Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Agentes Autônomos para Monitoramento e Alocação Dinâmica de Recursos para Nuvem Computacional

Aldo Henrique Dias Mendes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Célia Ghedini Ralha

Brasília
2017

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Mestrado em Informática

Coordenadora: Prof.ª Dr.ª Célia Ghedini Ralha

Banca examinadora composta por:

Prof.ª Dr.ª Célia Ghedini Ralha (Orientadora) — CIC/UnB
Prof.ª Dr.ª Alba Cristina Magalhões Alves de Melo — CIC/UnB
Prof. Dr. Luiz Augusto Fontes Laranjeira — FGA/UnB

CIP — Catalogação Internacional na Publicação

Dias Mendes, Aldo Henrique.

Agentes Autônomos para Monitoramento e Alocação Dinâmica de Recursos para Nuvem Computacional / Aldo Henrique Dias Mendes. Brasília : UnB, 2017.
92 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2017.

1. Computação em Nuvem, 2. Sistema Multiagente, 3. Predição de Recursos, 4. Elasticidade, 5. Racionalidade Baseada em Regras

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Agentes Autônomos para Monitoramento e Alocação Dinâmica de Recursos para Nuvem Computacional

Aldo Henrique Dias Mendes

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Célia Ghedini Ralha (Orientadora)
CIC/UnB

Prof.ª Dr.ª Alba Cristina Magalhães Alves de Melo
CIC/UnB

Prof. Dr. Luiz Augusto Fontes Laranjeira
FGA/UnB

Prof.ª Dr.ª Célia Ghedini Ralha
Coordenadora do Mestrado em Informática

Brasília, 17 de março de 2017

Dedicatória

Dedico esta dissertação ao meus pais Antônio e Lourdes, exemplos de tudo o que um dia quero ser.

Dedico também a minha orientadora Prof^{a} Dr^{a} Célia Ghedini Ralha, pela confiança, paciência, amizade e orientação.

Sem o apoio destes, este trabalho jamais teria sido concluído.

Agradecimentos

Agradeço

A Deus, pela vida e oportunidade do estudo.

A toda a minha família pelo apoio incondicional e pelo constante incentivo. Aos meus pais Antônio e Elvira, por me ensinarem a humildade e companheirismo, sendo sempre uma fonte de amor inesgotável. Às tias, tios, primos, primas e todos outros membros da família: muito obrigado.

À minha orientadora Drª Célia Ghedini Ralha, pela confiança depositada em mim, sempre apoiando e aconselhando durante toda jornada do projeto. Obrigado por todo apoio e direcionamento ao longo do processo.

À Thaynara Gomes, minha namorada, por ter suportado toda distância, sabendo lidar com minha grande variação de humor. Obrigado por estar sempre ao meu lado.

Ao amigo Michel Junio, pela parceria de tantos anos de amizade. Por todos os desafios e projetos que encaramos juntos. Agradeço as centenas de vezes que já me socorreu.

À Universidade de Brasília e ao Departamento de Ciência da Computação, que me forneceu um ambiente acolhedor e bem estruturado.

Aos amigos da minha Igreja Apostólica, pelos momentos de alegria e louvores que juntos entoamos a cada apresentação. Obrigado por sempre me apoiarem e compreenderem minha ausência nos últimos tempos.

A todos que de forma direta ou indireta participaram da minha jornada: obrigado, nada conseguiria sem vocês.

"Ama-se mais o que se conquista com esforço."
Benjamin Disraeli

Resumo

A computação em nuvem surgiu como um modelo computacional inovador que permite que usuários acessem recursos com alto poder computacional de forma distribuída e com baixo custo. Uma execução adequado de aplicações em nuvem requer o provisionamento apropriado de recursos. Monitorar tais aplicações para criar históricos de execução é uma alternativa adequada para desenvolver modelos de predição de uso de recursos das máquinas virtuais na nuvem. No entanto, essa abordagem não é trivial quando se deseja viabilizar o provisionamento dinâmico de recursos nas máquinas virtuais. Nesta dissertação foi definido um modelo de monitoramento, predição e provisionamento dinâmico de recursos na nuvem computacional através do uso de um sistema multiagente. Os agentes utilizam raciocínio lógico com regras de inferência através de uma abordagem de interação cooperativa. O modelo foi validado com um estudo de caso utilizando um simulador ambiental denominado MASE-BDI. O modelo de predição com regressão linear múltipla alcançou $96,41\%$ de acerto no uso de CPU e $94,72\%$ no tempo de execução. Os resultados experimentais demonstraram a potencialidade da proposta, uma vez que o uso médio de CPU ficou acima de 76%, além de manter um equilíbrio entre o uso de CPU, o tempo e o custo das execuções.

Palavras-chave: Computação em Nuvem, Sistema Multiagente, Predição de Recursos, Elasticidade, Racionalidade Baseada em Regras

Abstract

Cloud computing has emerged as an innovative computing model that allows ordinary users to access distributed computing resources with low cost. Monitoring the applications to create historical execution records in the cloud is an adequate approach to develop prediction models. Nevertheless, this is not a trivial approach when the intention is to allow dynamic provisioning of resources. This Msc dissertation proposes a multi-agent system to monitor, predict and dynamic provisioning of resources in the cloud in a transparent way, assuring elasticity and a better use of allocated resources. Agents use logical reasoning with inference rules through a cooperative interaction approach. The model was validated in a case study with the MASE-BDI environmental simulator. The prediction model using multiple linear regression achieves 96.41% hit of CPU use and 94.72% of execution time. The experimental results demonstrate the potential of the approach since the medium CPU use is over 76%, keeping the balance among the CPU use, time and cost of the executions.

Keywords: Cloud Computing, Multiagent Systems, Resource Prediction, Elasticity, Rule-based rationality

Abreviaturas

AgentSpeak - *Agent-Oriented Programming Language.*

AM - Agente de Monitoramento.

API - *Application Programming Interface.*

BC - Base de Conhecimento.

BDI - Modelo de raciocínio *Belief, Desire, Intention.*

BRMS - *Business Rules Management System.*

CLIPS - *C Language Integrated Production System.*

CPU - Unidade Central de Processamento (*Central Processing Unit*).

CPUUsed - Uso Médio de CPU.

CPUUsedLog - *Uso Médio de CPU dado em Logaritmo.*

CSV - *Comma-Separated Values.*

DAML - *DARPA Agent Markup Language.*

DRPMC - *Dynamic Resource Provisioning and Monitoring In Cloud.*

Dstat - *Versatile Resource Statistics Tool.*

FIPA - *Foundation for Intelligent Physical Agents.*

FIPA ACL - *FIPA Agent Communication Language.*

FNC - Forma Normal Conjuntiva.

GM - Gerente de Monitoramento.

GMV - Gerente de Máquinas Virtuais.

GRIDM - *GRID Manager.*

HD - Disco Rígido (Hard Disk).

HFD - Host Fault Detection. (*Container Table*).

IaaS - Infrastructure as a Service.

IoStat - Input/Out-put Statistics for Devices and Partitions.

JADE - *Java Agent Development Framework.*

JADEX - JADE *eXtension.*

Jason - *Java-based interpreter for an extended version of AgentSpeak.*

JEOPS - *The Java Embedded Object Production System.*

JESS - *Java Expert System Shell.*

JVM - Máquina Virtual Java (*Java Virtual Machine*).

KQML - *Knowledge Query and Manipulation Language.*

KSE - *Knowledge Sharing Effort.*

MAPE - *Mean Absolute Percent Error.*

MASE - *Multi-Agent System for Environmental Simulation.*

MASE-BDI - MASE com Modelo *Belief-Desire-Intention.*

MF - Máquina Física.

MpStat - *Report processors related statistics.*

MTP - Protocolos de Transporte de Mensagens (*Message Transport Protocols*).

MTS - Sistema de Transporte de Mensagens (*Message Transport System*).

MV - Máquina Virtual.

OWL - *Web Ontology Language.*

PaaS - *Platform as a Service.*

PAGE - *Perceptions, Actions, Goals and Environment.*

PS - Parâmetro Significativo.

QC - Quantidade de CPU.

QoS - Qualidade de Serviço (*Quality of Service*).

RLM - Regressão Linear Múltipla.

RLS - Regressão Linear Simples.

RN - Redes Neurais.

RDFS - *Resource Description Framework Schema.*

SaaS - *Software as a Service.*

SeSam - *Shell for Simulated Agent Systems.*

SL - *Semantic Language.*

SLA - Acordo de Nível de Serviço (*Service Level Agreement*).

SMA - Sistema Multiagente.

SPM - *Spatial Manager.*

SVM - *Support Vector Machines.*

TempoLog - Tempo Dado em Logaritmo.

TRA - *Transformation Agent.*

TRM - *Transformation Manager.*

vCPU - CPU virtual (*virtualCPU*).

XML - *eXtensible Markup Language.*

Sumário

Lista de Figuras **xiv**

Lista de Tabelas **xv**

1 Introdução **1**
1.1 Caracterização do Problema . . . 2
1.2 Objetivos . . . 2
1.3 Metodologia e Estrutura do Documento . . . 3

2 Fundamentação Teórica **5**
2.1 Sistemas Multiagentes . . . 5
2.1.1 Agentes . . . 5
2.1.2 Ambiente . . . 6
2.1.3 Comunicação . . . 7
2.1.4 Funções dos Agentes . . . 10
2.1.5 Coordenação de Agentes . . . 14
2.1.6 Protocolo de Interação . . . 14
2.1.7 Ferramentas para Desenvolvimento de SMA . . . 15
2.2 Agente com Dedução Lógica . . . 17
2.2.1 Representação do Conhecimento . . . 18
2.2.2 Raciocínio Lógico Baseado em Regras . . . 18
2.2.3 Motores de Inferência Baseados em Regras . . . 19
2.3 Computação em Nuvem . . . 21
2.3.1 Modelos de Implantação . . . 22
2.3.2 Arquitetura de Nuvem . . . 23
2.3.3 Modelos de Serviços . . . 23
2.3.4 Características Essenciais . . . 25
2.3.5 Monitoramento . . . 26
2.3.6 Provisionamento Dinâmico de Recursos . . . 27
2.3.7 Predição de Uso de Recursos . . . 27
2.3.8 Elasticidade . . . 32

3 Trabalhos Correlatos **35**

4 Desenvolvimento da Solução **39**
4.1 Arquitetura Geral . . . 39
4.2 Descrição dos Agentes . . . 42

4.2.1 Gerente de Máquinas Virtuais - GMV 42
4.2.2 Gerente de Monitoramento - GM 45
4.2.3 Agente de Monitoramento - AM 46
4.3 Regras de Inferência 47
4.3.1 Regras de Provisionamento 47
4.3.2 Regra de Elasticidade 51
4.4 MASE-BDI 54
4.5 Experimentos Iniciais 56

5 Experimentos e Resultados **59**
5.1 Predição de Recursos 59
5.1.1 Resultados da Regressão Linear Múltipla 59
5.1.2 Estimativas de Tempo e Uso de CPU 62
5.2 Provisionamento Dinâmico 65
5.2.1 Resultados da Elasticidade 66
5.2.2 Resultados da Recuperação da Aplicação 68

6 Conclusões e Trabalhos Futuros **70**

Referências **72**

Lista de Figuras

2.1 Arquitetura geral de um agente (traduzida de [9]). 6
2.2 Arquitetura de agente reativo simples (traduzida de [9]). 11
2.3 Arquitetura de agente reativo baseado em modelo (traduzida de [9]). . . . 12
2.4 Arquitetura de agente baseado em objetivos (traduzida de [9]). 12
2.5 Arquitetura de agente reativo baseado em utilidade (traduzida de [9]). . . . 13
2.6 Arquitetura de agente com aprendizagem (traduzida de [9]). 13
2.7 Arquitetura de agente baseado em conhecimento [9]. 17
2.8 Arquitetura básica de regras de produção. 20
2.9 Arquitetura de computação em nuvem [39]. 24
2.10 Esquema de classificação de SVM, adaptado de [54]. 29
2.11 Esquema de uma rede neural, adaptado [57]. 30
2.12 Regressão linear [58]. 30
2.13 Classificação de elasticidade (traduzida de [67]). 33

4.1 Esquema geral da arquitetura contendo: (a) arquitetura modular dos serviços oferecido; e (b) módulos da arquitetura. 40
4.2 Arquitetura geral com o *workflow* de execução dos módulos. 41
4.3 Módulo de elasticidade horizontal. 44
4.4 Esquema de recuperação da aplicação. 45
4.5 Colunas dos dados capturados pelo Dstat. 46
4.6 Regra de Inferência para o GMV realizar o provisionamento. 49
4.7 A segunda regra de provisionamento com base no dados do provedor. . . . 50
4.8 Regra de inferência para o GM realizar a elasticidade baseada nos dados monitorados. 53
4.9 Visão geral da simulação MASE-BDI [7]. 54
4.10 Arquitetura MASE-BDI [7]. 55
4.11 Execuções do MASE-BDI em diferentes MVs. 57

