiteturais e resultados que revelam a proficuidade de sua atuação como elemento de intermediação e QoS.

O capítulo seguinte da tese apresenta as conclusões finais, as contribuições diretas para a linha de pesquisa em que este trabalho se insere e os trabalhos futuros vislumbrados como continuidade do projeto de pesquisa.

---

7. Conclusões

Esta tese aborda o tópico de *Cloud Brokers*, considerando escalonamento de tarefas para sistemas de nuvem, modos de operação de *cloud brokers*, qualidade de serviço e ambiente de nuvem híbrida como características específicas da abordagem de trabalho.

Por se tratar de atividade crítica para que os sistemas de nuvem entreguem o desempenho necessário, a atividade de *brokerage* tornou-se tópico emergente para o campo de pesquisa de computação em nuvem. Na atividade de mediação de requisições para sistemas de nuvem, as políticas de escalonamento de tarefas de nuvem podem possuir vários objetivos, sendo alguns deles: aumento da qualidade dos serviços, melhor aproveitamento da infraestrutura disponível (tanto em desempenho quanto em economia de energia), balanceamento da carga de trabalho, entre outros.

Na literatura há diversos trabalhos que buscam a obtenção de QoS por meio de algoritmos de escalonamento de tarefas implementados em *Cloud Brokers*. Contudo, não foram encontrados *Cloud Brokers* cuja política de escalonamento considere ao mesmo tempo: atendimento a requisitos de QoS, alocação estática e dinâmica de serviços, emulação de um ambiente de nuvem SOA, processo de mediação que considere o conceito de intermediação de serviços embasado no modelo de referência NIST e ambiente de computação de nuvem híbrida.

O objetivo geral desta tese foi o de analisar, projetar, desenvolver e avaliar uma arquitetura de *Cloud Broker*, considerando elementos de arquiteturas orientadas a serviços e requisitos de qualidade de serviços para um ambiente de computação em nuvem híbrido. Com base nos resultados experimentais, foi possível demonstrar que é possível prover qualidade de serviço e outros aspectos positivos relacionados com a estratégia de alocação das tarefas entre os *datacenters* considerados no ambiente. Demonstrou-se ainda a possibilidade de ganhos no aproveitamento da infraestrutura de recursos de nuvem disponível.

Foram delineados cinco objetivos específicos como parte da metodologia a ser empregada para a confirmação da hipótese proposta de que o uso de um *Cloud Broker* como

elemento de intermediação e QoS pode prover significativo impacto no desempenho e qualidade dos serviços bem como no aproveitamento da infraestrutura disponível no ambiente computacional de nuvem híbrida. As avaliações de desempenho conduzidas durante o trabalho, bem como a implementação da arquitetura QBroker e seu acoplamento na arquitetura de nuvem CloudSim-BEQoS, conduziram este trabalho a alcançar os objetivos específicos propostos.

O primeiro objetivo trata da caracterização de um ambiente de nuvem computacional híbrida, elemento que foi modelado como um *datacenter* de nuvem privada e um *datacenter* de nuvem pública, usando componentes de simulação da API padrão do CloudSim. Ainda sobre o ambiente de nuvem híbrida, estabeleceu-se a regra de negócio fundamental de considerar a nuvem privada como sendo uma infraestrutura da própria organização isenta de custos de locação de recursos de nuvem, assim, a prioridade de escalonamento sempre foi dada em primeira instância para a nuvem privada. A modelagem da infraestrutura dos *datacenters* bem como a normas de escalonamento do ambiente de nuvem híbrida implementados dão por concluído este objetivo.

O segundo e terceiro objetivos específicos, isto é, a caracterização de uma arquitetura orientada a serviços e a elaboração dos mecanismos de alocação e desalocação de serviços serão explanados em conjunto, pois, suas implementações estão associadas. Conforme é possível perceber pelas avaliações de desempenho apresentadas da arquitetura QBroker, as tabelas de planejamento de experimentos assinalaram as quantidades de serviços considerados nos cenários de simulação. Tal implementação foi possível por meio do acréscimo de identificadores numéricos de serviço no formato de listas de serviços em cada VM. Além disso, para a instanciação dos serviços, tanto no início da simulação quanto em tempo de simulação e sob demanda, foram utilizadas rotinas de alocação de desalocação de serviços desenvolvidas especialmente para o QBroker e arquitetura CloudSim-BEQoS. Logo, por meio das justificativas apresentadas e dos resultados apresentados no trabalho esses objetivos também foram completados.

O quarto objetivo específico foi o desenvolvimento de um mecanismo de intermediação capaz de prover qualidade de serviço considerando pelo menos um parâmetro de QoS. Conforme foi mencionado anteriormente, a arquitetura do QBroker contemplou o modo de operação de intermediação de serviços (com e sem QoS), o qual considera o parâmetro de tempo máximo de execução de serviço por meio do qual foi possível negociar o atendimento ou a recusa na execução de tarefas nos experimentos que consideraram, principalmente, a intermediação com QoS. É oportuno mencionar que o referido modo de operação foi

desenvolvido com base no conceito de intermediação do modelo de referência do NIST e também considerou a regra de negócio de priorização da execução de tarefas no ambiente de nuvem privada, no caso dos experimentos com nuvem híbrida. Dado que os resultados experimentais assinalaram considerável impacto no desempenho em função política de escalonamento de intermediação, dá-se por concluído o objetivo específico atual.

Como último objetivo específico, foi levantado o requisito da emulação de clientes virtuais para o envio das cargas de trabalho durante o tempo de simulação ao *Cloud Broker*. Conforme apresentado no capítulo da Arquitetura CloudSim-BEQoS, foi desenvolvida a classe de objetos Cliente para a emulação dos clientes virtuais e seus comportamentos em tempo de simulação. Por meio dessa implementação é possível efetuar diferentes modos de submissão de requisições tendo sido considerado para os experimentos desta tese o modo de submissão de requisições em tempo de simulação. A classe Cliente foi de grande valia no enriquecimento do ambiente de nuvem modelado no simulador e causou um efeito muito mais realístico no que tange ao envio de requisições durante a simulação, tornando as cargas de trabalho sintéticas bastante próximas ao que se espera de um comportamento real. Dessa forma, se dão por completados todos os objetivos traçados no presente trabalho.

Durante a discussão dos resultados experimentais desta tese, foi possível demonstrar a proficuidade do modo de operação do QBroker em diversas situações e cenários com caracterizações diferentes. Em todos os casos de experimentos com algoritmo de intermediação com QoS, obteve-se resultados satisfatórios e interessantes para os dois componentes que se encontram nas extremidades do ecossistema de nuvem, isto é, os clientes e os provedores. Os clientes puderam obter a execução de suas tarefas dentro do *deadline* requisitado e os provedores puderam gerenciar melhor sua infraestrutura na acomodação da carga e na total utilização dos recursos, efetuando aproveitamento máximo dos recursos computacionais de que dispunham. Com base nesses argumentos, é possível deixar como legado desta tese que em ambientes de nuvem híbrida com regras de negócio específicas para a mediação das tarefas, é possível efetuar um processo de gestão de recursos e serviços de nuvem de forma otimizada com base na atividade de *brokerage*.

Dado que os objetivos gerais e específicos do presente trabalho foram alcançados e os resultados experimentais corroboram e validam a arquitetura QBroker como componente de mediação e provimento de escalonamento de tarefas com qualidade de serviço, pode-se considerar então correta a seguinte afirmação: “É possível utilizar um *Cloud Broker* com uma

política de intermediação de serviços, considerando uma arquitetura orientada a serviços e um ambiente de nuvem computacional híbrida, para obtenção de qualidade em serviços“.

7.1 Contribuições

O projeto de pesquisa referente a esta tese gerou contribuições diretamente ligadas com a hipótese proposta e também contribuições para o campo de pesquisa em que se insere este trabalho.

Dentre as principais contribuições resultantes desta tese tem-se os seguintes itens:

- **Uma Nova Arquitetura de Cloud Broker com tratativas de QoS:** esta foi a principal contribuição provida pela arquitetura QBroker. Os elementos diferenciais empregados no algoritmo de intermediação de serviços (tratativas de QoS, alocação estática e dinâmica de serviços, escalonamento para o ambiente de nuvem híbrida) permitiram ganhos em desempenho nas variáveis de resposta consideradas em sua avaliação de desempenho.
- **Uma arquitetura de Cloud Broker Aberta:** como foi projetado a partir das classes da API padrão do simulador CloudSim, o QBroker preserva a característica de herança para com sua superclasse na API do simulador e admite a criação de subclasses a partir de sua implementação. Essa característica representa importante contribuição, pois permite o emprego do QBroker em novas implementações que podem efetuar a sobrescrita de seus métodos, testar e avaliar novos modos de operação ou mesmo criar novas versões dos modos de operação já existentes. Tal contribuição pode ser levada em consideração como arquitetura aberta para novas pesquisas e também como uma contribuição direta para a evolução do CloudSim *Simulation Toolkit*.
- **Avaliação de três cenários experimentais que consideram diferentes características da implementação da arquitetura QBroker:** Com o planejamento de experimentos executado foi possível avaliar o comportamento das variáveis de resposta selecionadas, comparando o desempenho do QBroker com o algoritmo Round Robin e com experimentos nos quais as características do QBroker foram avaliadas.
- **Desenvolvimento de uma entidade para emulação de clientes virtuais:** Um componente que foi desenvolvido para emular a presença de clientes no ambiente de nuvem foi a Classe Cliente da arquitetura CloudSim-BEQoS.

Conforme já explanado no capítulo sobre a arquitetura BEQoS, essa entidade de simulação oferece, entre as suas muitas funcionalidades, a caracterização da carga de trabalho segundo três modos diferentes de submissão e ainda permite a emulação de cenários com rajadas. Essa implementação complementar foi crucial para a obtenção dos efeitos vistos nos gráficos de consumo de recursos, pois, tornou os valores dos resultados finais mais realísticos. Assim, a classe Cliente também se constitui de uma contribuição paralela ao desenvolvimento do QBroker.

- **Projeto e desenvolvimento de uma arquitetura que admite a avaliação de desempenho com política de escalonamento de tarefas de intermediação de serviços:** o QBroker é o componente de mediação da arquitetura de nuvem CloudSim-BEQoS. Assim, tornou-se o elemento central de negociação entre os clientes virtuais e provedores de nuvem, possibilitando, por meio de sua arquitetura de *cloud broker* a avaliação de muitos cenários possíveis de serem construídos com a coletânea de classes que formam a arquitetura BEQoS. Além disso, provê um mecanismo de intermediação eficiente e que emprega vários elementos para o provisionamento de qualidade em serviços.