5.1 Distribuição dos dados da média de uso do CPU. 60
5.2 Distribuição dos dados da média do tempo. 61
5.3 Gráficos dos resíduos. 62
5.4 Gráficos dos resíduos. 63
5.5 Gráficos da regressão do uso de CPU. 64
5.6 Gráficos da regressão do tempo. 64
5.7 Comparação entre o uso de CPU após elasticidade. 67

Lista de Tabelas

2.1 PAGE de um agente táxi automatizado. 6
2.2 Classificação do ambiente para o táxi automatizado. 7
2.3 Performativas de comunicação da FIPA ACL. 9
2.4 Resumo das ferramentas de desenvolvimentos de SMA. 16
2.5 Características dos encadeamentos progressivo e regressivo. 20

3.1 Trabalhos Correlatos 38

4.1 Relação entre a quantidade de CPU e quantidade mínima de memória fornecida pelo *Google Cloud Platform* 43
4.2 Exemplo de entradas para regra de provisionamento do agente GMV. . . . 48
4.3 Limites de agentes e uso de CPU 48
4.4 Exemplo de entradas para segunda regra de provisionamento do MASE-BDI. 51
4.5 Exemplo de entradas para regra de elasticidade do MASE-BDI. 54
4.6 Resultados simulação MASE-BDI [7]. 56

5.1 Comparação de R^2 para transformações de variáveis. 60
5.2 Resultado dos índices de avaliação. 65
5.3 Resultados de execuções das escolhas do GMV. 66
5.4 Resultado dos Experimentos com Elasticidade. 67
5.5 Resultado dos Experimentos da Recuperação da Aplicação. 68

Capítulo 1

Introdução

A expansão das tecnologias envolvidas com a Internet fez surgir novas ferramentas e modelos computacionais que de alguma forma respondessem as necessidades dos usuários. Um dos principais modelos é a Computação em Nuvem, que apresenta soluções que envolvem transparência, escalabilidade, elasticidade, entre outras várias vantagens. Além disso, computação em nuvem traz maior controle na utilização dos recursos e liberdade para o usuário decidir o quanto e quando usar recursos e serviços contratados. O desenvolvimento de aplicações voltadas para o ambiente de nuvem computacional deve buscar melhoria da utilização dos recursos, de modo a utilizar somente o necessário de forma inteligente. Assim, as aplicações devem ter uma arquitetura centrada na melhoria do uso dos recursos providos pela nuvem otimizando tempo e recursos [1].

O rápido crescimento da Computação em Nuvem ocorre por várias razões, mas é possível classificá-las em três principais categorias [2]. A primeira é pelo mercado, formado pelas grandes empresas de tecnologias (Google, Amazon, Microsoft, entre outras) uma vez que suas pesquisas e trabalhos têm interesse econômico oferecido por esse paradigma. Nesse novo modelo o cliente paga pelo que usa (*pay-per-use*) e a empresa pode redistribuir seu poder computacional entre vários clientes. O segundo motivo para justificar o rápido crescimento foi a maturidade dos componentes tecnológicos internos a nuvem, como *hardwares*, servidores, *clusters* de alta disponibilidade, computação em *grid* e tecnologias de virtualização, em que é baseado o paradigma computacional da nuvem. A aceitação positiva do público é outro fator fundamental para seu desenvolvimento rápido. Com a possibilidade do usuário ter acesso a seus dados, a qualquer momento em qualquer lugar, transferindo para o provedor a responsabilidade pela disponibilidade e segurança, aumentou a aceitação do público geral. A exemplificar, empresas e *startups* que estão se estabelecendo no mercado não precisam adquirir e gerenciar grande poder computacional físico, uma vez que podem contratar o mesmo poder em uma provedora de nuvem com menor custo financeiro [3] [2].

Segundo [4] [5], a Computação em Nuvem é composta algumas por características essenciais. Essas características dizem quais elementos são fundamentais e pontos nos quais se deve centrar a atenção dos projetistas ao desenvolver modelos (arquiteturas, aplicações, entre outros) para ambiente de nuvem: (1) os recursos sob demanda estão ligados ao fornecimento de poder de processamento, armazenamento e memória e podem ser adquiridos unilateralmente de acordo com as solicitações do usuário; (2) o amplo acesso à rede permite ao usuário ter acesso aos serviços a qualquer momento e em qualquer

dispositivo; (3) *pooling* de recursos, no qual os provedores agrupam os recursos para servir múltiplos usuários, ajustando de acordo com a demanda; (4) elasticidade, que permite adquirir recursos de forma rápida e dinâmica, muitas vezes automaticamente; e (5) serviço medido, em que os provedores controlam e otimizam a utilização de recursos por meio do monitoramento.

Com base no monitoramento dos recursos, um gestor ou projetista pode ajustar a alocação de recursos para cada perfil de usuário, buscando otimizar a distribuição dos recursos disponíveis para um maior número de usuários. Tomadas de decisões para garantir serviços como elasticidade e provisionamento dinâmico de recursos levam em consideração os dados capturados no monitoramento. Com um maior grau de inteligência nas tomadas de decisões, foi obtido ainda mais dinamismo.

Sistemas Multiagentes (SMA) uma sub-área de Inteligência Artificial, o uso de agentes inteligentes é uma forma de solução para tomadas de decisões em ambientes complexos e dinâmicos. O uso SMA em ambientes com essas características tem muitas vantagens, uma vez que pela sua própria definição, o comportamento e ações dos agentes são baseados no que eles percebem do ambiente [6].

1.1 Caracterização do Problema

Pela incerteza das necessidades de recursos que uma aplicação pode precisar, a possibilidade de um modelo que possa predizer e estender recursos, em tempo de execução, torna-se muito importante. Isso permitirá que a aplicação submetida continue sua execução em condições não contratadas, mesmo que seu comportamento não esteja de acordo com o que foi alocado inicialmente.

Para que sejam executadas as aplicações são criadas instâncias de máquinas virtuais (MV). Em casos de complicações em alguma das execuções faz-se necessário recriar novas instâncias de MV e nesses casos é importante que as execuções já concluídas não sejam perdidas e que seus resultados sejam mantidos.

Conforme o cenário apresentado, pode-se citar como questão de pesquisa: é possível definir um modelo de monitoramento e alocação dinâmica de recursos para execução do MASE-BDI em nuvem computacional, utilizando agentes inteligentes que deliberem de maneira autônoma sob o desperdício de recursos?.

Neste sentido, este trabalho assume como hipótese que é possível definir com agentes de dedução lógica um modelo de alocação de MV baseado em histórico de utilização dos recursos para otimizar a predição de tempo e uso de recursos e o provisionamento dinâmico na execução do MASE-BDI na nuvem.

1.2 Objetivos

O objetivo geral desse trabalho é a definição e implementação de uma solução SMA inteligente validada em ambiente real que utilize agentes autônomos para o monitoramento e alocação dinâmica de recursos em nuvem computacional, visando executar o MASE-BDI com o menor custo e tempo, minimizando o desperdício de recursos.

Para alcançar o objetivo geral descrito podem ser citados os seguintes objetivos secundários:

1. definir um modelo de autonomia dos agentes para execução do monitoramento e alocação dinâmica de recursos em nuvem;
2. determinar um modelo de predição de uso recursos por meio do histórico e comportamento do MASE-BDI.

Conforme os objetivos descritos, a presente dissertação de mestrado apresenta um modelo de agentes inteligentes que realiza monitoramento, alocação dinâmica e elasticidade para aplicações submetidas em ambiente de nuvem. Busca garantir através das características do ambiente que as aplicações sejam executadas e concluídas de forma otimizada.

Para validar o modelo proposto foi utilizada uma ferramenta desenvolvida no Departamento de Ciência da Computação da UnB, o *MultiAgent System for Environmental Modeling* (MASE). O MASE é um simulador ambiental de uso e cobertura da terra baseado em um modelo de agentes inteligente. O MASE tem uma segunda versão que utiliza um modelo Belief-Desire-Intention no raciocínio dos agentes, a qual foi denominada MASE-BDI [7] [8]. No MASE-BDI o comportamento interno dos agentes é não determinístico, necessitando de recursos de infraestrutura de forma dinâmica, o que aumenta o desafio de monitoramento e predição de recursos na nuvem computacional.

1.3 Metodologia e Estrutura do Documento

Este trabalho de mestrado tem natureza empírica com propósito de pesquisa exploratória e abordagem qualitativa. Foi feita uma pesquisa bibliográfica e realizado um estudo de caso. Uma investigação sobre SMA e Computação em Nuvem foi realizada, para que se pudesse definir um modelo arquitetural baseado em agentes para realizar os serviços de nuvem de forma cooperativa. Paralelamente, *frameworks* de SMA foram pesquisados com o propósito de buscar as ferramentas adequadas que atendessem as necessidades desejadas para o modelo. Foi realizado um estudo dos conceitos relacionados à Computação em Nuvem, com o objetivo de identificar características e serviços ofertados nesse ambiente incluindo a visão do estado da arte. Após identificados, foram definidos comportamentos necessários para executar as aplicações em nuvem desfrutando dos principais benefícios encontrados.

Na sequência, foi implantado um protótipo com o modelo de agentes autônomos definido para garantir o monitoramento e a alocação dinâmica de recursos em nuvem computacional. O estudo de caso para validação da proposta utilizou uma ferramenta desenvolvida para *desktop* na linguagem JAVA, denominado MASE-BDI [7]. O MASE-BDI foi a ferramenta escolhida por ser não determinística e suas execuções são dinâmicas, tanto para o tempo de execução quanto para ao uso de recursos alocados.

Os resultados das execuções do MASE-BDI em um ambiente de nuvem foram avaliados segundo metodologia empírica de seu comportamento com execuções em MVs na nuvem. Os resultados do tempo, custo e utilização dos recursos de um conjunto de execuções do MASE-BDI são comparados com e sem o modelo desenvolvido. Por meio dos resultados foi possível comparar os ganhos com as execuções utilizando o modelo, possibilitando responder a questão de pesquisa inicialmente proposta.

Conforme os objetivos definidos considera-se que a principal contribuição dessa dissertação é a definição de um modelo de agentes inteligentes, que utiliza regras de inferência

para garantir o monitoramento, provisionamento dinâmico e elasticidade horizontal para execução de aplicação em nuvem computacional. Foi definido uma arquitetura e implementado um protótipo para que a aplicação que antes não tinha seu foco em nuvem computacional, agora possam ser executada em um ambiente de nuvem obtendo todos os benefícios da mesma. Para alcançar os objetivos descritos serão comparados os experimentos com trabalhos correlatos da literatura.

Conforme a metodologia descrita foi redigida essa dissertação em seis capítulos. No Capítulo 2 se faz uma apresentação em uma revisão dos principais conceitos sobre SMA, modelo de Racionalidade em regras e Computação em Nuvem. Conceitos de serviços em nuvem computacional também são reunidos nesse capítulo, uma vez que são abordados serviços específicos da nuvem nesse trabalho.

No Capítulo 3 são apresentados os trabalhos correlatos, comparando os pontos em que existe convergência e nos quais se distanciam da presente dissertação.

No Capítulo 4 se faz a apresentação da proposta de solução do trabalho, sendo descrita a arquitetura geral do modelo, assim como cada camada do modelo implementado, de forma que se faz a apresentação e descrição dos agentes e as regras de inferência utilizadas para implementar a racionalidade de cada agente.

No Capítulo 5 são mostrados os experimentos realizados com a aplicação submetida em diferentes cenários de MV em nuvem. Ainda se apresentam os experimentos com a aplicação submetida sendo executada em um ambiente de nuvem, com e sem o modelo proposto nesse projeto. Os resultados também são apresentados como forma de expressar os acertos do modelo.

Por fim, no Capítulo 6 são apresentadas as conclusões e os possíveis caminhos para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

São apresentados neste capítulo conceitos de SMA e Computação em Nuvem, áreas emergentes de grande importância para pesquisa e para o mercado. É apresentada a base teórica para o desenvolvimento de SMA e o formalismo para representação do conhecimento baseado em regras. São mostrados os serviços básicos e as principais características de uma nuvem.

2.1 Sistemas Multiagentes

Inteligência Artificial (IA) é a área da Ciência da Computação que aborda questões de automação do raciocínio. As definições de IA vão além desta, a exemplificar tem-se a ideia de um sistema que realiza tarefas e resolve problemas de forma racional, assimilando-se a inteligência humana [9]. Para dizer que tem uma forma racional, um sistema inteligente pode tratar incertezas, inferências e reflexos.

Ainda em IA, existe uma categoria de sistemas que buscam soluções por meio de entidades distribuídas. SMA é uma subárea de IA distribuída. O estudo e uso de SMA em contextos reais e fictícios têm se tornado cada vez maior e mais importante. Um dos principais motivos para essa importância é o fato dos agentes possibilitarem determinar comportamentos e tomar decisões baseados no que se percebe do ambiente [6]. Além de conseguirem reproduzir comportamentos e características heterogênicas, em que é necessário um gerenciamento maior na tomada de decisões ao buscar soluções de problemas.

2.1.1 Agentes

Agentes são entidades computacionais capazes de perceber seu ambiente por meio de sensores e agir sobre ele por intermédio de atuadores, como ilustrado na Figura 2.1 [9]. Segundo [10], um agente é uma entidade computacional situada em um ambiente em que é capaz de realizar ações de forma autônoma para atingir seus objetivos. O projeto de um agente pode ser representando pelo PAGE (*Perceptions*, *Actions*, *Goals* e *Environment*). Na Tabela 2.1 é apresentado o exemplo de PAGE do agente táxi automatizado [9]. O agente apresentado tem os objetivos de buscar e levar os passageiros em segurança, maximizar lucros e obedecer as leis de trânsito, de forma que as percepções ocorrem por meio de imagens de vídeo, acelerômetros, medidores, sensores de motor e volante. Pela própria definição do agente, suas ações são as de um motorista de táxi, tais como dirigir, acelerar,

frear, buzinar e falar. Todas suas ações e comportamentos estão dentro do ambiente no qual um taxista está inserido, ruas, parques e rodovias.

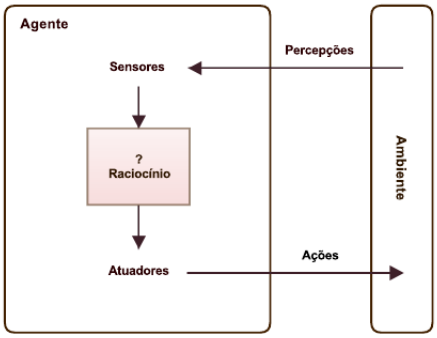

Figura 2.1: Arquitetura geral de um agente (traduzida de [9]).

Tabela 2.1: PAGE de um agente táxi automatizado.

Percepções (*Perceptions*)	Vídeo Acelerômetros Medidores Sensores do motor Volante
Ações (*Actions*)	Dirigir Acelerar Frear Buzinar Falar
Objetivos (*Goals*)	Chegar ao destino com segurança Maximizar lucros Obedecer leis de trânsito Manter os passageiros confortáveis
Ambiente (*Environment*)	Ruas Parques Rodovias

Segundo [6], um agente é um sistema computacional capaz de agir de maneira autônoma e flexível em um ambiente. Para isso é necessário apresentar características de (1) reatividade – perceber o ambiente e responder a mudanças; (2) proatividade – decidir por si mesmo o curso de ação para atingir seus objetivos; e (3) sociabilidade - interagir com os demais agentes em situações complexas por meio de negociação, cooperação e coordenação.

Segundo [9], existem cinco tipos de agentes: agentes reativos simples, agentes reativos baseados em modelos, agentes baseados em objetivos, agentes baseados na utilidade e agentes com aprendizagem apresentados na Seção 2.1.4.

2.1.2 Ambiente

Tão importante quando o agente é o ambiente em que o mesmo estará inserido. É fundamental que seja definido, tanto ambientes virtuais quanto ambientes físicos. O agente

irá utilizar seus sensores para perceber o ambiente, tomar decisões e realizar suas tarefas. De acordo com [9] é possível classificar um ambiente de acordo com suas características. O ambiente pode ser classificado dentro de cinco características, determinando seu tipo:

1. Determinístico x Estocástico: um ambiente é determinístico quando o estado atual e as ações dos agentes influenciam nos próximos estados, ou seja, previsível; senão, é estocástico. Um ambiente é determinístico ou estocástico de acordo com o ponto de vista de um agente;
2. Episódico x Sequencial: em ambientes episódicos os estados são divididos em forma segmentada e atômicos, tornando-os simples, uma vez que o agente não se preocupa com o futuro. Por sua vez, um ambiente sequencial a decisão dos agentes poderá afetar todas as decisões futuras;
3. Totalmente observável x Parcialmente observável: no ambiente totalmente observável os sensores do agente têm acesso total ao ambiente todo o tempo. No Ambiente parcialmente observável os sensores não conseguem ter acesso total ao ambiente, isso pode acontecer pela complexidade do ambiente ou por limitação dos sensores;
4. Dinâmico x Estático: se o ambiente muda, enquanto o agente está realizando suas ações, pode-se dizer que ele é dinâmico, senão é estático. Se o ambiente não muda com a passagem do tempo, mas a performance do agente sim, então ele é semi-dinâmico;
5. Discreto x Contínuo: em ambiente discreto existe um número distinto e visivelmente definido de percepção e ação em cada estado do ambiente. Em ambientes contínuos as percepções e ações mudam em um espectro contínuo de valores.

Ainda, seguindo o exemplo do agente taxista, na Tabela 2.2 são apresentadas as características para o ambiente do táxi automatizado.

Tabela 2.2: Classificação do ambiente para o táxi automatizado.

Característica	**Valor**
Determinístico x Estocástico	Estocástico
Episódico x Sequencial	Sequencial
Totalmente observável x Parcialmente observável	Parcialmente observável
Dinâmico x Estático	Dinâmico
Discreto x Contínuo	Contínuo

Um carro em ambiente urbano pode percorrer por toda cidade (ambiente), mas a percepção dele é limitada aos sensores ativos. Dessa forma, a percepção do taxista é limitada ao espaço e tempo de captura de suas percepções. Então, o agente taxista automatizado tem seu acesso parcial ao ambiente. O trânsito possui uma grande quantidade de variáveis, tais como: pedestres, sinalização, acidentes entre outros, ou seja, o ambiente é não determinístico, dinâmico, é sequencial e não episódico.

2.1.3 Comunicação

Em [9] encontra-se a definição de comunicação, sendo a troca intencional de informações provocada pela percepção de um sistema compartilhado. Assim, a linguagem é um sistema

de mensagem crucial na representação do conhecimento, e está fortemente centrada na IA, além disso segue uma determinada estrutura. Quando o agente produz uma linguagem, esta se apresenta como um ato comunicativo.

Segundo [11], a comunicação é primordial no desenvolvimento de SMA. Muitas vezes, para um agente alcançar seus objetivos é necessário se comunicar com outros agentes, não apenas com agentes de software, a comunicação é importante para com o usuário e outros sistemas. A comunicação é fundamental para que os agentes cooperem, colaborem ou negociem.

A interação entre agentes por meio de uma linguagem se denomina, de modo geral, como linguagem de comunicação de agentes. No governo americano por volta dos anos 90 surgiu a *Knowledge Sharing Effort* (KSE), responsável por desenvolver várias linguagens de comunicação de agentes. Uma das responsabilidades do grupo é desenvolver protocolos para comunicação entre sistemas de informação autônomos. A primeira linguagem de comunicação com grande impacto e amplitude foi desenvolvida pela KSE, chamada de *Knowledge Query and Manipulation Language* (KQML) [12].

A linguagem KQML permite realizar troca de informação e conhecimento baseado em mensagens. O formato da mensagem é definido de uma forma que o conteúdo não é relevante, a linguagem se preocupa com a estrutura do protocolo. Por fim, a linguagem KQML pode ser definida com uma ação performática e um grupo de parâmetros.

Performativas são enunciados que não podem ser classificados como verdadeiros ou falsos. Segundo [13], as performativas da linguagem se classificam em cinco classes:

- Representativas - expressam uma crença do locutor em relação ao receptor, ou seja, o locutor comunica que acredita na verdade do enunciado;
- Comissivas - expressam promessas. Mostra compromisso do locutor com um ação futura;
- Declarativas - afirma fatos ao mesmo tempo que altera o estado do ambiente;
- Diretivas: expressam um pedido ou comando. É a performativa utilizada para solicitações;
- Expressivas - expressam desculpas, agradecimentos ou estados psicológicos;
- Vereditos - expressam um julgamento.

A linguagem mais citada nas referências para comunicação de agentes é a FIPA-ACL - *Agent Communication Language* [1]. A *Foundation for Intelligent Physical Agents* (FIPA), fundada em 1996, desenvolve padrões para sistemas baseados em agentes. Ela padronizou a ACL para uso em sistemas baseados em agentes, assim como definiu outros padrões de interação.

As performativas de comunicação FIPA-ACL, distribuídos em suas categorias, são apresentados na Tabela 2.3.

[1]FIPA ACL Message Structure Specification - http://www.fipa.org/specs/fipa00061/

Tabela 2.3: Performativas de comunicação da FIPA ACL.

Performativos	**Descrição**
accept-proposal	A ação de aceitar um *propose* previamente submetido para executar uma ação.
agree	A ação de concordar em executar uma *request* feita por outro agente.
cancel	O agente quer cancelar um *request* anterior.
cfp	O agente emite um convite à apresentação de propostas. Contém as ações a serem realizadas e quaisquer outros termos do contrato.
confirm	O remetente confirma ao receptor a veracidade do conteúdo.
disconfirm	O remetente confirma ao destinatário a falsidade do conteúdo.
failure	Informe o outro agente de que um *request* anterior falhou.
inform	Informa algo a outro agente. O remetente deve acreditar na verdade da declaração.
inform-if	Usado como conteúdo de *request* para confirmar se uma declaração é verdadeira ou falsa.
inform-ref	Como *inform-if*, mas pede o valor da expressão.
not-understood	Enviado quando o agente não entendeu a mensagem.
propagate	Solicita que outro agente transmita a mesma mensagem para os outros.
propose	Usado como uma resposta a *cfp*. O agente propõe um acordo.
proxy	O emissor deseja que o receptor selecione os agentes de destino indicados por uma determinada descrição e envia uma mensagem a eles.
query-if	A ação de perguntar a outro agente se uma determinada proposição é verdadeira ou falsa.
query-ref	A ação de pedir a um outro agente um dado objeto referenciado por uma expressão.
refuse	Recusa a execução uma determinada ação, explicando o motivo da recusa.
reject-proposal	A ação de rejeitar uma proposta para realizar alguma ação durante uma negociação.
request	O remetente solicita que o receptor execute alguma ação.
request-when	O remetente quer que o receptor execute alguma ação, quando alguma proposição dada se torna verdadeira.
request-whenever	O remetente quer que o receptor execute alguma ação assim que alguma proposição se tornar verdadeira e, depois disso, cada vez que a proposição se tornar verdadeira executa a ação novamente.
subscribe	O ato de solicitar a intenção de receber notificações, quando um valor de referência for modificado e de notificar novamente sempre que o objeto identificado pela referência muda.

A FIPA padronizou uma semântica formal para a comunicação entre agentes. A semântica criada faz relação a uma linguagem formal chamada *Semantic Language* (SL).

Essa linguagem permite representar crenças, desejos e incertezas dos agentes, assim como as ações que o agente pode executar. A semântica da linguagem da FIPA mapeia cada mensagem ACL a uma fórmula SL, definindo um padrão de restrição que o remetente precisa seguir caso queira utilizá-la como protocolo de comunicação.

2.1.4 Funções dos Agentes

Um agente é uma entidade computacional que percebe o ambiente, toma decisões e podem realizar ações sobre o ambiente. Essa entidade computacional pode ser descrita por funções. Essas funções fazem o mapeamentos de percepções em ações, que podem variar de acordo com o nível de racionalidade do agente. No Algoritmo 2.1 está apresentado uma simples representação do mapeamento. O agente tem uma percepção externa como entrada, escolhe a melhor ação e retorna a ação escolhida. A memória é atualizada após a ação realizada com os resultados.

Algoritmos 2.1: Função de mapeamento ação-percepção de um agente (adaptado de [9]).

```
1  funcao Agente_Esqueleto (percecao) retorna acao
2  estatico: memoria ← a memoria do agente sobre o mundo
3
4  memoria ← ATUALIZA_MEMORIA(memoria, percepcao)
5  acao ← ESCOLHE_MELHOR_ACAO(memoria)
6  memoria ← ATUALIZA_MEMORIA(memoria, acao)
7  retorna acao
```

A inteligência de um agente está ligada a seu conhecimento, seus recursos computacionais e sua percepção [14]. [9] classifica os agentes em quatro categorias de acordo com o seu conjunto de atributos:

1. Agentes Reativos Simples: racionalidade mais simples de todas as categorias. Suas ações são mapeadas diretamente por meio de estruturas condicionais. Realiza ações baseadas no reflexo, como fechar os olhos, quando uma luz forte está acesa perto do mesmo, Algoritmo 2.2. Não são salvos em memória os resultados de seu comportamento, apenas segue as regras implementadas, conforme a Figura 2.2;

2. Agentes Reativos com Registro de Estados: também reflexivo baseado em estruturas condicionais, mas leva em consideração também o estado atual do agente. Leva em consideração uma máquina de estado interno para encontrar a melhor ação, Algoritmo 2.3. O estado futuro é guardado de acordo com a ação e o estado anterior, conforme a Figura 2.3;

3. Agentes Orientados a Objetivos: Com um raciocínio mais elaborado, escolhe suas ações levando a alcançar um ou vários objetivos. Junto aos estados, suas decisões buscam ações que podem modificar o ambiente externo (Algoritmo 2.4). A ação é eleita dentre um conjunto de possíveis ações. Os objetivos podem ser atualizados de acordo com as mudanças no ambiente, os objetivos já alcançados e o estado do próprio agente, conforme a Figura 2.4;

4. Agentes Orientados a Utilidades: outras variáveis alheias ao ambiente são levadas em consideração, além de considerar os objetivos e ponderar a melhor ação a tomar.