7.2 Produção Científica

Por meio da implementação realizada no desenvolvimento da arquitetura do QBroker, da avaliação de desempenho conduzida e dos resultados experimentais obtidos, foram escritos e publicados artigos cujas informações de publicação são mencionadas a seguir:

- Artigo publicado em conferência internacional:
  - **Conferência:** *Sixth International Conference on Computer Science, Engineering and Applications (CCSEA 2016)*, Dubai, UAE, January 23~24, 2016.
  - **Título do artigo:** “A Cloud Broker Approach with QOS Attendance and SOA for Hybrid Cloud Computing Environments”
  - **Autores:** Mário Henrique de Souza Pardo, Adriana Molina Centurion, Paulo Sérgio Franco Eustáquio, Regina Helena Carlucci Santana, Sarita Mazzini Bruschi and Marcos José Santana, *University of São Paulo (USP), Brazil*

- **Conteúdo:** Apresentação da arquitetura QBroker e avaliação de desempenho e discussão de resultados referentes ao modo de operação de intermediação do *cloud broker* proposto.
- **Anais da conferência:** [\[LINK\]](#)
- **Link para o artigo:** [\[LINK\]](#)
- **DOI:** 10.5121/csit.2016.60207
- **Data de Publicação:** 23/01/2016
- Artigo publicado em *Journal* internacional:
  - **Períodico:** *Journal PlosOne*
  - **Título do Artigo:** “*Evaluating the Influence of the Client Behavior in Cloud Computing*”.
  - **Autores:** Mário Henrique de Souza Pardo, Adriana Molina Centurion, Paulo Sérgio Franco Eustáquio, Regina Helena Carlucci Santana, Sarita Mazzini Bruschi, Marcos José Santana
  - **Conteúdo:** Discussão a respeito da nova entidade cliente implementada na arquitetura CloudSim-BEQoS e a proposição de uma avaliação de desempenho para investigação da influência da camada cliente em cenários de nuvem.
  - **Link para o artigo:** [\[LINK\]](#)
  - **DOI:** <http://dx.doi.org/10.1371/journal.pone.0158291>
  - **Data de publicação:** 21/07/2016

7.3 Trabalhos Futuros

Este trabalho foi idealizado, desde o seu início, para deixar um legado a partir de suas contribuições enquanto implementação feita em *software*. Como já mencionado, tanto a arquitetura do QBroker quanto a arquitetura CloudSim-BEQoS deverão ser, em breve, licenciados como *software livre* e publicados num repositório para projetos de *software* (como GitHub, por exemplo).

Todavia, com a experiência de implementação oferecida pelo projeto de pesquisa de doutorado, muitas ideias, críticas e sugestões foram sendo coletadas ao longo do período de desenvolvimento e de testes da arquitetura. Logo, são vislumbrados alguns itens como trabalhos futuros:

- **Implementação de modos de operação:** conforme citado durante a tese, o QBroker foi desenvolvido com todas as primitivas para ser compatível com os três modos básicos de operação descritos na arquitetura do NIST. Assim, é parte dos planos futuros a finalização das implementações para os modos de operação de agregação e arbitragem e a avaliação de desempenho dos mesmos em cenários com clientes virtuais trabalhando em modo de submissão de grupo de requisições. Tal implementação e estudo poderá trazer novas descobertas sobre o impacto dessas políticas de escalonamento no desempenho de ambientes de computação em nuvem.
- **Negociação com múltiplos requisitos de QoS:** essa futura implementação também será realizada por meio da consideração de listas de parâmetros de QoS cujos valores poderão ser analisados e mediados pelo QBroker por meio dos modos de operação com a possibilidade de uso de técnicas de otimização com múltiplos critérios. A ideia é não limitar o cenário quanto ao número de parâmetros de QoS aceitos nas requisições e oferecer um ambiente negocial mais complexo e rico em possibilidades para a execução das tarefas de usuários. Por meio dessa implementação será possível verificar como é possível prover a melhor sinergia entre as exigências dos clientes quanto a qualidade de serviços e o QoS real dos recursos de nuvem dos provedores.
- **Módulo de renegociação de tarefas não atendidas:** em várias ocasiões do fluxo de atividades do algoritmo de intermediação do QBroker é possível verificar que, nas situações onde o QoS é considerado um item obrigatório, existe o descarte de requisições. Embora essa política mantenha os níveis de desempenho do sistema sempre dentro da expectativa do usuário, por outro lado se tem uma grande quantia das requisições que não são executadas. Além disso, em sistemas de nuvem do mundo real, há sempre ocasiões em que os contratos estabelecidos de nível de serviços permitem uma renegociação dos parâmetros de QoS até certos limites. Assim, uma implementação futura também prospectada é a elaboração de um módulo de renegociação que permita o rebaixamento dos valores padrão dos requisitos de QoS de forma que se possa encontrar um recurso de nuvem que possa atender a requisição evitando, tanto quanto possível, o descarte. Essa implementação levará a um estudo que poderá revelar questões relacionadas ao aumento da quantidade de

tarefas executadas, verificação do percentual de quebra de contratos, observação do aumento no tempo de resposta médio de tarefas rebaixadas, impacto nos custos das execuções de tarefas, entre muitos outros.

- **Modelo de Custo:** elemento que é constantemente considerado em trabalhos de computação em nuvem que avaliam o desempenho desses sistemas é o modelo de custo. Desta forma, a consideração de múltiplos parâmetros de QoS (que já foi citada como implementação futura) deverá levar ao caminho natural da consideração de restrições de custo do lado cliente. Para que o QBroker possa estimar o custo da execução de serviços e de transporte via rede, há a necessidade de consultar esses elementos sob demanda como informação de estado do ambiente de nuvem. Além disso, será necessário implementar um módulo de gerenciamento de custos que permita, inclusive, a implementação de diferentes políticas de custos, pois, se houver essa possibilidade em aberto o número de testes e avaliações de desempenho que poderão ser efetuados será numeroso e proveitoso. Essa implementação abrirá caminho para o estudo do impacto de diferentes modelos de custo nos preços finais pagos por clientes na execução de suas tarefas e o modelo de custos poderá influenciar a forma como os provedores disponibilizam seus recursos, visando ampliar os lucros.
- **Caracterização de serviços:** os serviços atualmente considerados no ambiente da arquitetura CloudSim-BEQoS e, conseqüente, intermediados pelo QBroker, distinguem-se por um identificador (*service id*). Uma perspectiva futura é a de maturar a abstração dos serviços, criando uma diferenciação de características de serviço que possibilitem um melhor critério de escolha a respeito do mesmo serviço instanciado em VMs diferentes. Por exemplo, seria bastante interessante que cada serviço se caracterizasse pela alocação de um número determinado de MIPS que não fosse a mesma capacidade em MIPS da CPU virtual da VM. Essa implementação de trabalho futuro poderá levar a estudos para comprovar se a adoção do mesmo serviço com diferentes características traria ou não retornos positivos em desempenho e balanceamento de carga.
- **Implementação de clientes móveis:** os clientes virtuais em sua versão atual da arquitetura CloudSim-BEQoS são de natureza estática, isto é, eles são instanciados no cenário de simulação e permanecem até o final do tempo estipulado para o experimento ou finalizando quando termina a sua quantidade de requisições (modo de submissão de vetor de requisições). Contudo, para

aumentar o realismo do cenário simulado, seria interessante que alguns dos clientes também “desaparecessem” e “reaparecem” em algum instante da simulação, pois, essa é uma situação bastante rotineira no mundo real. A implementação dessa melhoria como trabalho futuro não é diretamente relacionada ao QBroker, todavia, permitirá a elaboração de cenários de simulação em que o comportamento de clientes móveis (podem aparecer numa dada posição do grafo de rede e depois em um novo nó) exigirá da política de escalonamento novas tratativas que considerem o tempo de permanência do cliente, visando a não execução de tarefas de clientes que não estão mais em contato com o *broker*.

- **Variação de configurações de infraestrutura e escalabilidade do *broker*:** uma estratégia que poderá revelar novas informações a respeito do ambiente de computação em nuvem híbrida seria o emprego de configurações variáveis para os elementos componentes da infraestrutura de nuvem, isto é, dos *hosts* e das VMs, pois, dessa forma, a heterogeneidade do ambiente seria ainda mais completo, pois, além dos fatores heterogêneos considerados para os clientes e carga de trabalho (característica já implementada nos experimentos desta tese) seria possível obter heterogeneidade na camada de gestão de infraestrutura de nuvem. Além disso, ainda pensando na infraestrutura do ambiente de nuvem híbrida, seria importante planejar um conjunto de experimentos nos quais fosse avaliada a escalabilidade do QBroker, de modo que se possa obter informações e conclusões a respeito da robustez da arquitetura proposta em cenários de testes de estresse para a camada de intermediação de requisições para serviços de nuvem.

Referências

AAZAM, M. Advance Resource Reservation and QoS Based Refunding in Cloud Federation. p. 139–143, 2014.

AAZAM, M.; HUH, E.-N. Broker as a Service (BaaS) Pricing and Resource Estimation Model. *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, p. 463–468, dez. 2014. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7037703>>. Acesso em: 10 jul. 2015.

ACHAR, R.; THILAGAM, P. S. A broker based approach for cloud provider selection. set. 2014, [S.l.]: IEEE Computer Society, set. 2014. p. 1252–1257. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6968439>>.

ADHianto, L. *et al.* HPCTOOLKIT: Tools for performance analysis of optimized parallel programs. *Concurrency Computation Practice and Experience*, v. 22, n. 6, p. 685–701, 2010.

ALAM, F. M. *et al.* A discrete event simulation tool for performance management of web-based application systems. v. 6, n. 1, p. 21–32, 2012. Disponível em: <<http://dx.doi.org/10.1057/jos.2011.8>>.

AL-ROOMI, M. *et al.* Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, v. 6, n. 5, p. 93–106, 31 out. 2013. Disponível em: <http://www.sersc.org/journals/IJGDC/vol6_no5/9.pdf>.

AMATO, A.; DI MARTINO, B.; VENTICINQUE, S. Cloud brokering as a service. *Proceedings - 2013 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013*, p. 9–16, 2013.

AMAZON. *Amazon Web Services*. Disponível em: <<http://aws.amazon.com/pt/ec2/>>.

AVETISYAN, A. I. *et al.* Open Cirrus: A Global Cloud Computing Testbed. *Computer*, v. 43, n. 4, p. 35–43, 2010.

BABU, L. D. D.; KRISHNA, P. V. An execution environment oriented approach for scheduling dependent tasks of cloud computing workflows. *International Journal of Cloud Computing*, PMID: 62275, v. 3, n. 2, p. 209–224, 2014. Disponível em: <<http://www.inderscienceonline.com/doi/abs/10.1504/IJCC.2014.062275>>.

BARKER, A.; VARGHESE, B.; THAI, L. Cloud Services Brokerage: A Survey and Research Roadmap. 2015. Disponível em: <<http://arxiv.org/abs/1506.00485>>.

BARRETT, E.; DUGGAN, J. A Learning Architecture for Scheduling Workflow Applications in the Cloud. p. 83–90, 2011.

BATISTA, B. G.; SANTANA, R. H. C. *Modelos de negócio para ambientes de computação em nuvem que consideram atributos de qos relacionados a desempenho e a segurança*. 2015. 166 f. Universidade de São Paulo, 2015.

BELALEM, G.; TAYEB, F. Z.; ZAOUI, W. Approaches to improve the resources management in the simulator CloudSim. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 6377 LNCS, p. 189–196, 2010. Disponível em: <<http://www.scopus.com/inward/record.url?eid=2-s2.0-78649497840&partnerID=40&md5=9448331e31c9d377fe5e5e17b8fd927d>>.

BELOGLAZOV, A. *et al.* *CloudSim: A Framework For Modeling And Simulation Of Cloud Computing Infrastructures And Services*. Disponível em: <<http://www.cloudbus.org/cloudsim/>>. Acesso em: 1 jan. 2016.

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, p. 755–768, 2012. Disponível em: <<http://dx.doi.org/10.1016/j.future.2011.04.017>>.

BHATT, T. P.; PROF, A.; PATEL, P. J. Survey On Virtualization With Xen Hypervisor. v. 1, n. 8, p. 1–8, 2012.

BITTENCOURT, L. F.; MADEIRA, E. R. M. HCOC: a cost optimization algorithm for workflow scheduling in hybrid clouds. *Journal of Internet Services and Applications*, v. 2, n. 3, p. 207–227, 2011. Disponível em: <<http://dx.doi.org/10.1007/s13174-011-0032-0>>.

BOHN, R. B. *et al.* NIST cloud computing reference architecture. *Proceedings - 2011*

IEEE World Congress on Services, SERVICES 2011, p. 594–596, 2011.

BUMPUS, W. NIST Cloud Computing Standards Roadmap. *NIST Cloud Computing Standards*, p. 1–3, 2013.

BUYYA, R. *et al.* Cloud computing and emerging IT platforms : Vision , hype , and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, p. 1–18, 2009. Disponível em: <<http://dx.doi.org/10.1016/j.future.2008.12.001>>.

BUYYA, R.; RANJAN, R.; CALHEIROS, R. N. InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services. 2010, [S.l: s.n.], 2010. p. 13–31.

BUYYA, R.; TOOSI, A. N.; CALHEIROS, R. N. Interconnected Cloud Computing Environments. *ACM Computing Surveys*, v. 47, n. 212, p. 1–47, 2014. Disponível em: <<http://0-dl.acm.org.cataleg.uoc.edu/citation.cfm?id=2620784.2593512>>.

CALHEIROS, R. N. *et al.* CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. n. August 2010, p. 23–50, 2011.

CALHEIROS, R. N. *et al.* CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *arXiv preprint arXiv:0903.2525*, p. 9, 2009. Disponível em: <<http://arxiv.org/abs/0903.2525>>.

CALHEIROS, R. N. *et al.* EMUSIM : an integrated emulation and simulation environment for modeling , evaluation , and validation of performance of Cloud computing applications. n. April 2012, p. 595–612, 2013.

CATARINA, S. *CLOUD COMPUTING 2015 - The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization*. [S.l: s.n.], 2015.

CENTURION, A. M.; SANTANA, R. H. C. *Impacto das rajadas no desempenho de serviços executados em ambientes em nuvens*. 2015. 201 f. Universidade de São Paulo, 2015. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-10092015-144229/pt-br.php>>.

CHANDRASEKAR, S. A Review of Literature on Cloud Brokerage Services. *International Journal of Computer Science and Business Informatics*, v. 10, n. 1, p. 25–40, 2014. Disponível em: <<http://ijcsbi.org/index.php/ijcsbi/article/view/312>>.

CHANG, V.; WALTERS, R. J.; WILLS, G. The development that leads to the Cloud Computing Business Framework. *International Journal of Information Management*, v. 33, n. 3, p. 524–538, 2013. Disponível em: <<http://dx.doi.org/10.1016/j.ijinfomgt.2013.01.005>>.

CHEN, H. *et al.* User-priority guided min-min scheduling algorithm for load balancing in cloud computing. *2013 National Conference on Parallel Computing Technologies, PARCOMPTECH 2013*, 2013.

CHEN, T. Y. *et al.* LaSA: A locality-aware scheduling algorithm for Hadoop-MapReduce resource assignment. *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, p. 342–346, 2013.

CHEN, W.; REY, M. WorkflowSim : A Toolkit for Simulating Scientific Workflows in Distributed Environments. 2012.

CHUNG, L. *et al.* A goal-oriented simulation approach for obtaining good private cloud-based system architectures. *Journal of Systems and Software*, v. 86, p. 2242–2262, 2013. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0164121212002865>\nfiles/1620/Chung et al. - 2013 - A goal-oriented simulation approach for obtaining .pdf\nfiles/1619/S0164121212002865.html>.

CHUNLIN, L.; LAYUAN, L. Optimal scheduling across public and private clouds in complex hybrid cloud environment. *Information Systems Frontiers*, 2015. Disponível em: <<http://link.springer.com/10.1007/s10796-015-9581-2>>.

CLAMP, P.; CARTLIDGE, J. Pricing the cloud: an adaptive brokerage for cloud computing. *SIMUL 2013, The Fifth International Conference on Advances in System Simulation*, n. VM, p. 113–121, 2013. Disponível em: <http://www.thinkmind.org/index.php?view=article&articleid=simul_2013_5_20_50079>.

CLOUD, S. *et al.* Social-optimized Win-win Resource Allocation for. n. Vmm, p. 251–258, 2011.

COMPUT, J. P. D. *et al.* Online optimization for scheduling preemptable tasks on IaaS cloud systems. *J. Parallel Distrib. Comput.*, v. 72, n. 5, p. 666–677, 2012. Disponível em: <<http://dx.doi.org/10.1016/j.jpdc.2012.02.002>>.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed Systems: Concepts and Design*. [S.l.: s.n.], 2012. v. 4. Disponível em: <<http://www.amazon.com/dp/0321263545>>.

COUTINHO, E. F. *et al.* Elasticity in cloud computing: a survey. *Annales des Telecommunications/Annals of Telecommunications*, p. 289–309, 2014.

CRAGO, S. *et al.* Heterogeneous cloud computing. 2011.

DE ALFONSO, C. *et al.* Infrastructure deployment over the cloud. *Proceedings - 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011*, p. 517–521, 2011.

DELDARI, A. *et al.* A Clustering Approach to Scientific Workflow Scheduling on the Cloud with Deadline and Cost Constraints. v. 46, n. 1, p. 19–29, 2014.

DEVI, R. K.; SUJAN, S. A Survey on Application of Cloudsim Toolkit in Cloud Computing. v. 3, n. 6, p. 13146–13153, 2014.

DEVIPRIYA, S.; RAMESH, C. Improved max-min heuristic model for task scheduling in cloud. *Proceedings of the 2013 International Conference on Green Computing, Communication and Conservation of Energy, ICGCE 2013*, p. 883–888, 2013.

DHARA, K. M. A Survey Paper on Service Oriented Architecture Approach and Modern Web Services. 2015.

DUAN, R.; PRODAN, R.; LI, X. Multi-Objective Game Theoretic Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds. *IEEE Transactions on Cloud Computing*, v. 2, n. 1, p. 29–42, 2014.

DUMMIES.COM. *Comparing Public, Private, and Hybrid Cloud Computing Options*. Disponível em: <<http://www.dummies.com/how-to/content/comparing-public-private-and-hybrid-cloud-computin.html>>. Acesso em: 2 fev. 2016.

DUTTA, D. A Genetic – Algorithm Approach to Cost-Based Multi-QoS Job Scheduling in Cloud Computing Environment. n. Icwet, p. 422–427, 2011.

EVOY, G. V. M.; SCHULZE, B.; GARCIA, E. L. M. Performance and deployment evaluation of a parallel application on a private Cloud ‡. n. January, p. 2048–2062, 2011.

FANIYI, F.; BAHSOON, R. A Systematic Review of Service Level Management in Cloud. v. 1, n. 1, p. 1–30, 2015.

FARD, H. M. *et al.* A Bi-Criteria Truthful Mechanism for Scheduling of Workflows in Clouds. 2011.

- FAZIO, M. *et al.* Virtual Resource Management Based On Software Transactional Memory. 2011.
- FINGAR, P. Call Your Broker : The Rise of Cloud Service Brokerages. p. 1–4, 2012.
- GAO, Y. *et al.* A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, v. 79, n. 8, p. 1230–1242, 2013. Disponível em: <<http://dx.doi.org/10.1016/j.jcss.2013.02.004>>.
- GARG, S. K.; BUYYA, R. NetworkCloudSim: Modelling parallel applications in cloud simulations. 2011, [S.l: s.n.], 2011. p. 105–113.
- GARG, S. K.; BUYYA, R. NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations. *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, n. Vm, p. 105–113, dez. 2011. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6123487>>. Acesso em: 2 maio 2015.
- GENEZ, T. A. L.; BITTENCOURT, L. F.; MADEIRA, E. R. M. Workflow Scheduling for SaaS / PaaS Cloud Providers Considering Two SLA Levels. p. 906–912, 2012.
- GILL, S. S. Autonomic Cloud Computing: Research Perspective. *arXiv preprint arXiv:1507.01546*, 2015. Disponível em: <<http://arxiv.org/abs/1507.01546>>.
- GOOGLE. *Google Application Engine (GAE)*. Disponível em: <<http://cloud.google.com/appengine>>. Acesso em: 2 maio 2016a.
- GOOGLE. *Google Cloud Platform*. Disponível em: <<http://cloud.google.com/products/>>.
- GROSSMAN, R. *et al.* The Open Cloud Testbed: A Wide Area Testbed for Cloud Computing Utilizing High Performance Network Services. 27 jul. 2009. Disponível em: <<http://arxiv.org/abs/0907.4810>>.
- HE, T. *et al.* Scheduling parallel tasks onto opportunistically available cloud resources. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, p. 180–187, 2012.
- HUA, A.; CHANG, S.-L. Algorithms and Architectures for Parallel Processing. *Saudi Med J*, v. 33, p. 3–8, 2012.
- HUSSEIN, J.; MOREAU, L.; SASSONE, V. Trust Management IX. *IFIP*

International Conference on Trust Management, v. 454, p. 109–125, 2015. Disponível em: <http://link.springer.com/chapter/10.1007/978-3-319-18491-3_8/fulltext.html>.

HWANG, E.; KIM, K. H. Minimizing Cost of Virtual Machines for Deadline-Constrained MapReduce Applications in the Cloud. *2012 ACM/IEEE 13th International Conference on Grid Computing*, p. 130–138, set. 2012. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6319163>>. Acesso em: 18 maio 2015.

JADEJA, Y.; MODI, K. Cloud computing - Concepts, architecture and challenges. *2012 International Conference on Computing, Electronics and Electrical Technologies, ICCEET 2012*, p. 877–880, 2012.

JAIN, P. Study and Comparison of CloudSim Simulators in the Cloud Computing. v. 1, n. 4, p. 111–115, 2013.

JAIN, R. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. [S.l.: s.n.], 1991. v. 491.

KALLONIATIS, C.; MOURATIDIS, H.; ISLAM, S. Evaluating cloud deployment scenarios based on security and privacy requirements. *Requirements Engineering*, v. 18, n. 4, p. 299–319, 2013.

KANG, D.-K. *et al.* Cost adaptive workflow scheduling in cloud computing. *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, p. 1–8, 2014.

KARNOUSKOS, S. *et al.* A SOA-based architecture for empowering future collaborative cloud-based industrial automation. *IECON Proceedings (Industrial Electronics Conference)*, p. 5766–5772, 2012.

KAVIS, M. J. *Architecting the cloud: design decisions for cloud computing service models (SaaS, PaaS, and IaaS)*. [S.l.]: John Wiley & Sons, 2014.

KHAN, I.; REHMAN, H. U.; ANWAR, Z. Design and deployment of a trusted Eucalyptus cloud. *Proceedings - 2011 IEEE 4th International Conference on Cloud Computing, CLOUD 2011*, p. 380–387, 2011.

KHANNA, P.; JAIN, S. Cloud Broker Definition Differential : Gartner Versus NIST and New Models. v. 2, p. 127–132, 2015.

- KIANI, S. L. *et al.* Towards Context Caches in the Clouds. 2011.
- KLANÈNIK, T.; JERMAN, B. Context-Aware Information Broker for Cloud Computing. n. January, 2010.
- KLIAZOVICH, D. *et al.* E-STAB: Energy-efficient scheduling for cloud computing applications with traffic load balancing. *Proceedings - 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, GreenCom-iThings-CPSCom 2013*, p. 7–13, 2013.
- KLIAZOVICH, D.; BOUVRY, P. GreenCloud : a packet-level simulator of energy-aware cloud computing data centers. p. 1263–1283, 2012.
- LALEH, T. *et al.* Context-Aware Cloud Service Brokerage: A Solution to the Problem of Data Integration Among SaaS Providers. *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*, p. 46–55, 2015. Disponível em: <<http://doi.acm.org/10.1145/2790798.2790804>>.
- LELOGLU, E.; AYAV, T.; ASLAN, B. G. A review of cloud deployment models for e-learning systems. *Proceedings of the International Conference on Dependable Systems and Networks*, p. 4–5, 2013.
- LIM, S. *et al.* MDCSim : A Multi-tier Data Center Simulation Platform. 2009.
- LIU, F. *et al.* *NIST Cloud Computing Reference Architecture: Recommendations of the National Institute of Standards and Technology (Special Publication 500-292)*. USA: CreateSpace Independent Publishing Platform, 2012.
- LIU, X. *et al.* Data-driven composition for service-oriented situational web applications. *IEEE Transactions on Services Computing*, v. 8, n. 1, p. 2–16, 2015.
- LU, X.; GU, Z. A load-adaptive cloud resource scheduling model based on ant colony algorithm. *2011 IEEE International Conference on Cloud Computing and Intelligence Systems*, p. 296–300, 2011.
- MAHESHWARI, K. *et al.* Workflow performance improvement using model-based scheduling over multiple clusters and clouds. *Future Generation Computer Systems*, v. 54, p. 206–218, 2014. Disponível em: <<http://dx.doi.org/10.1016/j.future.2015.03.017>>.
- MALAWSKI, M. *et al.* Scheduling multilevel deadline-constrained scientific workflows on clouds based on cost optimization. *Scientific Programming*, v. 2015, 2015.