A escolha da melhor ação pode ser algo que tem influência no raciocínio do agente, por exemplo a ação escolhida pode ser a mais econômica (Algoritmo 2.5). Muito parecido com o agente orientado a objetivos, se diferindo pelo uso de uma métrica de utilidade do agente, que influencia a escolha da ação e atualização dos objetivos, conforme a Figura 2.5.

5. Agentes com Aprendizagem: podem atuar em ambientes totalmente desconhecidos e se tornarem mais eficientes de acordo com seu conhecimento sobre o ambiente. Em agentes sem aprendizagem, tudo o que o agente sabe foi colocado nele pelo projetista, diferente de agentes com aprendizagem em que o agente eleva o seu saber conforme descobre o mundo, conforme a Figura 2.6.

Algoritmos 2.2: Função de mapeamento ação-percepção de um agente reflexivo simples.

```
1  funcao Agente_Reflexivo_Simples (percecao) retorna acao
2
3  se percepcao = percepcao1 entao acao ← acao1;
4  ...
5  senao se percepcao = percapcao2 entao acao ← acao2;
6  ...
7  senao
8       acao ← acaoN
9  retorna acao
```

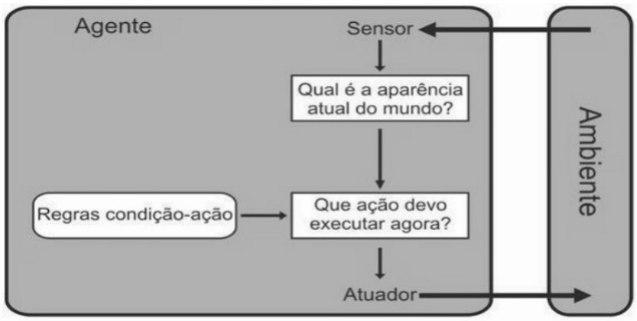

Figura 2.2: Arquitetura de agente reativo simples (traduzida de [9]).

Algoritmos 2.3: Função de mapeamento ação-percepção de um agente reflexivo com registro de estado.

```
1  funcao Agente_Reflexivo_Com_Estados (percecao) retorna acao
2  estatico: estado
3
4  acao ← Elscolher_Acao_Conforme_Estado_E_Percepcao(estado,percepcao);
5  estado ← Atualizar_Estado(acao,estado);
6  retorna acao
```

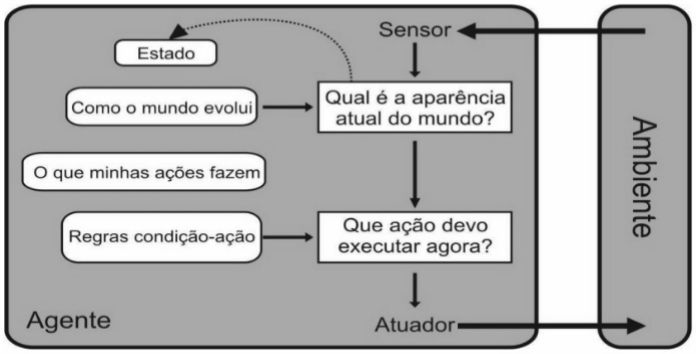

Figura 2.3: Arquitetura de agente reativo baseado em modelo (traduzida de [9]).

Algoritmos 2.4: Função de mapeamento ação-percepção de um agente orientado a objetivos.

```
1  funcao Agente_Orientado_A_Objetivos (percecao) retorna acao
2  estado: estado, objetivos
3
4  acao ← Eleger_Melhor_Acao(estado,percepcao,objetivos);
5  estado ← Atualizar_Estado(acao,estado);
6  objetivos ← Atualizar_Objetivos(estado,objetivos,percepcao);
7  retorna acao
```

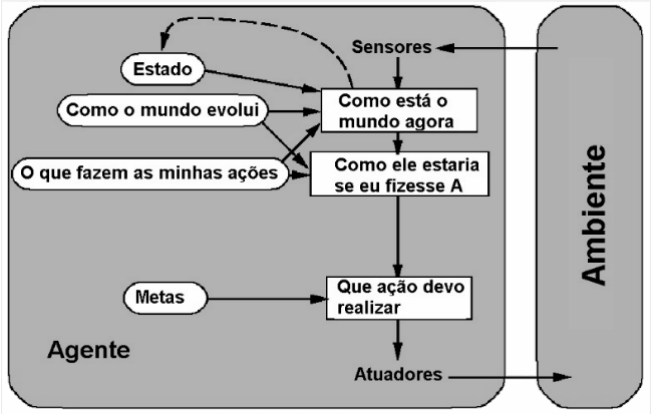

Figura 2.4: Arquitetura de agente baseado em objetivos (traduzida de [9]).

Algoritmos 2.5: Função de mapeamento ação-percepção de um agente orientado a utilidades.

```
1   funcao Agente_Orientado_A_Utilidades (percecao) retorna acao
2   estado: estado, objetivos
3
4   acao ← Eleger_Melhor_Acao(estado,percepcao,objetivos);
5   estado ← Atualizar_Estado(acao,estado);
6   objetivos ← Atualizar_Objetivos(estado,objetivos,percepcao,utilidade);
7   retorna acao
```

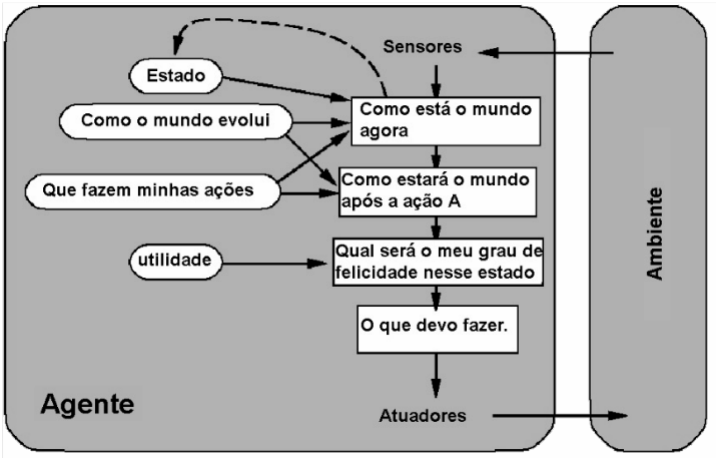

Figura 2.5: Arquitetura de agente reativo baseado em utilidade (traduzida de [9]).

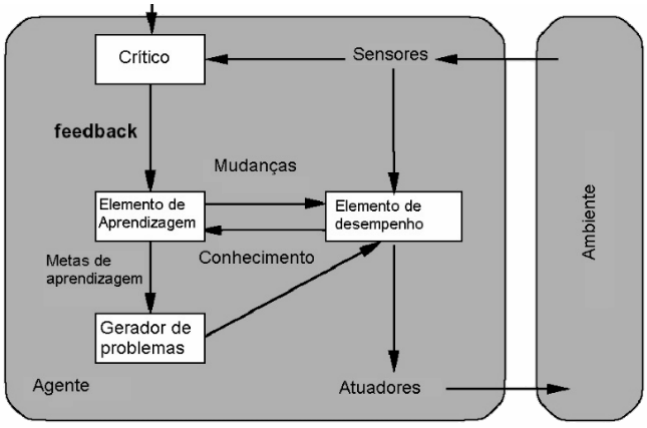

Figura 2.6: Arquitetura de agente com aprendizagem (traduzida de [9]).

Um ambiente multiagente prevê a interação entre os agentes, seja para cooperar ou para competir. Em geral, as suas interações são em prol de algum objetivo e/ou por algum recurso. É interessante que os agentes realmente tenham tarefas em comum, ou que possam se comunicar por meio de troca de mensagens, caso contrário não pode ser considerado um sistema multiagente, é apenas um sistema de agentes com tarefas distintas. Outra característica importante na ocorrência de agentes competitivos é a necessidade da existência de regras de negociação para resolver os conflitos [15].

As interações entre os agentes acontecem por meio de protocolos que definem como deve ocorrer a sintaxe, semântica e sincronização da comunicação. Esses protocolos utilizam linguagens de comunicação e interação. Os protocolos, em geral, dependem de qual framework o desenvolvedor utilizará para desenvolver o sistema multiagente [16].

2.1.5 Coordenação de Agentes

A coordenação de agentes é um aspecto muito importante, pois feito de forma errônea pode ter efeitos catastróficos [17]. Dentro de SMA há propriedades importantes como: a distribuição de atividades, a descentralização dos dados e a comunicação assíncrona. Essas propriedades necessitam de algum tipo de coordenação dos componentes para que todos os conflitos sejam solucionados na execução de cada tarefa dos agente e no alcance dos objetivos de cada agente ou para um grupo de agentes [18]. Atribui-se a maior complexidade desses sistemas à coordenação dos agentes, devido a necessidade de formalizar como se darão as tomadas de decisão, o controle e a comunicação dos agentes.

SMA possui, naturalmente um controle global explícito. Dessa forma, o desenvolvedor precisa escolher a estratégia de coordenação que melhor se encaixe na solução de seu problema. A ideia de uma estrutura de agentes em hierarquia é amplamente utilizada, de forma que existem gerentes que coordenam agentes [18]. No entanto, tem-se o problema da centralização por vezes indesejável em sistemas distribuídos.

O planejamento de um grupo de agentes pode também ser deliberado por um gerente ou um grupo. Quando dessa forma, os gerentes definem os planos e os subordinados, assim diminuindo as ocorrências de conflitos e melhorando o uso de recursos compartilhados [18]. É importante definir o protocolo de coordenação dos SMA. No entanto, é um problema não trivial a definição dessa coordenação, pois deve-se balancear a liberdade do agente, sem interferir na sua autonomia deliberativa, mas permitir que os objetivos do grupo de agentes sejam alcançados no processo de resolução do problema como um todo [9].

2.1.6 Protocolo de Interação

Algumas das principais características dos SMA são: a distribuição do controle, a descentralização dos dados e a comunicação assíncrona [19]. No entanto, a utilização de um SMA requer o uso de mecanismos de controle de atuação coletiva, uma vez que os agentes são autônomos, podendo trabalhar de forma cooperativa ou competitiva, em busca de objetivos coletivos ou individuais. Esses protocolos de interação e comunicação são essenciais na tecnologia de SMA, existindo várias abordagens para definir os mesmos [9].

Existem ferramentas de desenvolvimento de SMA que possuem os protocolos de interação e comunicação já implementados facilitando a utilização dos mesmo agentes implementados, como o *Java Agent Development Framework* (JADE) [2] [11] [18]. Conforme o desenvolvimento aumenta, cresce a necessidade de novas instâncias de agentes de forma a não centralizar muitas funções em um único agente. O controle desses agentes se torna mais complexo, tendo em vista as características de distribuição de processamento e as diversas tecnologias que podem ser utilizadas na solução. A troca de mensagens entre os agentes também pode acarretar um gasto de processamento e memória, com a finalidade de encontrar um equilíbrio ou até formas alternativas de comunicação.

[2]Java Agent DEvelopment Framework - JADE – http://www.jade.tilab.org/

Ao se desenvolver um projeto de SMA, que possua características de raciocínio, distribuição deve utilizar mecanismos de coordenação, seja por cooperação, seja por competição. Muitas estratégias de procolos de interação têm sido apresentadas como maneira de obter melhor desempenho em SMA, diminuindo ao máximo a necessidade de intervenção humana. O uso de SMA como ferramenta em um modelo para nuvem computacional pode apresentar diversas vantagens, por exemplo utilizando somente o necessário de recursos tanto para gerenciar quanto para o uso da aplicação gerenciada, podendo ser inserido em um ambiente de nuvem real.

[9] apresenta as seguintes características que um agente pode apresentar:

1. Sociabilidade: a capacidade do agente de interagir e se comunicar;
2. Mobilidade: poder transitar entre ambientes, quando necessário;
3. Veracidade: garantia de uma comunicação com informações verdadeiras;
4. Benevolência: capacidade de ajudar outros agentes a cumprirem seus objetivos, desde que isso não conflite com os objetivos individuais;
5. Racionalidade: capacidade de agir sempre em busca dos próprios objetivos;
6. Aprendizado e Adaptação: capacidade de se adaptar e atualizar os objetivos de acordo com as mudanças no ambiente.

Assim, os agentes podem interagir e trabalhar em torno de um objetivo comum.

2.1.7 Ferramentas para Desenvolvimento de SMA

O projeto de SMA pode fazer uso de ferramentas de desenvolvimento, uma vez que existem meios para auxiliar o projetista. Assim, esta seção apresenta diferentes ferramentas para desenvolvimento de SMA, expondo suas principais características.

O JADE é um *framework* implementado na linguagem Java, que segue as especificações da FIPA. Permite interoperabilidade entre diferentes plataformas de softwares ou físicos, como exemplo em plataformas *web*, *desktop* e *mobile* [6]. O JADE permite implementar uma arquitetura na qual os agentes conseguem realizar trocas de mensagens, porém não têm uma implementação explícita de racionalidade, não sendo possível, por exemplo, criar agente com arquitetura baseada em *belief–desire–intention software* – BDI. Assim, fica a cargo do projetista realizar a implementação explícita do modelo de racionalidade.