MALLAYYA, D.; RAMACHANDRAN, B.; VISWANATHAN, S. An Automatic Web Service Composition Framework Using QoS-Based Web Service Ranking Algorithm. *Scientific World Journal*, v. 2015, 2015.

MALLICK, S.; KUSHWAHA, D. S. An Efficient Web Service Discovery Architecture for Static and Mobile Environments. *GSTF Journal on Computing (JoC)*, v. 3, n. 1, p. 130, 2013.

MASTELIC, T. *et al.* Cloud Computing: Survey on Energy Efficiency. *ACM Computing Surveys*, v. 47, n. 2, p. 1–36, 2014. Disponível em: <https://www.bsi.bund.de/DE/Themen/CloudComputing/CloudComputing_node.html\n<http://dl.acm.org/citation.cfm?doid=2658850.2656204>>.

MICROSOFT. Microsoft Windows Azure Platform. 2016.

MOHTASHIM, S. *et al.* Limitations of Service Oriented Architecture and its Combination with Cloud Computing. v. 8, n. 1, p. 7–13, 2015.

NANDHINI, B. K. B. P. J. M. Platform Autonomous Custom Scalable Service using Service Oriented Cloud Computing Architecture. v. 2, n. 2, p. 1467–1471, 2012.

NGUYEN, T. *et al.* A framework for dynamic resource provisioning and adaptation in IaaS clouds. 2011.

NITHYA, G.; JAYAPRATHA, G. A Multi-agent Brokering Approach and Jumper Firefly Algorithm for Job Scheduling in Cloud Computing. *2014 International Conference on Intelligent Computing Applications*, p. 52–58, mar. 2014. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6965010>>. Acesso em: 8 jul. 2015.

NUTHULA, V.; CHALLA, N. R. Cloudifying apps - A study of design and architectural considerations for developing cloudenabled applications with case study. *2014 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2014*, 2015.

ORACLE. *Oracle Cloud Platform*. Disponível em: <<https://cloud.oracle.com/home>>. Acesso em: 1 jan. 2016.

PARDESI, V.; KHAMPARIA, A.; BAGDE, N. K. A secure framework in brokerage of heterogeneous cloud environment for multiple cloud providers. *2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence)*,

p. 854–857, set. 2014. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6949331>>.

PARK, J.; AN, Y.; YEOM, K. Virtual cloud bank: An architectural approach for intermediating cloud services. *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2015 - Proceedings*, 2015.

PATEL, G.; MEHTA, R.; BHOI, U. Enhanced Load Balanced Min-min Algorithm for Static Meta Task Scheduling in Cloud Computing. *Procedia Computer Science*, v. 57, p. 545–553, 2015.

PATEL, K. S.; SARJE, A. K. VM Provisioning Policies to Improve the Profit of Cloud Infrastructure Service Providers. n. July, 2012.

PIRAGHAJ, S. F.; CALHEIROS, R. N. Virtual Machine Customization and Task Mapping Model for Efficient Allocation of Cloud Data Center Resources Introduction to Cloud Computing : 2014.

POINT, T. *WSDL Document Example*. Disponível em: <http://www.tutorialspoint.com/wsdl/wsdl_example.htm>. Acesso em: 2 nov. 2016.

POOLA, D. *et al.* Robust scheduling of scientific workflows with deadline and budget constraints in clouds. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, p. 858–865, 2014.

PSIUK, M.; BUJOK, T.; ZIELIŃSKI, K. Enterprise service bus monitoring framework for SOA systems. *IEEE Transactions on Services Computing*, v. 5, n. 3, p. 450–466, 2012.

PUTHAL, D. *et al.* Cloud Computing Features, Issues, and Challenges: A Big Picture. *2015 International Conference on Computational Intelligence and Networks*, p. 116–123, 2015. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7053814>>.

RAJ, G. An Efficient Broker Cloud Management System. p. 72–76, 2011.

RAJ, G.; SETIA, S. Effective Cost Mechanism for Cloudlet Retransmission and Prioritized VM Scheduling Mechanism over Broker Virtual Machine Communication Framework. v. 2, n. 3, p. 41–50, 2012.

RAJU, R. *et al.* A heuristic fault tolerant MapReduce framework for minimizing

makespan in Hybrid Cloud Environment. *Proceeding of the IEEE International Conference on Green Computing, Communication and Electrical Engineering, ICGCCEE 2014*, p. 1–4, 2014.

REZK, E.; FOUFOU, S. A survey of semantic web concepts applied in web services and big data. *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, v. 2014, p. 767–772, 2015.

SAKELLARI, G.; LOUKAS, G. Simulation Modelling Practice and Theory A survey of mathematical models , simulation approaches and testbeds used for research in cloud computing. *Simulation Modelling Practice and Theory*, v. 39, p. 92–103, 2013. Disponível em: <<http://dx.doi.org/10.1016/j.simpat.2013.04.002>>.

SAMREEN, F.; BLAIR, G. S.; ROWE, M. Adaptive decision making in multi-cloud management. *Proceedings of the 2nd International Workshop on CrossCloud Systems - CCB '14*, p. 1–6, 2014. Disponível em: <<http://0-dl.acm.org.cataleg.uoc.edu/citation.cfm?id=2676662.2676676>>.

SANDHU, R.; SOOD, S. K. Scheduling of big data applications on distributed cloud based on QoS parameters. *Cluster Computing*, p. 817–828, 2014. Disponível em: <<http://dx.doi.org/10.1007/s10586-014-0416-6>>.

SENNA, C. R. *et al.* An Environment for Evaluation and Testing of Service Workflow Schedulers in Clouds. p. 301–307, 2011.

SINDHU, S.; MUKHERJEE, S. Efficient Task Scheduling Algorithms for Cloud Computing Environment. ... *Performance Architecture and Grid Computing*. [S.l: s.n.], 2011. p. 79–83. Disponível em: <<http://www.springerlink.com/index/R681083132486572.pdf>>.

SINGH, R. M.; PAUL, S.; KUMAR, A. Task Scheduling in Cloud Computing : Review. v. 5, n. 6, p. 7940–7944, 2014.

SOFTLAYER. *SoftLayer - Private Clouds*. Disponível em: <<http://www.softlayer.com/pt-br/private-clouds>>. Acesso em: 2 fev. 2016.

SOTIRIADIS, S.; BESSIS, N.; ANTONOPOULOS, N. Towards Inter-cloud Simulation Performance Analysis: Exploring Service-Oriented Benchmarks of Clouds in SimIC. *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, p. 765–771, mar. 2013a. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6550488>>. Acesso em: 14

nov. 2015.

SOTIRIADIS, S.; BESSIS, N.; ANTONOPOULOS, N. Towards Inter-cloud Simulation Performance Analysis: Exploring Service-Oriented Benchmarks of Clouds in SimIC. *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, p. 765–771, mar. 2013b. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6550488>>. Acesso em: 21 maio 2015.

TANENBAUM, A. S.; STEEN, M. VAN. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006.

TEREFE, M. B. *et al.* Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing. *Pervasive and Mobile Computing*, 2015. Disponível em: <<http://dx.doi.org/10.1016/j.pmcj.2015.10.008>>.

TEWARI, V. User Oriented Web Services Discovery based on QoS Parameters in Multiple Registries. v. 46, n. 24, p. 8–12, 2012.

TSAI, C.-W. *et al.* A Hyper-Heuristic Scheduling Algorithm for Cloud. *IEEE Transactions on Cloud Computing*, v. PP, n. 99, p. 1–1, 2014. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6784130>>.

TYAGI, V.; KUMAR, T. ORT Broker Policy: Reduce Cost and Response Time Using Throttled Load Balancing Algorithm. *Procedia Computer Science*, v. 48, n. Iccc, p. 217–221, 2015. Disponível em: <<http://linkinghub.elsevier.com/retrieve/pii/S1877050915006821>>.

VAN DEN BOSSCHE, R.; 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. Disponível em: <<http://dx.doi.org/10.1016/j.future.2012.12.012>>.

VARALAKSHMI, P.; MAHESHWARI, K. Cost-optimized resource provisioning in cloud. *2013 International Conference on Recent Trends in Information Technology (ICRTIT)*, p. 108–112, jul. 2013. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6844189>>.

VÁZQUEZ-POLETTI, A. N. J. L. *et al.* iCanCloud : A Flexible and Scalable Cloud Infrastructure Simulator. p. 185–209, 2012.

VEEN, J. S. VAN DER *et al.* Deployment strategies for distributed applications on

cloud computing infrastructures. *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, v. 2, p. 228–233, 2013.

VOS, L. H. INFORMS Tutorials in Operations Research Decision Analytics for Cloud Computing : A Classification and Literature Review Decision Analytics for Cloud Computing : A Classification and Literature Review. n. May 2015, 2014.

WICKREMASINGHE, B.; CALHEIROS, R. N.; BUYYA, R. CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications. *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, p. 446–452, 2010. Disponível em: <<http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5474733>>. Acesso em: 4 maio 2015.

WILLS, G. B. Cloud Computing. p. 1–52, 2014. Disponível em: <<https://secure.ecs.soton.ac.uk/notes/info3003/docs/>>.

WU, D. *et al.* Cloud manufacturing: Strategic vision and state-of-the-art. *Journal of Manufacturing Systems*, v. 32, n. 4, p. 564–579, 2013.

WU, L.; GARG, S. K.; BUYYA, R. SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments. 2011.

WU, L.; KUMAR GARG, S.; BUYYA, R. SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments. *Journal of Computer and System Sciences*, v. 78, n. 5, p. 1280–1299, 2012. Disponível em: <<http://dx.doi.org/10.1016/j.jcss.2011.12.014>>.

YANG, Z.; YIN, C.; LIU, Y. A Cost-based Resource Scheduling Paradigm in Cloud Computing. p. 417–422, 2011.

YOKOYAMA, S.; YOSHIOKA, N. An academic community cloud architecture for science applications. *Proceedings - 2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet, SAINT 2012*, p. 108–112, 2012.

ZHAO, W. *et al.* Modeling and Simulation of Cloud Computing : A Review. p. 20–24, 2012.

ZHENG, Z.; ZHANG, Y.; LYU, M. R. Investigating QoS of real-world web services. *IEEE Transactions on Services Computing*, v. 7, n. 1, p. 32–39, 2014.

ZHU, K. *et al.* Hybrid Genetic Algorithm for Cloud Computing Applications. 2011.

APÊNDICE A

Modificações efetuadas na Classe Cloudlet

Conforme explanado durante a discussão desta tese, modificações em classes da API do CloudSim se fizeram necessárias para atingir os objetivos do trabalho e os efeitos desejados em simulação. Assim, esse apêndice irá apresentar, de forma breve, quais foram as alterações realizadas na classe Cloudlet.

A classe *Cloudlet* original da API do CloudSim foi modificada sendo acrescidos campos de instância e métodos ao seu código para adaptá-la às necessidades da arquitetura BEQoS. Algumas das variáveis de classe inseridas foram *priority* (*int*), *maxExecutionTime* (*double*), *nonQoS* (*boolean*), *serviceID* (*int*), *arrivalTime* (*double*), *scheduleTime* (*double*), *clientID* (*int*), *groupID* (*int*), *groupLength* (*int*), *sendTime* (*double*) e *ReceiveTime* (*double*).

A

Tabela A-1 possui a descrição de cada uma das variáveis de instância mencionadas.

Tabela A-1: Descrição dos campos de instância adicionados na classe *Cloudlet*.

Campo de instância	Tipo	Descrição
<i>priority</i>	int	Usada em conjunto com a política de escalonamento de consumo eficiente de energia. Os valores que essa variável assume estão associados com constantes que estipulam o padrão para as prioridades de tarefas.
<i>maxExecutionTime</i>	double	Parâmetro majoritário das políticas de escalonamento com QoS, pois define o tempo máximo que uma requisição pode aguardar para ser executada.
<i>nonQoS</i>	boolean	Utilizada pelo <i>broker</i> para as situações na qual a requisição se utilizará de um algoritmo que não utiliza uma política de QoS rígida.
<i>serviceID</i>	int	Define o identificador de serviço que a requisição de cliente está solicitando.
<i>arrivalTime</i>	double	valor referente ao tempo de chegada na requisição no <i>broker</i> .
<i>scheduleTime</i>	double	Armazena o valor do horário de início para uma tarefa que tenha sido agendada por um algoritmo de escalonamento que considere a estratégia de agendamento de horário para início das tarefas.
<i>clientID</i>	int	Contém o número identificador do cliente de origem para a requisição (<i>cloudlet</i>).
<i>groupID</i>	int	Registra um número identificador único para grupo que foi enviado por um cliente.
<i>groupLength</i>	int	Armazena a quantidade de requisições que fazem parte de um dado grupo.
<i>sendTime</i>	double	Registra o momento, conforme relógio do simulador, em que a requisição sai do cliente.
<i>receiveTime</i>	double	Registra o momento em que a requisição é devolvida (com a resposta) ao cliente.

É importante citar que cada uma das variáveis de instância supracitadas possui seus respectivos métodos *getter* e *setter* para a manipulação de seus valores, garantindo, desta forma, o encapsulamento de dados de cada objeto *cloudlet*. Conforme já mencionado, algumas constantes também foram inseridas na classe *Cloudlet* para padronizar os valores estabelecidos para os algoritmos de escalonamento de tarefas baseados em consumo eficiente de energia com diferenciação no atendimento por prioridades. As constantes criadas foram: `NO_PRIORITY` (com valor zero), `HIGH_PRIORITY` (com valor 1), `MEDIUM_PRIORITY` (com valor 2) e `LOW_PRIORITY` (com valor 3).

APÊNDICE B

Codificação de uma classe de experimento de simulação

Este anexo visa a demonstração e explanação de um programa de simulação completo utilizando os componentes da arquitetura CloudSim-BEQoS.

Os programas que representam experimentos são compostos pelo método principal (ou método *main*) e pelas declarações, instanciações e configurações efetuadas por meio de classes da API do simulador CloudSim em conjunto com as classes da arquitetura CloudSim-BEQoS.

Ao criar o arquivo do programa de experimento de simulação, uma classe contendo o método *main* é estruturada, conforme apresentado no Quadro B.1.

Quadro B.1: Exemplo de declaração de classe de objetos para programa de simulação.

```
public class Experimento1 {  
    public static void main(String[] args) {  
        //TO-DO...  
    }  
}
```

No conteúdo do método *main*, inicia-se pela configuração de alguns objetos de base do simulador CloudSim (conforme Quadro B.2), como é o caso do objeto que habilita a impressão dos comandos de saída (objeto *Log*) na janela de terminal do sistema operacional ou IDE que estiver sendo utilizado. O simulador é inicializado por meio do método *init*, o qual recebe como parâmetros: um valor inteiro referente ao número de entidades do *users* (no caso da API padrão do CloudSim os *DatacenterBrokers* são considerados usuários do ambiente de nuvem), uma instância do objeto *Calendar* (usado para inicializar o relógio lógico do ambiente de simulação) e um valor booleano para habilitar ou não o *tracing* (ou rastreamento) de eventos ocorridos durante a execução.

Quadro B.2: Inicialização de objetos de base do simulador CloudSim.

```
Log.enable();  
Calendar calendario = Calendar.getInstance();  
boolean trace_flag = false;  
CloudSim.init(Parametros.num_usuarios, calendario, trace_flag);
```

Na sequência, a próxima declaração deverá ser a de uma instância da classe *DatacenterBroker* ou, no caso do uso da arquitetura CloudSim-BEQoS, usa-se uma instância do *QBroker*, conforme apresentado no Quadro B.3. Após a declaração e inicialização da instância do *cloud broker* é configurado o modo de operação padrão para o processo de mediação de requisições.

Quadro B.3: Declaração e inicialização de instância da classe *QBroker*.

```
QBroker qbroker = new QBroker("QBroker");  
qbroker.setOperation_mode(QBroker.INTERMEDIATION_SCHEDULE);
```

No caso do código do Quadro B.3, é possível verificar que o modo de operação configurado é o de intermediação. A classe *QBroker* fornece uma constante para cada um dos modos de operação (intermediação, agregação e arbitragem) de modo a facilitar a configuração desse componente.

A próxima sequência de comandos, conforme apresentado no Quadro B.4, é referente à declaração de objetos do tipo *DatacenterConfiguration*, os quais são utilizados na definição das características utilizadas pelo simulador para representar configurações dos componentes *Hosts* e algumas informações referentes ao custo do uso de recursos conforme o modelo de custo padrão do CloudSim.

A seguir duas instâncias da classe *Datacenter* (ainda conforme Quadro B.4) são criadas a partir da chamada ao método *createCustomDatacenter* da classe *Utilidades* (ou *Utilities*). O referido método de construção dos *datacenters* recebe como parâmetros o nome do *datacenter* a ser utilizado no ambiente de simulação, o tipo do *datacenter* considerado (quando privado usa-se o valor “*prv*” e quando público usa-se o valor “*pub*”) e uma instância de *DatacenterConfiguration*, provendo como retorno uma instância de da classe *Datacenter* pronta a ser utilizada na simulação.

Quadro B.4: Definição de objetos de configuração e declaração das instâncias de *Datacenter*.

```
DatacenterConfiguration conf_dc_privado = new
DatacenterConfiguration();
    conf_dc_privado.hostId = 1;
    conf_dc_privado.mips = 10000;
    conf_dc_privado.num_cores = 4;
    conf_dc_privado.num_hosts = 15;
    conf_dc_privado.ram = 8000;
    conf_dc_privado.bw = 1000;
    conf_dc_privado.storage = 500000;
    conf_dc_privado.os = "Linux";
    conf_dc_privado.vmm = "Xen";
    conf_dc_privado.cost = 0.0;
    conf_dc_privado.costPerBw = 0.0;
    conf_dc_privado.costPerMem = 0.0;
    conf_dc_privado.costPerStorage = 0.0;

DatacenterConfiguration conf_dc_publico = new
DatacenterConfiguration();
    conf_dc_publico.hostId = 9000;
    conf_dc_publico.mips = 20000;
    conf_dc_publico.num_cores = 8;
    conf_dc_publico.num_hosts = 30;
    conf_dc_publico.ram = 32000;
    conf_dc_publico.bw = 10000;
    conf_dc_publico.storage = 1000000;
    conf_dc_publico.os = "Linux";
    conf_dc_publico.vmm = "Xen";
    conf_dc_publico.cost = 0.0;
    conf_dc_publico.costPerBw = 0.0;
    conf_dc_publico.costPerMem = 0.0;
    conf_dc_publico.costPerStorage = 0.0;

Datacenter prv_cloud =
Utilidades.createCustomDatacenter("Nuvem_Privada", "prv",
conf_dc_privado);
Datacenter pub_cloud =
Utilidades.createCustomDatacenter("Nuvem_Publica", "pub",
conf_dc_publico);
```

Após a instanciação dos objetos *Datacenters* são instanciados objetos *VMConfiguration* (conforme apresentado no código do Quadro B.5), os quais são usados na definição de configurações homogêneas utilizadas pelas VMs a serem instanciadas para cada um dos *datacenters*.

Quadro B.5: Definição de objetos de configuração e criação de listas de VMs.

```

VMConfiguration conf_vm_prv = new VMConfiguration();
conf_vm_prv.idShift = 1;
conf_vm_prv.bw = 100;
conf_vm_prv.pesNumber = 1;
conf_vm_prv.mips = 10000;
conf_vm_prv.vmType = "prv";
conf_vm_prv.vmm = "Xen";
conf_vm_prv.size = 125000;
conf_vm_prv.ram = 2000;
conf_vm_prv.numInitialServices = 5;
conf_vm_prv.maxNumberServicesByVM = 5;
conf_vm_prv.totalNumberServices = 5;

VMConfiguration conf_vm_pub = new VMConfiguration();
conf_vm_pub.idShift = 1301;
conf_vm_pub.bw = 1000;
conf_vm_pub.pesNumber = 1;
conf_vm_pub.mips = 20000;
conf_vm_pub.vmType = "pub";
conf_vm_pub.vmm = "Xen";
conf_vm_pub.size = 250000;
conf_vm_pub.ram = 4000;
conf_vm_pub.numInitialServices = 5;
conf_vm_pub.maxNumberServicesByVM = 5;
conf_vm_pub.totalNumberServices = 5;

```

Em seguida, as listas de VMs são criadas por meio do método *createCustomVM* da classe *Utilidades*, conforme código apresentado no Quadro B.6. O resultado da chamada a método citada resulta numa lista de VMs que é passada como argumento para a lista de VMs da nuvem privada ou pública que consta da classe de configuração *Parametros* (a qual possui as variáveis ou objetos de configuração comuns), conforme o caso.

Quadro B.6: Criação de listas de VMs no programa de simulação.

```

Parametros.lista_vm_prv.clear();
Parametros.lista_vm_prv.addAll(Utilidades.createCustomVM(qbroke
r.getId(), 20, conf_vm_prv));
Parametros.lista_vm_pub.clear();
Parametros.lista_vm_pub.addAll(Utilidades.createCustomVM(qbroke
r.getId(), 10, conf_vm_pub));

```

As listas de VMs já associadas com seus respectivos *datacenters* de origem são atribuídas à lista de controle de VMs do componente mediador, isto é, de uma instância do tipo *QBroker*, conforme o Quadro B.7.

Quadro B.7: Atribuição das listas de VMs geradas para o *QBroker*.

```

qbroker.submitVmList(Parametros.lista_vm_prv);
qbroker.submitVmList(Parametros.lista_vm_pub);

```

As próximas instruções de código são para a criação de uma tabela de demanda de serviços (conforme apresentado no código do Quadro B.8), a qual é utilizada pela entidade cliente na determinação de comprimentos de tarefa específicos para cada serviço do ambiente SOA. Assim, cada identificador (ou ID) numerado de serviço deverá associar-se a um número inteiro que significa o valor médio em MI (Milhões de Instruções) que o serviço irá atribuir as tarefas que forem geradas com aquele dado ID. Os valores médios em MI são associados a funções exponenciais usando a classe de geração de número exponenciais *ExponentialDistr*, pertencente a API padrão do CloudSim.

Quadro B.8: Definição de objeto referente a tabela de demanda de serviços (*ServiceDemandTable*).

```
ServiceDemandTable sdt = new ServiceDemandTable();  
sdt.putServiceDemand(1, 30000); //S1-30.000MI  
sdt.putServiceDemand(2, 50000); //S2-50.000MI  
sdt.putServiceDemand(3, 70000); //S3-70.000MI  
sdt.putServiceDemand(4, 90000); //S4-90.000MI  
sdt.putServiceDemand(5, 110000); //S5-110.000MI
```

A próxima etapa na codificação do programa de simulação, conforme Quadro B.9, é a definição de uma instância da classe *Cloudlet* (ou tarefa), a qual deverá possuir as configurações básicas a serem seguidas pelas instâncias de clientes virtuais (referente a classe *Client*) para a geração de requisições e submissão ao *cloud broker* em tempo de simulação.