JADE *eXtension* – JADEX é um motor de raciocínio que segue o modelo *belief-desire-intention software* – BDI [20]. O BDI é uma abordagem de racionalidade de agentes inteligentes, apresenta uma teoria filosófica do raciocínio prático, tentando representar o raciocínio humano dos seguintes pontos: crenças, desejos e intenções [21]. É uma extensão do JADE, que possibilita a comunicação entre agentes utilizando troca de mensagens. Implementado na linguagem JAVA e com XML permite o projetista declarar um conjunto de descrição que representa suas crenças, objetivos e planos. Com essas descrições é definida a racionalidade do agente.

Jason[3] é um interpretador para a linguagem de programação orientada a agentes AgentSpeak. Implementa a semântica operacional da linguagem, e fornece uma plata-

[3]http://jason.sourceforge.net/

forma para o desenvolvimento de SMA, dando liberdade para o usuário criar características personalizadas. Jason tem seu código aberto e é distribuído sob a licença GNU LGPL. A troca de mensagem é feita com o padrão FIPA de interação, dessa forma é possível a comunicação entre os agentes JADE e JADEX [22].

ZEUS[4] é um ambiente que possibilita a construção de agentes. Já está incluso ao ZEUS um mecanismo de regras, ferramentas para planejamento e visualização. Um ponto forte é a extensão para projetos de web semântica *DARPA Agent Markup Language* (DAML) e recursos de integração de *Web Services*. Desenvolvido pela British Telecomunnications e foi implementado na linguagem JAVA, a mesma linguagem na qual os seus agentes são criados [23].

Shell for Simulated Agent Systems – SeSam[5], utilizando a linguagem JAVA permite construir agentes de forma simplificada, mas também dá suporte ao desenvolvimento de sistemas complexos, em que podem ser incluídas dependências entre os elementos de forma dinâmica. Desenvolvido pela Universidade de Würzburgo, apresenta um ambiente genérico para experimentação com simulação baseada em agentes. O SeSam traz um ambiente de modelagem visual, um ambiente flexível de desenvolvimento na linguagem JAVA, um módulo para definir situações na simulação e um sistema de análise integrado. Os agentes projetados no SeSam podem se comunicar com agentes desenvolvidos nas plataformas JADE e JADEX por meio de troca de mensagens utilizando o padrão FIPA-ACL.

A Tabela 2.4 apresenta um resumo das ferramentas descritas nessa seção.

Tabela 2.4: Resumo das ferramentas de desenvolvimentos de SMA.

Framework	**Especificações FIPA**	**Principais Características**	**Modelos de Racionalidade**
JADE	Sim	SMA distribuído, modelagem simples	Lógica booleana, lógica fuzzy, redes neurais, classificadores bayesianos e árvores de decisão
Jadex	Sim	Modelagem BDI, extensão do JADE	Primariamente BDI
Jason	Sim	Interpreta a linguagem orientada a agentes AgentSpeak	Primariamente BDI
ZEUS	Sim	Modelagem de aplicações orientadas a tarefas	Nenhum modelo explicitamente implementado
SeSam	Sim	Ambiente de modelagem visual e análise integrado	Nenhum modelo explicitamente implementado

[4]https://sourceforge.net/projects/zeusagent/

[5]http://www.cs.bham.ac.uk/research/projects/poplog/packages/simagent.html

2.2 Agente com Dedução Lógica

Como apresentado na Seção 2.1,existem várias funções para implementar o raciocínio dos agentes que mapeiam as percepções em ações. Um agente reflexivo simples, por exemplo, não possui conhecimento para agir, suas ações são reflexos diretos de percepções de estados do ambiente. No entanto, problemas mais complexos carecem de um nível de conhecimento para sua solução, neste sentido um modelo de tratamento de conhecimento se faz oportuno. Em ambientes parcialmente observáveis e dinâmicos, a capacidade de conhecimento é crucial para o agente. Quando o agente necessita que suas tarefas sejam atualizadas de acordo com as mudanças do ambientes também é primordial a capacidade de conhecimento [9]. Agente baseado em conhecimento tem a representação do mundo em uma base de conhecimento (BC), que possui um conjunto de regras.

Utilizando uma BC um agente pode armazenar um fato, que represente a realidade do mundo através de uma diretiva TELL, pode questionar a BC através de uma diretiva ASK ou pode remover um fato por meio de uma diretiva RETRACT. Quando o agente escreve a operação atualiza a BC e para que o agente leia, é solicitado uma sentença da BC, que atenda ao atual ambiente. Existem operações básicas em agente baseado em conhecimento como: (1) quando a BC é atualizada o agente percebe essa atualização de conhecimento; (2) quando o agente executa uma ação, ele informa para a BC que foi realizado uma ação. Com um agente baseado em conhecimento é possível tratar novos fatos que ocorrem no ambiente para subsidiar a tomada de decisão dos agentes.

A Figura 2.7 apresenta a arquitetura de um agente baseado em conhecimento com estas capacidades e características.

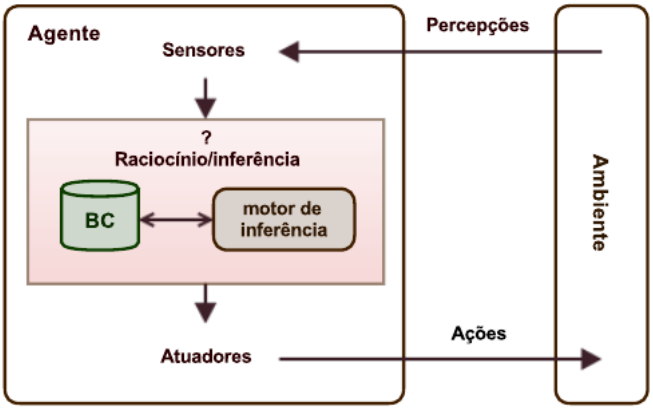

Figura 2.7: Arquitetura de agente baseado em conhecimento [9].

Na arquitetura apresentada na Figura 2.7 dentro do raciocínio/inferência é utilizado um motor de inferência junto a BC para realizar associações, derivar novas representações do mundo e escolher as melhores ações que o agente pode realizar. Ainda estão inclusas as básicas funcionalidades de um motor de inferência, descrito na Seção 2.2.3. Alguns modelos de representação de conhecimento exigem um mecanismo de raciocínio específico associado, como regras declarativas, ontologias, árvores de decisão, entre outros.

Agentes baseados em conhecimento não possuem um mecanismo arbitrário para calcular ações. De acordo com as definições TELL, ASK e RETRACT, agentes com essas características possuem a habilidade de se adaptarem a mudanças do ambiente, podendo

alcançar objetivos conforme as regras definidas na BC. É possível tornar o agente ainda mais autônomo, uma vez que o mesmo pode aprender, o que lhe confere a capacidade de identificar características ocultas do ambiente.

Russel & Norvig [9] definem os procedimentos de um agente baseados em conhecimento genérico como mostrado no Algoritmo 2.6 (código adaptado), em que é considerada a BC para as ações do agente.

Algoritmos 2.6: Função de mapeamento ação-percepção de um agente orientado a objetivos.

```
1  PERCEIVE ( entrada ): percebe uma entrada
2  TELL ( BC, entrada ): informa à BC o que é percebido ( entrada )
3  ação ← ASK ( BC, inferência): consulta à BC para deduzir / inferir uma ação
4  TELL ( BC, ação): informa à BC a ação tomada
5  RETORNA ação: realiza ação
```

2.2.1 Representação do Conhecimento

A exteriorização inteligente prediz aquisição, armazenamento e inferência do conhecimento. O armazenamento do conhecimento só é possível ser feito, quando este conhecimento possa ser representado. De acordo com [24], qualquer máquina que tenha um comportamento inteligente necessita conter estruturas que permitam descrever proporcionalmente o conhecimento exibido.

A representação do conhecimento possui características desejáveis, quando se deseja implementá-la. Essas características podem ser utilizadas como apoio na validação da representação. As características são: definir claramente e explicitamente os objetos e suas relações; deixar explícito todas as restrições do ambiente e dos objetos envolvidos; e explicitar os detalhes relevantes e suprimir os irrelevantes.

Além das características citadas é importante que a representação possua: transparência, o que permite a compreensão do que está ocorrendo e sendo expresso; rapidez, para possibilitar a escrita e a leitura na BC em um tempo aceitável; e que seja computável, o que possibilita sua criação utilizando um procedimento computacional.

Para criar a representação do conhecimento, um engenheiro do conhecimento necessita de um especialista do ambiente, a fim de elaborar sentenças e regras que podem representar o conhecimento. Com as regras e sentenças modeladas é necessário que sejam validadas em entrevistas, obtendo uma BC concisa e confiável.

Com a BC modelada e bem definida, o agente poderá ser inserido no ambiente para qual foi implementado. E utilizando o comando TELL informa a BC o que percebe e conhece, e com o ASK questiona as ações que serão realizadas para aquele estado, assim como os procedimentos descritos no Algorítimo 2.6.

2.2.2 Raciocínio Lógico Baseado em Regras

Agentes com raciocínio lógico baseado em regras de inferência possuem uma BC formada por regras fixas ou incrementais, além de um conjunto de fatos que variam de acordo com ocorrências no ambiente. Uma BC pode ser formada por um conjunto grande de regras,

necessitando portanto um motor de inferência com bom desempenho para processá-las para não impactar a tempo de resposta do sistema.

Para representação do conhecimento faz-se necessário uma linguagem específica, que possui uma sintaxe adequada para transmitir a informação de maneira que as estruturas se organizem e se relacionem; e uma semântica que dá significado as estruturas de acordo com o contexto. A linguagem pode ser denotada por formalismos lógicos, como a lógica proposicional, lógica de primeira ordem, lógica descritiva, entre outros.

As cláusulas de Horn apresentam uma sintaxe, que retrata a ideia de consequência lógica entre sentenças na forma clausal, como apresentado em [25] [9]. A sintaxe mostra basicamente na forma "se A, então B", onde B é a consequência de A. Nesta expressão denota-se que A é a condição necessária para a consequência, e B são as ações que serão executadas de acordo com as regras que forem ativadas.

Regras de inferência são transformações sintáticas, que podem usar sentenças lógicas para descobrir provas [9]. A forma normal conjuntiva (FNC) é introduzida como um mecanismo de inferência pelo princípio da resolução. Através deste princípio, para dar como verdade a sentença $\beta \leftarrow \alpha$, deve-se concluir por contradição a equivalência lógica $\neg\beta \wedge \alpha$ na FNC seja não-satisfatível.

A inferência com cláusulas de Horn é o mecanismo utilizado por motores de inferência baseados em regras, explicados na Seção 2.2.3, seguindo o princípio da resolução, o qual pode ser feito por meio dos seguintes algoritmos [26] [9]:

- **encadeamento progressivo**: raciocínio dirigido a dados. Os fatos da base sobre os quais é considerada a parte antecedente (condição, premissa) das regras, e sucedem-se até que um objetivo ou ação seja alcançado;
- **encadeamento regressivo**: raciocínio dirigido a objetivos e subobjetivos.

É importante entender os encadeamentos lógicos (progressivo e regressivo) como estratégias de busca de problemas. Dessa forma, estados intermediários correspondem às hipóteses intermediárias pelo encaminhamento regressivo, ou conclusões intermediárias por meio do encadeamento progressivo. No encadeamento regressivo, a explanação das ações é feita com maior facilidade, tendo em vista que os objetivos são hipóteses conhecidas. Este comportamento exibe um raciocínio dirigido a objetivos. Por sua vez, no encadeamento progressivo, as ações possuem uma maior complexidade, uma vez que através dos fatos percebidos sobre o ambiente e, a partir destes, criam-se novos fatos, de maneira que se tentam encontrar os objetivos e deduzir suas ações. Com este comportamento tem-se um raciocínio dirigido a dados.

Na Tabela 2.5 são apresentadas as principais características de cada encadeamento.

2.2.3 Motores de Inferência Baseados em Regras

Motor de inferência pode ser definido como um programa, que utiliza dados como BC para encontrar soluções para determinados problemas, isso é possível com a combinação de fatos e regras. A BC é formada por fatos e por regras, exemplo: SE condição ENTÃO conclusão (e.g., $A, B \rightarrow C$).

Um motor de inferência utiliza mecanismos gerais de combinação de fatos e regras, seu funcionamento é feito de forma cíclica. Um ciclo pode ser descrito por duas etapas: (1)

Tabela 2.5: Características dos encadeamentos progressivo e regressivo.

Encadeamento Progressivo	**Encadeamento Regressivo**
Raciocínio para frente	Raciocínio para trás
Raciocínio orientado pelos dados	Raciocínio orientado pelos objetivos
Parte-se de sentenças (regras de inferência)	Parte-se de uma hipótese que se quer provar
Problemas típicos: planejamento, projeto e classificação	Problemas típicos: diagnóstico

avaliação, que procura as regras possíveis de serem acionadas, em decorrência do estado corrente da BC em escolha das regras ativas efetivamente; e (2) execução, modifica a BC.

Os motores de inferência são compostos pelos seguintes componentes [27]:

1. **casamento de padrões**: realiza o casamento entre as regras e os fatos, quando feita a união são ativadas as execuções, seguindo uma determinada ordem para solucionar possíveis conflitos;
2. **agenda**: controle da ordem de execução, garante que após as regras ativadas, sejam executadas as tarefas na ordem e na prioridade correta; e
3. **motor de inferência**: responsável por executar as regras ativas e ordenadas na agenda. A execução pode ser por encadeamento progressivo ou regressivo, permitindo inclusive que sejam combinados os diferentes mecanismos de inferência.

Na Figura 2.8 são apresentados modelos de arquiteturas básicas de raciocínio baseado em regras de produção, em que (a) foi estabelecido por [9] e (b) por [28].

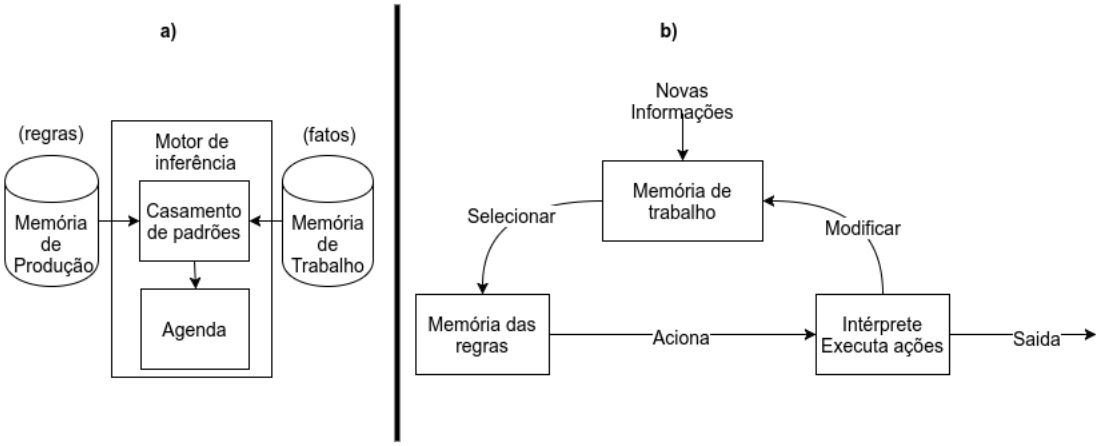

Figura 2.8: Arquitetura básica de regras de produção.

Ferramentas

Na literatura de IA são citadas várias ferramentas para tratar regras de produção. Algumas dessas ferramentas foram estudadas nesse trabalho e serão brevemente apresentadas.

Drools [29] é uma solução de *Business Rules Management System* (BRMS), a plataforma de integração de lógica de negócios fornece uma ferramenta unificada e integrada para Regras, *Workflow* e processamento de eventos. Desenvolvido na linguagem Java, é um motor de inferência baseado em regras, seu mecanismo de inferência ocorre por encadeamento progressivo e regressivo. Implementado em seu motor de inferência, os algoritmos Rete [30] e Leaps [31]. O Algoritmo Rete é utilizado para determinar quais regras da BC serão executadas conforme a memória de trabalho (fatos). O algoritmos Leaps realiza o processamento das regras. Com esses algoritmos é possível realizar a junção de padrões entre fatos (novos e/ou existentes) e regras. O Drools permite sua implantação em diferentes tipos de projetos, uma vez que combina a inferência de regras com ocorrências de eventos complexos, além de processos de auto-planejamento e *workflows*.

JESS (*Java Expert System Shell*) [32], desenvolvido em linguagem Java pela Sandia National Laboratories nos anos 90. Seu desenvolvimento foi baseado na linguagem de regas *C Language Integrated Production System* (CLIPS). Implementa os encadeamentos progressivo e regressivo. Utiliza o algoritmo Rete para realizar casamento de padrões entre regras e fatos.

Apache Jena [6] é um *framework* Java para construir aplicações de Web Semântica. Desenvolvido pela HP Labs em 2000 na linguagem Java, se tornou parte da Fundação APACHE em 2010. Fornece uma extensa biblioteca Java para ajudar os desenvolvedores nas implementações. O motor de inferência é baseado em regras para executar raciocínio baseado em *Ontology Web Language* (OWL)[7] que descreve um idioma para ontologias e ontologias, *Resource Description Framework* (RDF) e *RDF Schema* (RDFS)[8] uma linguagem para escrever ontologias, e diferentes estratégias de armazenamento de RDF. Pode-ser usar o RDF para expressar fatos com ator, predicado e objeto. Ontologias e regras podem ser acopladas em uma BC, o que permite um mecanismo hibrido de inferência entre regras e ontologias. O mecanismo de raciocínio baseado em regras inclui raciocínios por encadeamentos progressivo e regressivo.

JEOPS (*Java Embedded Object Production System*) [33] é um mecanismo de regras progressivas baseadas em Java. Este motor de regras é usado para ligar processos de negócio através de regras em Servidores de aplicação Java, aplicativos cliente, e *Servlets*. Permite o desenvolvimento de aplicações inteligentes, por exemplo SMA e/ou sistemas especialistas. JEOPS utiliza o algoritmo Rete para realizar casamento de padrões. O mecanismo de raciocínio é feito por encadeamento progressivo.

PROLOG [34][35], desenvolvido em 1970, é um mecanismo de resolução sobre predicados especiais, cláusulas de Horn, chamado unificação. Uma linguagem puramente lógica, baseada no conceito de resolução linear proposto por Kowalski [36]. O mecanismo de inferência é feito por encadeamento regressivo.

2.3 Computação em Nuvem

Pesquisadores divergem na definição de computação em nuvem. No entanto, percebe-se que as definições disponíveis na literatura estão sempre relacionadas a termos como

[6] Apache Jena - https://jena.apache.org/

[7] https://www.w3.org/OWL/

[8] https://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

computação orientada a serviço, virtualização, provimento de recursos por demanda e a elasticidade.

Neste sentido, Buyya et al. [37] definem Computação em Nuvem como um tipo de sistema paralelo e distribuído, que consiste em uma coleção de computadores interconectados, que são provisionados dinamicamente e apresentados como um ou mais recursos computacionais unificados, baseados em acordos de nível de serviço estabelecidos entre provedor de recursos e o consumidor.

De acordo com Armbrust et al. [38], a Computação em Nuvem pode ser definida como aplicações sendo oferecidas como serviço através da Internet, com o hardware e software localizados em *datacenters* em que é disponibilizado o serviço. Essa definição pode conduzir à dupla interpretação. É possível, por exemplo confundi-la com a definição de outros paradigmas de computação distribuída, como *grid* computacional [39].

Segundo Foster et al. [3], Computação em Nuvem é um modelo em que se pode utilizar e compartilhar recursos (redes, arquitetura, aplicações e serviços) de forma independente. Em outras palavras, é um modelo no qual o usuário consegue configurar, contratar e alterar seus recursos com o mínimo de interação humana. Pode-se ainda dizer que a Computação em Nuvem é dividida em cinco características essenciais, três modelos de serviço e quatro modelos de implantação, apresentado nas próximas seções [4].

Pela própria literatura de nuvem, um provedor fornece serviços aos clientes e, dessa forma, estes precisam ter especificado o quanto e como deve ocorrer a qualidade destes serviços. A definição de Qualidade de Serviço (*Quality of Service* - QoS) é a capacidade de atender a certos requisitos para diferentes serviços, como desempenho, disponibilidade, confiabilidade ou custo. Provedores e clientes têm a negociar um Acordo de Nível de Serviço (*Service Level Agreement* - SLA), que lhes permita especificar formalmente a QoS e concordar com os requisitos [40].

2.3.1 Modelos de Implantação

A implantação de uma nuvem computacional pode ser diferente para cada necessidade. A escolha por restrição ou abertura de acesso depende diretamente do modelo de negócio adotado por quem irá implantar. Determinadas empresas podem precisar que seus serviços de nuvem tenham maior segurança e restrição, a certas classes e usuários. Nasce então a necessidade de ambientes com maior restrição [4].

Mell e Grance [4] apresentam quatro modelos de implantação de computação em nuvem: nuvem privada, nuvem comunitária, nuvem pública e nuvem híbrida.

Nuvem Privada

No modelo de implantação de uma nuvem privada, a infraestrutura da nuvem é provida para uso exclusivo de uma única organização, sendo administrada pela própria empresa ou por terceiros. São empregadas políticas de utilização próprias da empresa para o acesso aos serviços.

Nuvem Pública

No modelo de implantação de uma nuvem pública, a infraestrutura é provisionada para uso aberto pelo público em geral, sendo disponibilizado para qualquer usuário que conheça a localização dos serviços.

Nuvem Comunitária

No modelo de implantação de uma nuvem comunitária, a infraestrutura é provisionada para uso exclusivo por uma comunidade específica de consumidores. Esse modelo de nuvem pode ser compartilhado por diversas empresas que compartilham interesses. Sua existência pode ser local ou remota, sendo administrada por empresas da comunidade ou por terceiros.

Nuvem Híbrida

No modelo de implantação de uma nuvem híbrida, a infraestrutura é uma composição de duas ou mais infraestruturas distintas de nuvem, o que permite ser caracterizada como uma nuvem privada, comunitária ou pública, implementada como uma única nuvem.

2.3.2 Arquitetura de Nuvem

Existem na literatura diferentes propostas de arquitetura de nuvem [41] [42]. Uma delas é a arquitetura de nuvem proposta por [39], apresentada na Figura 2.9, que identifica três grupos de serviço: *software*, plataforma e infraestrutura.

Os primeiros atores são os provedores de infraestrutura responsáveis por oferecer recursos físicos aos provedores de serviços. Assim, os provedores de serviços não têm de se preocupar com a estrutura física. Já os provedores de serviços são responsáveis em oferecer seus serviços aos usuários finais, sendo essa entrega feita de forma flexível e escalar. Por fim, os usuários de serviço têm acesso as aplicações fornecidas através da Internet e a cobrança é feita de acordo com o consumo.

A arquitetura apresentada na Figura 2.9 é dividida em três camadas. De baixo para cima, existe primeiramente a camada de infraestrutura, que oferece aos usuários recursos como poder computacional, memória RAM, disco, rede entres outros, encapsulados como SVM. Na camada de infraestrutura (*Infrastructure as a Service* - IaaS) é feita a instanciação de SVM. Na camada de Plataforma como serviço (*Platform as a Service* - PaaS) estão as plataformas de desenvolvimento, nestas são encontradas ferramentas que auxiliam desenvolvedores no ambiente de nuvem. Por fim, tem-se a camada mais próxima do usuário, nela estão as aplicações que permitem o usuário final acessar os serviços (*Software as a Service* - SaaS).

2.3.3 Modelos de Serviços

Os modelos de serviços são muito importantes quando se caracteriza nuvem computacional, pois é através deles que podem ser explanados os tipos de serviços e como o provedor os estará prestando ao consumidor. De acordo com [4], os modelos de serviços estão classificados como: SaaS, PaaS e IaaS.

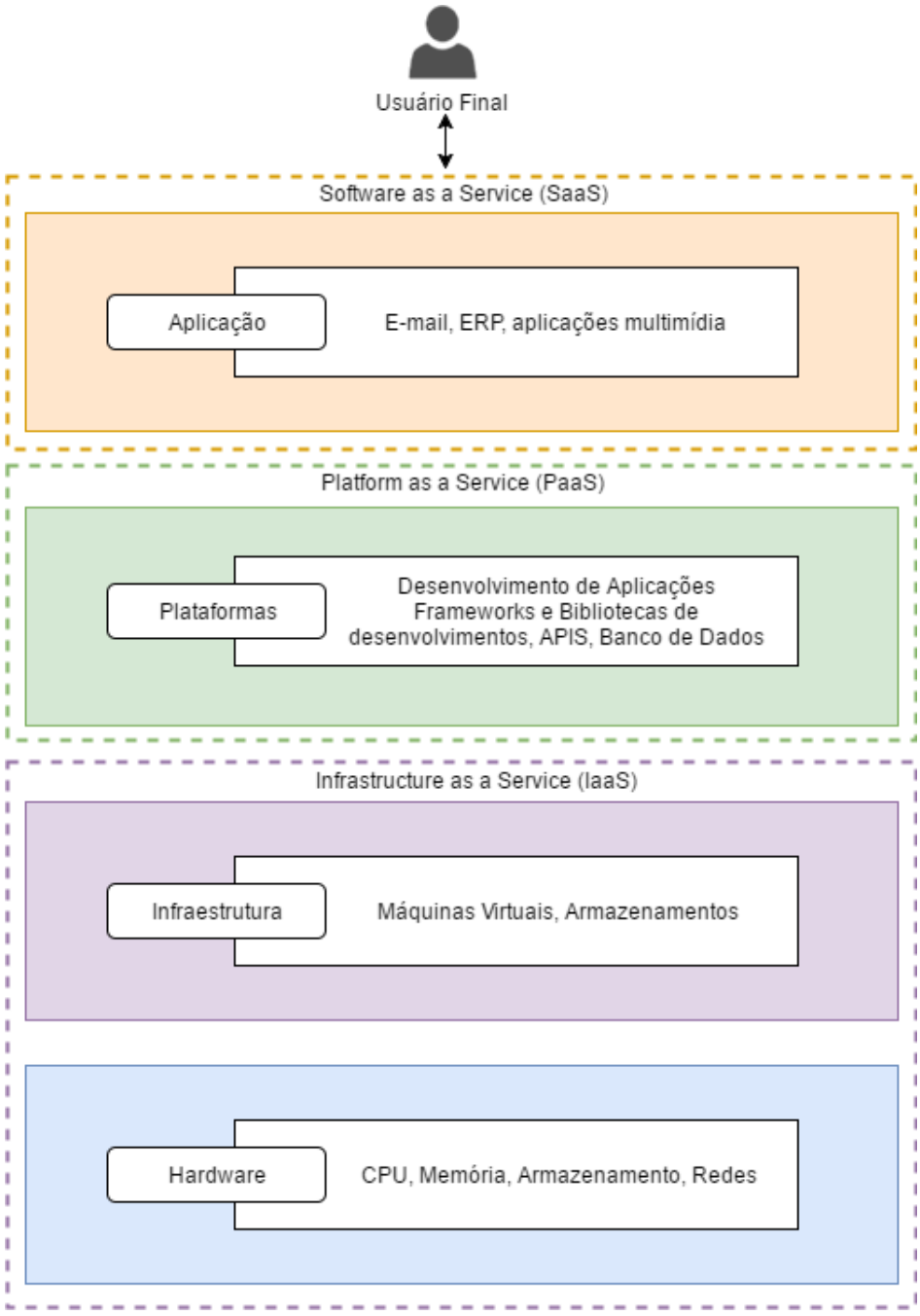

Figura 2.9: Arquitetura de computação em nuvem [39].