Quadro B.9: Instância de *cloudlet* como protótipo para geração de requisições em tempo de simulação.

```
Cloudlet prototipo = new Cloudlet(1, 2, 1, 300, 300,  
new UtilizationModelFull(), new UtilizationModelFull(),  
new UtilizationModelFull());  
prototipo.setMaxExecutionTime(0.0);  
prototipo.setNonQoS(false);
```

Para o código que consta do Quadro B.9 é importante mencionar que a criação de uma *cloudlet* como protótipo permite a definição dos modelos de utilização da tarefa. No caso apresentado, usa-se o modelo *UtilizationModelFull* o qual é associado com a execução de tarefas que executam como *jobs*, isto é, uma vez que estão de posse de uma CPU a tarefa irá executar até o seu final. Além disso, outros parâmetros importantes são configurados, como o atributo de classe *maxExecutionTime* (valor do tipo *double*), o qual representa o parâmetro de QoS de tempo máximo de execução da tarefa em segundos (ou instantes de simulação). No exemplo do Quadro B.9, o valor do atributo *maxExecutionTime* foi definido como zero, significando para a classe *Client* que o valor deste campo deverá ser gerado com um valor

proporcional ao tamanho da tarefa, item que, conforme já discutido é gerado a partir da tabela de demanda de serviço.

No Quadro B.10, é possível verificar que o conjunto subsequente de comandos definirá, por meio de um *looping*, a quantidade de clientes virtuais. Conforme o valor da variável *num_clientes* o número de clientes deverá ser de 150. Dentro do bloco de comandos do *looping* do tipo *for*, é possível perceber um comando de instanciação de objeto da classe *Client*, cujo construtor usado define o ID de entidade de simulação, a instância de *cloud broker* com a qual o cliente estará associado, um texto que define o nome do cliente virtual e uma constante que define o modo de submissão de requisições do cliente (no caso, está sendo utilizado o modo de submissão em tempo real). Para resumir a codificação, os demais comandos são, em sua maioria, métodos *setters* cujo objetivo é parametrizar as características de comportamento do objeto cliente para o tempo de execução da simulação. Dentre as parametrizações do cliente estão incluídas a configuração do número de serviços, desabilitação do uso de geração de prioridades randômicas para tarefas, habilitação do uso de geração de identificadores de serviço randômicos para tarefas, valor inicial para geração de identificadores de tarefas, definição da *cloudlet* protótipo, habilitação de espera pela resposta de requisição, habilitação de intervalo entre chegadas e demanda de serviço sem rajadas, configuração do valor médio de intervalo de chegadas (definido em 7,0 segundos), atribuição da referência ao objeto da tabela de demanda de serviço, configuração de valor referente ao tempo médio de fila a ser considerado para geração do atributo de QoS de tempo máximo de execução de tarefa, habilitação do uso uma distribuição em porcentagem para a geração de identificadores de serviço. Após a finalização da configuração do cliente, é gerado um valor aleatório referente ao *delay* de rede a ser considerado no envio e recebimento de mensagens entre cada cliente e o *cloud broker*. O valor gerado é atribuído como novo *link* de comunicação do ambiente de nuvem por meio do método *addLink* do objeto *NetworkTopology* (pertencente a API original do *CloudSim*).

Quadro B.10: Criação e configuração das instâncias de clientes virtuais no programa de simulação.

```

int num_clientes = 150;
int idshift = 1;
Random rdn = new Random();
for (int i = 0; i < num_clientes; i++) {
    Client ul = new Client(-1, qbroker, ("User_" + i),
        Client.REAL_TIME_REQUESTS);
    ul.setNumServices(5);
    ul.setRdn_prioridade(false);
    ul.setRdn_servico(true);
    ul.setCloudletIdShift(idshift);
    ul.setCloudlet_default(prototipo);
    ul.enableWaitingForResponse();
    ul.enableSDWithoutBurst();
    ul.enableIATWithoutBurst();
    ul.setMeanIAT(7.0);
    ul.setServiceDemandTable(sdt);
    ul.setMediumQueueValue(10.0);
    ul.setRandomServiceIdWithDistribution(true);

    double delay = 50 + (rdn.nextInt(450));
    NetworkTopology.addLink(qbroker.getId(), ul.getId(), 1000,
        delay / 1000);
    idshift += 500;
}

```

Os próximos componentes declarados no programa de simulação são os monitores, os quais dão suporte de informação de estado do ambiente de nuvem ao *cloud broker*. Conforme código apresentado no Quadro B.11 são criadas instâncias das classes *MonitorRecursos* e *MonitorRequisicoes*. A instância de monitor de recursos é utilizada para acompanhar o uso de recursos dos *datacenters* retornando a porcentagem total de consumo de recursos. O monitor de requisições é utilizado no acompanhamento e registro da chegada de requisições ao *broker*, além disso, ainda no código do Quadro B.11 é configurado o valor de intervalo de monitoramento para o monitor de requisições.

Quadro B.11: Instanciação de componentes Monitores da arquitetura CloudSim-BEQoS.

```

MonitorRecursos mnt1 = new MonitorRecursos("Mon_Recursos",
qbroker);
MonitorRequisicoes mnt2 = new
MonitorRequisicoes("Mon_Requisicoes", qbroker);
mnt2.setIntervalo(14.0);
qbroker.setMonitorRequisicoes(mnt2);

```

Embora não seja utilizado no exemplo de programa de simulação, um monitor de energia (representado pela classe *MonitorEnergia*) pode ser utilizado para o caso de cenários que considerem o consumo energético nos *datacenters*.

Com os comandos de programa apresentados até o presente momento, já é possível inicializar o processo de simulação. Contudo, como o tipo de modo de submissão de requisições dos clientes é o de tempo real (*Client.REAL_TIME_REQUESTS*), é necessário configurar o tempo de simulação total (conforme comando apresentado no Quadro B.12).

Quadro B.12: Comandos de configuração de tempo de simulação e início do *runtime* do simulador CloudSim.

```
CloudSim.terminateSimulation(9000.0);  
CloudSim.startSimulation();
```

Após a execução completa do tempo de simulação (no total de 9000 segundos ou instantes de simulação que correspondem a duas horas e meia de execução de tempo real) os próximos comandos do programa coletam dados dos objetos instanciados no ambiente de simulação e usam-nos para gravar arquivos de saída ou imprimir as informações em janela de terminal de caracteres de texto. No Quadro B.13 são apresentados os comandos cuja função é a leitura das *cloudlets* que foram executadas durante o tempo de simulação, a impressão do relatório de saída do CloudSim no terminal (por meio do método estático *printCloudletList* da classe Utilidades) e a gravação desses mesmos dados em um arquivo com extensão XLS para uso na ferramenta de planilhas eletrônicas Microsoft Excel. É válido frisar que muitos outros tipos de dados e informações podem ser gravados em arquivos de saída XLS por meio do uso de outros métodos da classe *Utilidades*, por meio da qual podem ser gravadas informações a partir dos componentes monitores instanciados, informações sobre monitoramento da taxa de chegada das *cloudlets* bem como informações da carga de trabalho de *datacenters* conforme intervalos de tempo de monitoramento.

Quadro B.13: Comandos de finalização do programa de simulação e leitura ou gravação de dados e informações.

```
List<Cloudlet> resultados = qbroker.getCloudletReceivedList();  
String retorno = Utilidades.printCloudletList(resultados);  
Utilidades.registraSaidaCloudSimReport(retorno);
```

A junção de todos os comandos de programa, apresentados como exemplos nesta subseção, em um método principal de uma classe de programa de simulação constituem um cenário experimental possível de ser executado com a arquitetura CloudSim-BEQoS.

APÊNDICE C

Simulação com arquitetura CloudSim-BEQoS e QBroker

Neste apêndice será dada ênfase à demonstração de como se dá a utilização da arquitetura CloudSim-BEQoS e da arquitetura do QBroker em programas de simulação.

A intenção é explicar alguns pontos principais da codificação, apresentando as diferenças de codificação e comportamento em relação aos programas de simulação que se utilizam da API padrão do CloudSim.

A seguir, uma explanação completa dos principais trechos de código da implementação da classe QBroker será apresentada por meio de subseções que discutem as particularidades importantes da implementação realizada.

Constantes e variáveis de instância da classe QBroker

Esta seção apresenta a discussão de algumas partes do código da classe QBroker pertencente ao *br.usp.icmc.lasdpc.BEQoS.classes*. O objeto é explanar as principais variáveis, constantes e métodos que possibilitam a execução dos modos de operação.

Inicialmente, é importante reafirmar que a QBroker (componente do conjunto de classes que compõem a arquitetura CloudSim-BEQoS) é uma classe estendida, por meio de herança, da classe *DatacenterBroker* (esta última por sua vez é estendida da classe *SimEntity*) da API original do CloudSim, conforme é possível perceber pela assinatura de classe apresentada no Quadro C.14. Dessa forma, a classe QBroker passa a ser considerada uma classe que represente uma entidade de simulação.

Quadro C.14: Assinatura da classe QBroker.

```
public class QBroker extends DatacenterBroker {  
    //Códigos de programa da classe...  
}
```

Dentre os principais itens componentes do código da classe QBroker incluem-se um conjunto de constantes (conforme Quadro C.15) referentes aos identificadores dos eventos de simulação atendidos e também há constantes que representam os modos de operação que podem ser configurados no *cloud broker*.

Quadro C.15: Constantes da classe QBroker.

```
//Constantes de evento para o DatacenterBroker
public static final int NEW_CLOUDLET_ARRIVAL = 11001;
public static final int NEW_GROUP_CLOUDLET_ARRIVAL = 11002;
public static final int RECEIVE_NETWORK_PACKET = 11003;
public static final int SUBMIT_SCHEDULED_CLOUDLET = 11004;
//Constantes de configuração do método executeScheduling
public static final int INTERMEDIATION_SCHEDULE = 2000;
public static final int AGGREGATION_SCHEDULE = 2001;
public static final int ARBITRATION_SCHEDULE = 2002;
public static final int CLOUDSIM_DEFAULT_SCHEDULE = 2003;
```

Um conjunto de variáveis de instância (Quadro C.3) foi empregado na configuração e controle do *cloud broker* quanto aos aspectos de comportamento do mesmo no processo de mediação.

Quadro C.3: Variáveis de instância da classe QBroker.

```
//Variáveis de instância
protected static int vmIndex = 0;
private int operation_mode = 0;
private boolean service_sched = false;
private boolean priority_sched = false;
private boolean qos_sched = false;
private boolean power_sched = false;
private boolean workload_sched = false;
private boolean messages = true;
private int cloudletCount = 0;
private MonitorRequisicoes mr = null;
private List<Cloudlet> missedCloudletList;
private int missedCloudletCount = 0;
```

Um considerável número de métodos foi implementado, dentre eles métodos *getter* e *setter* para acesso e modificação de valores das variáveis de instância, alguns métodos são sobrescritas (*overriding*) de métodos herdados das classes *DatacenterBroker* ou *SimEntity*. Há métodos utilitários para checagem de *status* das filas (implementadas como objetos do tipo lista ou *List*) do *broker* e os três principais métodos criados para o processo de mediação referentes aos modos de operação: *intermediation*, *agregation* e *arbitration*. A classe QBroker ainda oferece uma versão com melhorias da versão original do algoritmo de escalonamento da classe *DatacenterBroker*, a qual é possível utilizar com o QBroker a partir da configuração do modo

de operação `CLOUDSIM_DEFAULT_SCHEDULE` (Quadro C.15). O método referente a essa implementação na classe `QBroker` tem o nome *submitIndividualCloudletRRobin*.