- SaaS - os serviços são prestados em nível de aplicação, de forma que o provedor já tenha implementado suas aplicações e forneça apenas aplicações finais para os usuários. Nesse caso, as aplicações podem ser acessadas por um navegador *web*. O usuário não tem acesso à implementação, banco de dados, rede, sistemas operacionais. Seu acesso é limitado a execução da aplicação e não nos aspectos de desenvolvimento. Por ser um *software* fornecido pela *web*, o usuário pode acessar a qualquer momento e em qualquer lugar. O provedor pode realizar atualizações e disponibilizá-las sem que estas tenham de ser reinstaladas para cada usuário final tornando-se mais acessível, pois o usuário fica livre de licenças e pode pagar apenas pelo que usa.
- PaaS - o consumidor recebe serviços para implantar, integrar e implementar aplicações na nuvem. Apesar de ainda não possuir controle sobre a infraestrutura, o usuário tem controle sobre ferramentas e bibliotecas para implementar aplicações em ambiente de nuvem. Dentro do modelo PaaS é fornecido um sistema operacional, linguagem de programação e ambientes para desenvolvimento, geralmente, sendo permitida ainda a colaboração entre diferentes desenvolvedores.
- IaaS - o principal objetivo é dar acesso ao usuário direto aos recursos (virtuais), tais como: redes, armazenamento e processamento. O consumidor não gerencia ou controla a parte física da nuvem, porém tem controle da infraestrutura das máquinas virtuais, dos sistemas operacionais, armazenamento e aplicativos implementados.

2.3.4 Características Essenciais

De acordo com as definições de [4] e [39], as principais caraterísticas que compõem a computação em nuvem são: serviços por demanda, suporte a diversas plataformas, medição de serviço, agrupamento de recursos e elasticidade rápida.

Serviços por demanda

O consumidor pode adquirir recursos, como quantidade de memória e armazenamento, através de catálogos fornecidos pelo provedor, sem uma interação humana direta. Dentro da nuvem, os *hardwares* e *softwares* podem ser reconfigurados pelo provedor de modo transparente para o usuário, aumentando o poder computacional da nuvem. Dessa forma, o consumidor pode aumentar e diminuir o número e a configuração dos recursos alocados de acordo com sua necessidade, sendo possível que o consumidor tenha um ambiente computacional personalizado para cada perfil.

Suporte a diversas Plataformas

Nessa característica da nuvem os recursos podem e devem estar disponíveis independentemente da plataforma usado para acesso, seja ela móvel ou fixa (computadores, *notebooks*, celulares e/ou *tablets*). Os serviços são oferecidos via *web*, dessa forma deve existir a possibilidade do usuário acessar os serviços diretamente de um navegador ou aplicativo que tenha acesso à Internet.

Medição de Serviços

Essa característica proporciona o monitoramento, controle e relatório, proporcionando transparência para o consumidor. A medição permite ao provedor controlar e otimizar o uso de recursos. Essa característica usa monitoramento para garantir o QoS e SLA.

Agrupamento de Recursos

Os provedores organizam seus recursos em um *pool* para servir múltiplos usuários usando o modelo multi-inquilino (*multi-tenant*). Os recursos são atribuídos aos usuários dinamicamente e tarifados de acordo com o uso dos mesmos. Os usuários não precisam saber onde os recursos estão localizado geograficamente, pois o acesso aos recursos é feito em um alto nível de abstração, através de *middlewares*.

Elasticidade Rápida

Essa característica permite que a quantidade de recursos seja expandida ou reduzida de forma elástica para o consumidor. Para o consumidor os recursos parecem ilimitados, pois o aumento ou diminuição destes são feitos de forma transparente. A virtualização é um ponto forte para a característica de elasticidade, pois cria sobre a infraestrutura física diversas instâncias com variação de recursos.

2.3.5 Monitoramento

A fim de assegurar que os recursos de *software* e *hardware* implantados cumpram com o SLA, um processo de monitoramento contínuo de recursos é indispensável. O monitoramento busca realizar a detecção e a coleta de informações sobre cada um dos recursos envolvidos na execução das tarefas do usuário, ou seja, o monitoramento é de suma importância para ambas as partes, tanto para o provedor quanto para o cliente [43].

Na computação em nuvem, o monitoramento pode ser de diferentes níveis. Monitoramento de alto nível está relacionado com o status de plataforma virtual, também pode se ter monitoramento sobre as aplicações. O monitoramento de nível médio está relacionado aos PaaS, assim monitorando as plataformas de desenvolvimento, por exemplo monitoramento da JVM (*Java Virtual Machine*). Por fim, o monitoramento de baixo nível está relacionado com as informações recolhidas sobre o estado da infraestrutura, como uso de CPU, memória, entre outros, das máquinas virtuais [44].

Segundo [45] existem diferentes ferramentas para monitoramento de recursos com estratégias diversas:

- IoStat[9] é um comando Linux usado para monitorar dispositivos de entrada/saída e CPU. Gera relatórios com detalhes de taxas médias de transferências;
- MpStat[10] é um comando Linux que monitora e relata as estatísticas relacionadas à CPU. Ele é usado no monitoramento de computadores para diagnosticar problemas ou para construir estatísticas sobre o uso dos recursos de CPU;

[9] http://linuxcommand.org/man$_p ages/iostat1.html$

[10] http://www.linuxcommand.org/man$_p ages/mpstat1.html$

- Ganglia[11] Ganglia é um sistema de monitoramento distribuído escalável para sistemas de computação de alto desempenho, como clusters e Grids. Ganglia é um projeto open-source licenciado pela BSD;
- Dstat[12] é um substituto para MpStat e IoStat no Linux. Supera limitações e adiciona recursos extras para as outras ferramentas. Dstat permite visualizar todos os recursos do sistema em tempo real. Fornece informações seletivas detalhadas em colunas e indica claramente em que magnitude e unidade a saída é exibida.

2.3.6 Provisionamento Dinâmico de Recursos

O provisionamento dinâmico de recursos em nuvem é investigado, principalmente, por provedores, com o objetivo de identificar o perfil de uso dos recursos pelo usuário. A computação em nuvem trouxe grandes vantagens como elasticidade e tarifação por uso de recursos para o usuário, deixando para os provedores o custo de manter a infraestrutura computacional sempre funcional [38]. Dessa forma, para os provedores é importante garantir seus serviços de forma que o gerenciamento dos recursos seja feito de forma otimizada. Uma das formas de garantir o melhor uso dos recursos é com o provisionamento dinâmico, que busca por meio de técnicas de predição de recursos identificar as necessidades do usuário.

Provisionamento pode ser classificado em três categorias: adequado, sub-provisionado e super-provisionado. Quando os recursos provisionados estão adequados, o usuário está utilizando o que foi alocado de maneira adequada. Quando sub-provisionado o desempenho está inferior ao esperado, ou seja, os recursos alocados não são suficientes para a tarefa que está sendo executada. No caso de um super-provisionamento são alocados recursos além dos necessários para realizar uma tarefa, exitem recursos sendo pagos e não sendo utilizados [46].

Em provedores de IaaS é possível provisionar recursos variados, como rede (largura de banda), *virtualCPU* (vCPUs), *Hard Disk* (HD). As tomadas de decisões sobre o balanceamento entre custo e uso de recursos precisam ser feitas com muito critério, de forma que se busque encontrar um limiar que atenda as necessidades com custo aceitável [47].

2.3.7 Predição de Uso de Recursos

Para realizar o provisionamento dinâmico é fundamental predizer as necessidades do usuário ou do serviço que será executado. O serviço de predição deve ser capaz de prever as elevações de demanda das aplicações, para que a decisão certa seja tomada. Para o provedor uma das vantagens é a garantir a QoS e para o usuário a diminuição de custo e/ou de tempo, e o aumento da qualidade na execução da tarefa. A busca das necessidades futuras de recursos computacionais, armazenamento, entre outros, deve levar em consideração os custos, tempo e recursos que trazem uma previsão efetiva de um ambiente controlável, o que eleva a confiança no sistema [48].

Criar um serviço de predição adequado requer a utilização de modelos que podem prever as necessidades de recursos para uma aplicação. Esse modelo deve ainda se ajustar

[11] http://www.ganglia.sourceforge.net
[12] http://dag.wiee.rs/home-made/dstat/

com sua utilização adaptando-se as mudanças. Um modelo de predição pode ser implementado utilizando diversas abordagens, tais como aprendizado de máquina [49] [50] [51] e estatística [49] [52].

O aprendizado de máquina é um campo de pesquisa da Inteligência Computacional, que estuda o desenvolvimento de métodos capazes de extrair informações a partir de amostras de dados [53]. Em geral, os algorismos de aprendizado de máquina são utilizados para classificar um conjunto de dados em determinados conjuntos. As técnicas de aprendizado de máquina são empregadas para classificação, sendo capaz de classificar em uma classe uma determinada instância.

Segundo [53] a área de aprendizado de máquina possui três paradigmas que podem ser utilizados nos preditores: supervisionado, não-supervisionado e por reforço. A escolha do paradigma determina como a técnica de predição irá se relacionar ao conjunto de dados.

No paradigma supervisionado, o algoritmo de aprendizado é treinado a partir de conjuntos de exemplos já conhecidos, com o objetivo de aprender uma função desejada. No paradigma não-supervisionado, o algoritmo aprende a agrupar as entradas submetidas segundo uma medida de qualidade, uma vez que os conjuntos existentes não são conhecidos. No paradigma por reforço, o aprendizado se comporta por meio de recompensas, garantindo desempenho próximo da função desejada.

Segundo [9] a área de aprendizado de máquina pode ser dividida em aprendizado por exemplos, aprendizado baseado em conhecimento, aprendizado a partir de modelos probabilísticos e aprendizado por reforço. Considerando a área de aprendizado por exemplos existem métodos como regressão e classificação com modelos lineares, redes neurais e máquinas de vetor de suporte (*Support Vector Machines* - SVM). Na sequência serão apresentados esses métodos como técnicas de predição.

Técnicas de Predição

O uso da predição pode auxiliar diversas tomadas de decisão, tais como balanceamento de carga, ajuste de recursos, custos e tempo de execução. A escolha da técnica de predição deve ser feita de forma minuciosa, uma vez que uma determinada técnica pode atender melhor a determinadas necessidades.

Máquinas de Vetor de Suporte

As SVM podem ser definidas como uma técnica de aprendizado supervisionado, que reconhece padrões a partir de amostras de dados, usados para classificação e análise de regressão. Algumas características das SVM são: boa capacidade de generalização, robustez em grandes dimensões, convexidade da função objetivo e teoria bem definida [54]. Para conceituar SVM tem-se a entrada de um conjunto de dados que são mapeados de forma não linear em características de alta dimensionalidade, por meio de um mapeamento estabelecido inicialmente. Normalmente, através deste espaço de características, é construído um conjunto de decisão linear, que expressa as propriedades responsáveis por garantir a habilidade de generalização da máquina de aprendizagem. A generalização é uma das principais características das SVM. Por meio destas podem-se atingir eficiência na classificação de dados que não pertençam ao conjunto de dado utilizados no treinamento [55].

A Figura 2.10 mostra um esquema que busca separar, ao máximo, os dados (círculos verdes e azuis) de um conjunto, utilizando uma técnica básica de SVM com conjunto de decisão linear. Onde os círculos verdes e azuis são dados os quais podem ser separados pelas retas azul, vermelha e marrom. Aplicando essa técnica para predição do uso de recursos, pode-se por exemplo agrupar diferentes configurações de uma aplicação e predizer seu tempo de execução, baseado no grupo em que esteja inserido [54]. É importante ressaltar que essa técnica tem a desvantagem de depender de um conjunto massivo de dados para ajustar seu aprendizado, sem o que se torna ineficaz.