Construtor de Classe

Ao se trabalhar com herança entre classes de objeto, alguns métodos abstratos definidos nas superclasses precisam, obrigatoriamente, ser implementados. No caso da classe `QBroker`, a qual é subclasse direta de *DatacenterBroker*, foi necessária a implementação de apenas um método: um construtor que recebe um parâmetro do tipo *String* referente ao nome de entidade a ser dado para identificação do *cloud broker* em tempo de simulação.

Além disso, outros comandos de configuração inicial foram introduzidos, conforme apresentado no Quadro C.4.

Quadro C.4: Implementação de métodos obrigatórios pela herança da classe *DatacenterBroker*.

```
public QBroker(String name) throws Exception {  
    super(name);  
    MetaCloudletScheduler.setBrk(this);  
    this.setMissedCloudletList(new ArrayList<Cloudlet>());  
    missedCloudletCount = 0;  
}
```

Método para processamento de eventos

Além do construtor, o método de instância *processEvent* da superclasse é sobrescrito para a adição de novas funcionalidades.

O método *processEvent* manipula os eventos que são enviados para a entidade durante o tempo de execução do simulador. Um resumo da implementação do referido método é apresentado no Quadro C.5.

Quadro C.5: Resumo de implementação do método *processEvent* na classe QBroker.

```
@Override
public void processEvent(SimEvent ev) {
    switch (ev.getTag()) {
        case NEW_GROUP_CLOUDLET_ARRIVAL:
            //códigos de programa do evento
        case NEW_CLOUDLET_ARRIVAL:
            //códigos de programa do evento
        case QBroker.SUBMIT_SCHEDULED_CLOUDLET:
            //códigos de programa do evento
            break;
        case CloudSimTags.CLOUDLET_RETURN:
            this.processCloudletReturn(ev);
            break;
        case CloudSimTags.END_OF_SIMULATION:
            //códigos de programa do evento
            break;
        default:
            processOtherEvent(ev);
            break;
    }
}
```

A partir da análise do código do Quadro C.5 é possível verificar que os eventos `NEW_GROUP_CLOUDLET_ARRIVAL`, `NEW_CLOUDLET_ARRIVAL` e `SUBMIT_SCHEDULED_CLOUDLET` são implementações novas que se referem a métodos de instância que lidam, respectivamente, com a chegada de grupos de *cloudlets*, chegada de *cloudlets* em modo individual e o despacho de *cloudlets* que foram escalonadas com sucesso pelo processo de mediação do *broker*.

O evento `CLOUDLET_RETURN` foi tratado por meio de uma nova implementação (*overriding*) no método *processCloudletReturn*, o qual possui uma versão original na classe *DatacenterBroker*. O evento `END_OF_SIMULATION` foi tratado por meio de alguns comandos de finalização necessários ao QBroker.

Evento de Chegada de Requisição

Na explanação corrente será dada ênfase ao código de programa do evento `NEW_CLOUDLET_ARRIVAL`, o qual está diretamente relacionado com o modo de operação de intermediação, o qual é o foco principal desta tese. Conforme apresentado no Quadro C.6, são executados vários comandos para converter os dados do evento para uma *cloudlet*, registrar o tempo de chegada da tarefa, sinalizar sua chegada ao monitor de requisições e, finalmente, invocar o método de escalonamento (*executeScheduling*) que irá acionar os mecanismos de mediação conforme a configuração do *broker*.

Quadro C.6: Comandos de tratamento do evento NEW_CLOUDLET_ARRIVAL do método *processEvent*.

```
@Override
public void processEvent(SimEvent ev) {
    switch (ev.getTag()) {

        //Demais casos de captura de eventos de simulação...

        case NEW_CLOUDLET_ARRIVAL:
            Cloudlet caux = (Cloudlet) ev.getData();
            caux.setArrivalTime(CloudSim.clock());
            if (mr != null) {
                mr.sinalizarChegadaRequisicao();
            }
            caux = (Cloudlet) executeScheduling(caux);
            if (caux != null && caux.getVmId() != -1 &&
                caux.getStatus() != Cloudlet.NOT_SCHEDULED) {
                getCloudletList().add(caux);
                submitIndividualCloudlet();
            } else {
                client = null;
                if (caux.getClientId() > 0) {
                    client = (Client)
                        CloudSim.getEntity(caux.getClientId());
                    client.receiveCloudletResponse();
                }
            }
            break;
    }
}
```

Ainda referente ao código do programa do Quadro C.6, é interessante perceber que a instância de *cloudlet* recebida no evento e persistida na variável *caux* é escalonada por meio da chamada ao método *executeScheduling* e, logo após, várias verificações são feitas no comando de desvio *if* subsequente para verificar se a mesma foi corretamente escalonada. Se todas as condições de verificação são verdadeiras, isto é, se a *cloudlet* retornada não é nula, se ela possui um identificador de VM diferente de -1 e se o *status* da mesma não foi devolvido como não escalonada (NOT_SCHEDULED) então ela é adicionada na lista de tarefas do *broker* e o método *submitIndividualCloudlet* é invocado. Caso a tarefa não tenha sido escalonada, a rotina do código efetua a devolução da resposta ao respectivo cliente.

Método para execução de política de escalamento

A próxima discussão é referente ao código de implementação do método *executeScheduling* (conforme Quadro C.7), o qual é responsável por acionar o modo de operação correto conforme a configuração que tiver sido efetuada no componente QBroker.

O método *executeScheduling* (Quadro C.7) recebe como parâmetro de entrada uma instância de *Object* (classe genérica que pode referenciar qualquer objeto) como nomeada como

“*o*”. No início do código, a variável *o* é verificada por meio da palavra-chave *instanceof* com o intuito de descobrir se o objeto recebido é uma *cloudlet* individual ou um grupo de tarefas (vetor de *cloudlets*). Na sequência, por meio de um comando de chaveamento (*switch*) é acionado o caso de execução conforme o valor que estiver hospedado na variável de instância *operation_mode*, a qual trata-se de um valor inteiro usado na configuração do modo de operação do *broker*.

No bloco de instruções do comando *switch* (ainda referente ao Quadro C.7) é possível verificar os casos de execução INTERMEDIATION_SCHEDULE, AGREGATION_SCHEDULE, ARBITRATION_SCHEDULE e CLOUDSIM_DEFAULT_SCHEDULE, os quais correspondem aos modos de operação compatíveis com a arquitetura do QBroker. Os comandos internos de cada caso de execução acionam o respectivo método que recebe o valor do objeto “*o*” devidamente convertido para *cloudlet* ou grupo de *cloudlets* e, após a execução da *procedure* de escalonamento do modo de operação, o elemento de requisição (*cloudlet* ou grupo) é obtido como retorno da chamada. Os comandos subsequentes verificam se o elemento de requisição foi ou não escalonado com sucesso, emitindo uma mensagem de erro nos casos de não escalonamento ou retornando à requisição escalonada para o código de programa do método *processEvent*.

O próximo item a ser discutido é o método *intermediation*. Este método possui a rotina referente ao algoritmo de intermediação com ou sem QoS que já foi discutido nas seções anteriores deste capítulo. Por trata-se de um método consideravelmente longo, sua explicação será dividida em: assinatura e comandos iniciais, rotina de escalonamento, rotina de alocação de serviços e rotina de intermediação sem QoS.

Quadro C.7: Implementação do método *executeScheduling* da classe QBroker.

```
protected Object executeScheduling(Object o) {
    Cloudlet c = null;
    Cloudlet grupo[] = null;
    if (o instanceof Cloudlet) {
        c = (Cloudlet) o;
    } else if (o instanceof Cloudlet[]) {
        grupo = (Cloudlet[]) o;
    }
    switch (this.getOperation_mode()) {
        case INTERMEDIATION_SCHEDULE:
            c = this.intermediation(c);
            if (c != null && c.getVmId() != -1) {
                return c;
            } else {
                c.setReceiveTime(CloudSim.clock() +
                    NetworkTopology.getDelay(this.getId(),
                        c.getClientId()));
                c.setStatus(Cloudlet.NOT_SCHEDULED);
                this.getMissedCloudletList().add(c);
                missedCloudletCount++;
                if (messages) {
                    Log.println("Cloudlet não atendida:" +
                        c.getCloudletLength() + "MIs | id=" +
                        c.getCloudletId() + " | serviceId=" +
                        c.getService());
                }
            }
            break;
        case AGGREGATION_SCHEDULE:
            grupo = aggregation(grupo);
            if (grupo != null) {
                return grupo;
            } else {
                Log.println("Não possível processar um grupo de
                    tarefas.");
                return null;
            }
        case ARBITRATION_SCHEDULE:
            grupo = arbitration(grupo);
            if (grupo != null) {
                return grupo;
            } else {
                Log.println("Não possível processar um grupo de
                    tarefas.");
                return null;
            }
        case CLOUDSIM_DEFAULT_SCHEDULE:
            c = submitIndividualCloudletRRRobin(c);
            break;
    }
    return c;
}
```

Método de intermediação

O código de programa da assinatura do método *intermediation*, bem como seus comandos iniciais são apresentados no Quadro C.8.

Quadro C.8: Assinatura e comandos iniciais do método *intermediation*.

```
public Cloudlet intermediation(Cloudlet c) {  
    List<Vm> listVm = null;  
    Vm vmx = null;  
    listVm = MetaCloudletScheduler.classifyVmsByServiceId(  
        c.getService(), getVmsCreatedList());  
}
```

É possível verificar, conforme o Quadro C.8, a criação de uma lista de VMs usada na seleção de VMs com o serviço solicitado na requisição de cliente.

Na sequência do código do método *intermediation* (Quadro C.9), há um comando de desvio que verifica se o método acionado no componente *MetaCloudletScheduler* retornou uma lista vazia ou cheia (referente à seleção de VMs com o serviço requisitado). Se a lista de VMs retorna vazia, então os procedimentos de tentativa de alocação de serviço são executados primeiro nas VMs ativas privadas e posteriormente nas VMs ativas públicas.

Para os comandos de alocação de serviço nas VMs ativas da nuvem privada é possível verificar, no Quadro C.9, que é feita uma solicitação ao componente *MetaCloudletScheduler* para ordenar todas as VMs por ordem descendente de tempo de fila. Em seguida, um comando de laço *for* verifica quais entre as VMs pertencem ao *datacenter* privado. Essa operação é feita por meio do ID das VMs, pois, por convenção, usou-se o valor numérico inteiro (usando uma sistemática de *hard code*) 1300 para representar o limite entre os IDs de VMs privadas (que usam valores entre 1 e 1299) e IDs de VMs públicas (que usam valores maiores ou iguais a 1300). Caso a VM tenha ID menor que 1300, são calculados então o tempo de execução da tarefa na VM e o tempo de fila da VM em questão. De posse dessas informações o algoritmo verifica se a VM ainda possui espaço de alocação para mais um serviço, se já não possui o serviço requerido, se não está com sua lista de serviços vazia (significa que é uma VM ainda não inicializada, em *Standby* ou ainda inativa) e se o tempo de fila é maior do que zero. Se todas essas condições forem satisfeitas a VM é candidata a alocar a tarefa, porém, é ainda necessário verificar o QoS de tempo máximo de execução. Tal verificação é realizada comparando-se o valor hospedado no campo *maxExecutionTime* da tarefa com a soma do tempo de execução da mesma mais o tempo de fila, se o tempo de serviço na VM for inferior ao *deadline* (considera-se como *deadline* o valor atribuído à variável *maxExecutionTime*) informado na referida restrição de QoS da requisição, então a VM é escolhida como recurso.

Em seguida, o código aloca o serviço na VM e associa a tarefa com o ID da VM escolhida, despachando a tarefa na sequência. Caso não exista VM privada para alocação da tarefa, o fluxo do código segue para a alocação de serviço em VMs ativas da nuvem privada seguindo, basicamente, uma implementação bastante semelhante, conforme apresentada no Quadro C.9.

No Quadro C.10 é apresentado o código de alocação de serviços em VMs em *Standby* nas nuvens privada e pública. Conforme já explanado, a *procedure* de verificação das VMs segue aspecto de codificação bastante similar ao procedimento de alocação em VMs ativas. Contudo, as diferenças principais entre a alocação de serviço em VMs ativas e em VMs *standby* encontra-se na verificação que é feita quanto ao tempo de fila das VMs e a quantidade de serviços instanciados. Para serem consideradas como VMs em *standby*, a informação do tempo de fila deve retornar o valor zero e a informação do número de serviços instanciados também deve ser zero. As VMs ativas possuem, pelo menos, um serviço instanciado e disponível ao ambiente de nuvem, além disso, podem possuir (ou não) um tempo de fila maior do que zero.

As atividades de alocação de serviços realizadas conforme o código do Quadro C.10 são as últimas atividades do bloco de código do comando de desvio *if*, o qual foi mencionado no Quadro C.9.

Encerrando-se esse primeiro membro do comando *if*, é possível verificar (ainda no Quadro C.10) o início do bloco de código do comando *else*. O bloco de comandos do comando *else* é executado quando o comando de seleção de VMs que contém o serviço solicitado pela requisição retonar a lista (variável *listVM*) com elementos (VMs). Quando ocorre essa situação, significa que existem VMs que possuem o serviço requisitado instanciado, assim, é possível tentar, em primeira instância, escalonar a requisição para uma VM ativa que já possua o serviço.

Quadro C.9: Comandos de alocação de serviço em VMs ativas do método *intermediation*.

```

if (listVm.isEmpty()) {
    listVm = MetaCloudletScheduler.
        getVmsDescendingOrderedQueue(getVmsCreatedList());
    for (Vm vm : listVm) {
        if (vm.getId() < 1300) {
            double queueTime = this.getTotalVmQueueTime(vm);
            double execTime = c.getCloudletLength() /
                (c.getNumberOfPes() * vm.getMips());
            if (vm.getServices().size() < vm.getNumMaxServices()
                && !vm.getServices().contains(c.getService())
                && !vm.getServices().isEmpty()
                && queueTime > 0.0) {
                if ((queueTime + execTime) <=
                    c.getMaxExecutionTime()) {
                    vm.addService(c.getService());
                    Log.println("Alocação de nova instância de
                        serviço " + c.getService() + " na VM #" +
                        vm.getId() + ".");
                    c.setVmId(vm.getId());
                    int posicao =
                        Utilidades.findVmCreatedIndex(vm.getId(),
                            getVmsCreatedList());
                    this.getVmsCreatedList().set(posicao, vm);
                    return c;
                }
            }
        }
    }
}

for (Vm vm : listVm) {
    if (vm.getId() >= 1300) {
        double queueTime = this.getTotalVmQueueTime(vm);
        double execTime = c.getCloudletLength() /
            (c.getNumberOfPes() * vm.getMips());
        if (vm.getServices().size() < vm.getNumMaxServices()
            && !vm.getServices().contains(c.getService())
            && !vm.getServices().isEmpty()
            && queueTime > 0.0) {
            if ((queueTime + execTime) <=
                c.getMaxExecutionTime()) {
                vm.addService(c.getService());
                Log.println("Tentativa 1: Alocando nova
                    instância de serviço " + c.getService() + "
                    na VM #" + vm.getId() + ".");
                c.setVmId(vm.getId());
                int posicao =
                    Utilidades.findVmCreatedIndex(vm.getId(),
                        getVmsCreatedList());
                this.getVmsCreatedList().set(posicao, vm);
                return c;
            }
        }
    }
}

//demais códigos do bloco atual de comandos...

```

Para as tentativas de escalonamento das tarefas conforme a lista de VMs retornada com elementos, a primeira atividade do fluxo de execução é tentar alocar a requisição para VMs com tempo de fila zero.

Quadro C.10: Comandos de alocação de serviço em VMs *standby* do método *intermediation*.

```
for (Vm vm : getVmsCreatedList()) {
    if (vm.getId() < 1300) {
        double queueTime = this.getTotalVmQueueTime(vm);
        double execTime = c.getCloudletLength() /
            (c.getNumberOfPes() * vm.getMips());
        if (queueTime == 0.0 && vm.getServices().isEmpty() &&
            (queueTime + execTime) <= c.getMaxExecutionTime()) {
            vm.addService(c.getService());
            Log.println("Alocação de nova instância do serviço"
                + c.getService() + " na VM #" + vm.getId() + ".");
            c.setVmId(vm.getId());
            int posicao =
                Utilidades.findVmCreatedIndex(vm.getId(),
                    getVmsCreatedList());
            this.getVmsCreatedList().set(posicao, vm);
            return c;
        }
    }
}
for (Vm vm : getVmsCreatedList()) {
    if (vm.getId() >= 1300) {
        double queueTime = this.getTotalVmQueueTime(vm);
        double execTime = c.getCloudletLength() /
            (c.getNumberOfPes() * vm.getMips());
        if (queueTime == 0.0
            && vm.getServices().isEmpty()
            && (queueTime + execTime) <= c.getMaxExecutionTime()) {
            vm.addService(c.getService());
            Log.println("Alocação de nova instância do serviço"
                + c.getService() + " na VM #" + vm.getId() + ".");
            c.setVmId(vm.getId());
            int posicao =
                Utilidades.findVmCreatedIndex(vm.getId(),
                    getVmsCreatedList());
            this.getVmsCreatedList().set(posicao, vm);
            return c;
        }
    }
}
} //fim do if VMs vazias
else
{
    //Caso a lista de VMs possua itens...
    //demais códigos de programa para escalonamento de tarefas...
```

Seguindo a premissa de prover prioridade às VMs do *datacenter* privado, primeiramente tenta-se efetuar tal atividade com a nuvem privada e, na sequência, caso não seja

possível alocar nas VMs privadas, então é feita a tentativa na nuvem pública. O bloco de código com essa lógica de execução é apresentado no Quadro C.11.

No código de escalonamento para VMs que possuem o serviço requisitado com tempo de fila zero (Quadro C.11), é interessante destacar o uso das funções de suporte do componente *MetaCloudletScheduler*, como é o caso dos métodos *getPrivateVmZeroQueue* e *getPublicVmZeroQueue*. Tais métodos representam uma lista de funcionalidades úteis ao desenvolvimento, pois, significam aproveitamento de códigos de programa frequentemente utilizados para as verificações a respeito do estado das VMs, dos serviços e de outras características que envolvem os recursos disponíveis ao uso no ambiente de nuvem.

Quadro C.11: Comandos de escalonamento de tarefa para VMs com tempo de fila zero.

```
vmx = MetaCloudletScheduler.getPrivateVmZeroQueue(listVm);
if (vmx != null) {
    double execTime = c.getCloudletLength() / (c.getNumberOfPes() *
        vmx.getMips());
    double queueTime = this.getTotalVmQueueTime(vmx);
    if ((queueTime + execTime) <= c.getMaxExecutionTime()) {
        c.setVmId(vmx.getId());
        return c;
    }
}

vmx = MetaCloudletScheduler.getPublicVmZeroQueue(listVm);
if (vmx != null) {
    double execTime = c.getCloudletLength() / (c.getNumberOfPes() *
        vmx.getMips());
    double queueTime = this.getTotalVmQueueTime(vmx);
    if ((queueTime + execTime) <= c.getMaxExecutionTime()) {
        c.setVmId(vmx.getId());
        return c;
    }
}
```

Após a tentativa de escalonar a tarefa para VMs que possuem o serviço instanciado com fila zero, o algoritmo de intermediação tenta realizar o escalonamento da tarefa para a VM com a menor fila entre as que possuem o serviço, conforme é possível verificar no Quadro C.12. Fazendo menção novamente às funcionalidades do *MetaCloudletScheduler*, é possível notar a invocação do método *getVmLowerQueue*, o qual retorna a VM com a menor fila dentre as que foram selecionadas pelo método de seleção *classifyVmsByServiceId* (conforme comando apresentado no Quadro C.8), também pertencente ao *MetaCloudletScheduler*.

Quadro C.12: Comandos de escalonamento de tarefa para VM que possui o serviço com a menor fila.

```
vmx = MetaCloudletScheduler.getVmLowerQueue(listVm);  
if (vmx != null) {  
    double execTime = c.getCloudletLength() /  
        (c.getNumberOfPes() * vmx.getMips());  
    double queueTime = this.getTotalVmQueueTime(vmx);  
    if ((queueTime + execTime) <= c.getMaxExecutionTime()) {  
        c.setVmId(vmx.getId());  
        return c;  
    }  
}
```

Após a tentativa de escalonar a tarefa para a VM com o serviço requisitado com a menor fila, se esta tentativa não obtiver sucesso, então serão executados os procedimentos de *tentativa de alocação de nova instância de serviço em VMs ativas* e depois a *tentativa de alocação de nova instância de serviço em VMs standby*, sempre considerando primeiro as VMs da nuvem privada e depois as VMs da nuvem pública, em ambos os procedimentos. Por serem códigos de programa que já foram apresentados e explanados conforme Quadro C.9 e Quadro C.10, esses trechos do algoritmo não serão novamente discutidos.

Após as tentativas de alocação de serviço, ocorre a finalização do bloco *else* que pertencia ao comando de desvio *if* de verificação da variável *listVm* (Quadro C.9). Em seguida, há um comando de desvio *if* (Quadro C.13) que verifica o conteúdo do atributo booleano *nonQoS* da *cloudlet* por meio do método *getter* chamado *isNonQoS*. Quando o valor do referido atributo do objeto *cloudlet* possui o valor verdadeiro (*true*), executa-se o bloco de comandos do desvio, assim, é feito o escalonamento da tarefa para a VM que possui o serviço querido instanciado com a menor fila possível.

O referido desvio (Quadro C.13) é utilizado quando se deseja que o cenário de simulação considere o processo de intermediação sem QoS, de forma que as tarefas com o atributo *nonQoS* configurado como *true* nunca são descartadas, porém, podem se escalonadas para VMs cujo tempo de serviço será superior ao tempo definido no atributo *maxExecutionTime*, o qual representa o principal parâmetro de QoS referente a desempenho de execução de tarefas.

É importante reforçar que todos os códigos de programa apresentados nos quadros desta seção, os quais tentam efetuar as atividades alocação ou escalonamento de tarefas, sempre verificam o parâmetro de QoS de tempo máximo de execução (*maxExecutionTime*).

Quadro C.13: Comando de desvio usado para configuração de intermediação sem QoS.

```
if (c.isNonQoS()) { //Apenas para intermediação SEM QoS
    listVm = MetaCloudletScheduler.classifyVmsByServiceId(
        c.getService(), getVmsCreatedList());
    vmx = MetaCloudletScheduler.getVmLowerQueue(listVm);
    if (vmx != null) {
        c.setVmId(vmx.getId());
        return c;
    }
}
return c; // #Cloudlet não foi atendida#
} //Fim do método intermediation
```

Na porção final de código de programa do método *intermediation*, conforme é possível observar no Quadro C.13, há um comando *return* antes da última chave de encerramento do bloco de comandos do método. Tal comando, quando executado, significa que todas as tentativas de negociação feitas pelo processo de intermediação com QoS do QBroker não obtiveram sucesso em atender a requisição de cliente de acordo com o valor definido na variável de tempo máximo de execução. Isso significa que o método que receber esse valor de retorno irá verificar que a *cloudlet* não foi atendida e então a tarefa é descartada.

É importante mencionar que é possível evitar o excesso de descarte de tarefas por meio de módulos que implementam mecanismos de renegociação.

A explicação provida sobre o código de implementação da classe QBroker da presente seção teve o objetivo de prover mais detalhes a cerca de como é o funcionamento da arquitetura de *cloud broker* proposta nesta tese. Algumas classes de objeto e trechos de código de programa foram subtraídas da explicação de modo a não alongar a explanação demais.