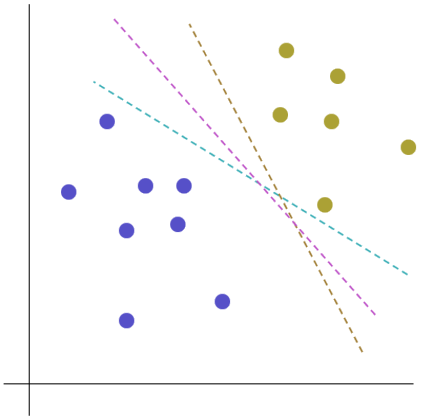

Figura 2.10: Esquema de classificação de SVM, adaptado de [54].

Redes Neurais

A técnica de Redes Neurais (RN) é um modelo computacional matemático inspirado no sistema nervoso central do cérebro [56]. A técnica de aprendizagem das RN é classificada como aprendizado supervisionado ou por exemplos [9]. Esse modelo é capaz de realizar o reconhecimento de padrões através de várias unidades de processamento (neurônios), geralmente conectadas por canais de comunicação que estão associados a determinados pesos. [57]. No aprendizado ocorre um processo de mudança de comportamento obtido através da experiência construída ao decorrer do tempo e o reconhecimento de padrões é a classificação de objetos dentro de um número de categorias ou classes. As unidades fazem operações apenas sobre seus dados visíveis (locais), que são entradas recebidas por suas conexões. O comportamento inteligente de RN ocorre pelas interações entre as unidades de processamento da rede.

Para ilustrar um exemplo simples do funcionamento de RN é apresentada a Figura 2.11. De forma geral, a operação em uma unidade de rede ocorre da seguinte maneira: as setas de entrada são as entradas e os primeiros neurônios (círculos rosas) recebem esses dados; os quais são repassados aos neurônios localizados ao centro (círculos laranjas) para tratá-los e transformá-los em informações úteis. Finalmente, o último neurônio (círculo azul) recebe as informações e apresenta a saída.

As RN também são utilizadas para predição de uso de recurso. O uso dessa técnica de predição ocorre, majoritariamente, em modelos que buscam mensurar e predizer recursos computacionais, como processamento, memória, armazenamento e largura de banda, entre

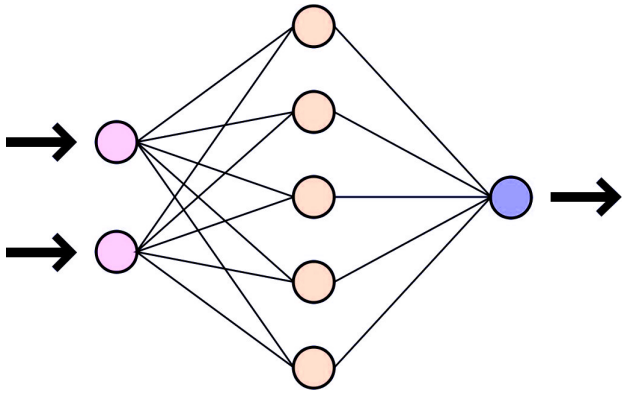

Figura 2.11: Esquema de uma rede neural, adaptado [57].

outros [49]. As RN são também utilizadas em modelos de elasticidade para predizer recursos em tempo real.

Regressão Linear e Índices de Avaliação

A regressão linear estuda o comportamento de variáveis em quantidades distintas. Em outras palavras, mede o grau de associação entre essas variáveis através de uma relação linear [58]. Através da regressão linear é possível estimar um valor com base em dados de entrada. A regressão pode ser classificada como Regressão Linear Simples (RLS) e Regressão Linear Múltipla (RLM) [59]. A equação de RLS busca apresentar a relação entre duas variáveis, como apresentado na Equação 2.1.

$$y = \beta_0 + \beta_1 x + \epsilon \tag{2.1}$$

Na Equação 2.1 o y é a variável dependente, ou seja, o valor que se deseja predizer. O β_0 é a constante que representa a interceptação da reta no eixo y, β_1 é a constante que representa o coeficiente angular da reta, como representado na Figura 2.12, e ϵ a variável que representa os fatores de resíduos dos erros de medição.

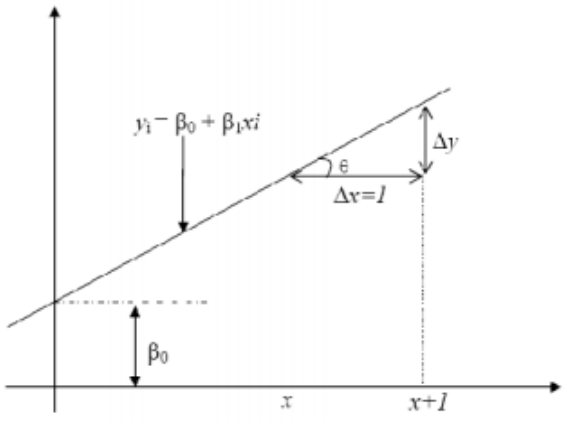

Figura 2.12: Regressão linear [58].

A RLM envolve três ou mais variáveis, ou seja, uma variável dependente (y) e duas ou mais variáveis explanatórias ($\beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k$). A descrição das variáveis segue a mesma da RLS. Então, tem o objetivo também de estabelecer uma equação que possa predizer valores de y para valores dados das k variáveis independentes. Dessa forma, pode-se dizer que a RLS possui $k = 1$. O modelo da RLM está expresso na Equação 2.2.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + \epsilon \tag{2.2}$$

Para medir a variabilidade que explica um modelo de regressão linear é utilizado o R^2 ou coeficiente de determinação. O R^2 é a porcentagem da variação da variável de resposta explicada pela relação com uma ou mais variáveis preditoras. Quanto maior o valor de R^2 melhor o modelo ajusta os dados. Esse modelo de validação usado de forma isolada pode se tornar tendencioso nas predições, sendo necessários outros métodos estatísticos para avaliar o modelo [60].

Os erros ou resíduos de uma regressão representam os valores de variabilidade em y não explicados por x. O menor valor dos resíduos significa uma melhor modelagem entre as variáveis dependentes e independentes. A avaliação dos resíduos tem como dependência a normalidade e a distribuídos independentemente dos erros, com média zero e desvio-padrão constante. Para validar o modelo de regressão é necessário realizar a análise dos resíduos e verificar se atendem os itens citados.

Diferentemente das técnicas de predição SVM e RN, a regressão linear geralmente consegue resultados satisfatórios quando treinada com um conjunto de dados menor. As SVM e as RN necessitam de um conjunto de dados massivos para um treinamento de qualidade. Para predição de recursos em nuvem, no caso da existência de um conjuntos de dados de execução expressivo tem-se que avaliar o custo para se obter resultados adequados, neste caso, o ideal é utilizar conjuntos menores de dados aplicando técnicas de regressão linear.

Quando é utilizado um modelo de previsão como regressão linear, faz-se necessário apresentar medidas de avaliação do modelo. Os métodos de avaliação: Bias (viés de tendências), *Mean Absolute Error* (MAE), *Root Mean Square Error* (RMSE) e *Mean Absolute Percent Error* (MAPE) são utilizados para validar a acurácia dos modelos [61] [62].

O Bias (erro médio) apresenta o desvio médio do modelo quando relacionado a uma variável. Esse método também informa o desempenho a longo prazo do modelo, erro sistemático. O resultado do Bias pode variar entre valores positivos e negativos, porém valores mais próximos de zero representam melhores resultados, conforme apresentado na Equação 2.3 [63].

$$Bias = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i) \tag{2.3}$$

Nas Equações 2.3, 2.4, 2.5 e 2.6, o N representa a quantidade de elementos na base de dados avaliada, o $\hat{y}_i$ o valor predito em i e o y_i é valor real em i.

O MAE apresenta o erro médio absoluto da diferença entre os valores preditos e reais [64]. O melhor acerto do modelo de RLM pode ser explicado pelo valor do MAE, uma

vez que quanto mais próximo de zero maior é o acerto, sendo medido também o erro total das predições. A Equação 2.4 representa o MAE [64].

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i| \tag{2.4}$$

O RMSE apresenta as diferenças individualizadas entre os valores preditos e os reais. Por meio do RMSE se expressam os erros sistemáticos e aleatórios, em outras palavras é expressa a magnitude do erro [64]. A Equação 2.5 representa o RMSE.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2} \tag{2.5}$$

O MAPE mede o tamanho do erro da regressão em termos de percentagem. Dessa forma, o menor valor do MAPE indica uma menor porcentagem de erro. O MAPE é expresso na Equação 2.6 [65].

$$MAPE = 100\frac{\sum_{i=1}^{N}|\frac{\hat{y}_i - y_i}{y_i}|}{N} \tag{2.6}$$

Tanto o Bias, RMSE e MAE apresentam os resultados nas mesmas unidades que as variáveis que estão sendo testadas, diferente do MAPE onde o resultado é apresentado em porcentagem de erros.

2.3.8 Elasticidade

Elasticidade permite aos provedores adicionar ou remover recursos, sem interrupção e em tempo de execução, para lidar com as mudanças de carga. Para os usuários da nuvem, os recursos disponíveis parecem ser ilimitados e podem ser comprados em qualquer quantidade e a qualquer momento. Os recursos podem ser adquiridos de forma rápida e automática, para atender as demandas de aumentos e reduções das cargas de trabalho [4].

A elasticidade é, frequentemente, associada à escala dos sistemas, mas há algumas diferenças. Escalabilidade é definida como a capacidade de um sistema para adicionar mais recursos para atender a uma carga de trabalho maior [66]. Elasticidade consiste no crescimento e redução dos recursos de acordo com a carga de trabalho, enquanto a escalabilidade consiste na adaptação, quando adicionados novos recursos. Além disso, a escalabilidade é uma noção livre de tempo e não leva em consideração quanto tempo é necessário para que o sistema atinja sua capacidade máxima, enquanto que para a elasticidade, o tempo é um aspecto central, depende da velocidade de resposta a uma determinada elevação na carga de trabalho.

Métodos de Elasticidade

Segundo [67], a elasticidade de instâncias de MV pode ser dividida em duas partes: métodos e modelos, conforme apresentado na Figura 2.13.

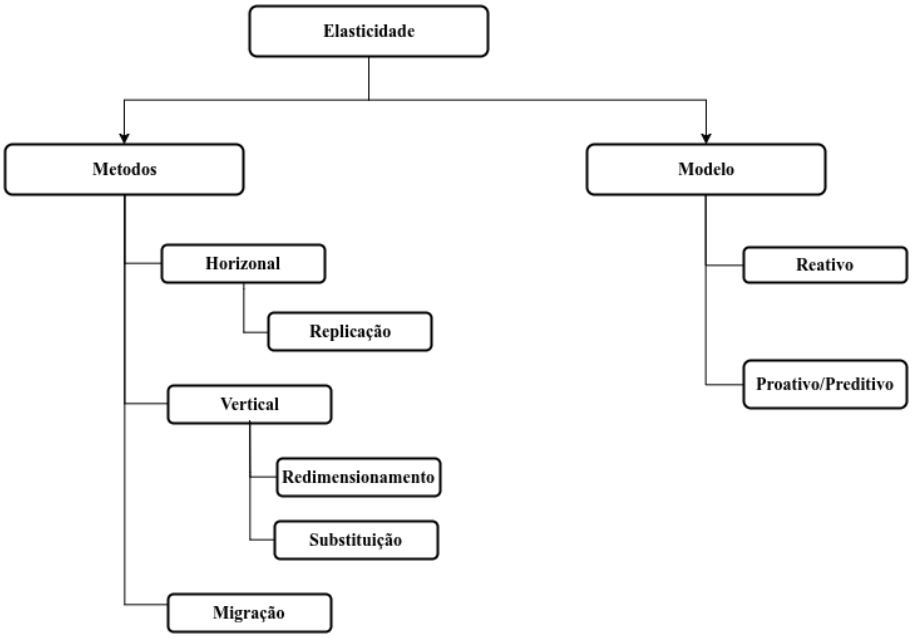

Figura 2.13: Classificação de elasticidade (traduzida de [67]).

Em relação aos métodos, estes podem ser:

- Elasticidade Horizontal (replicação) que consiste em criar ou remover instâncias de MV no ambiente virtual do usuário. Atualmente, a replicação é o método mais utilizado para realizar elasticidade;
- Elasticidade vertical consiste em aumentar ou diminuir os recursos direto na própria instância. Em [67] encontram-se duas abordagens para realizar escalonamento vertical: (1) redimensionamento – capacidade de alterar os recursos das instâncias em tempo de execução, atribuindo por exemplo, mais memórias ou CPUs às MV; (2) substituição – consiste em substituir o servidor menor por um servidor maior capacidade;
- Elasticidade por migração consiste em transferir uma máquina virtual entre servidores ou aplicativos entre máquinas virtuais. As SVM podem ser movimentadas para realizar balanceamento de carga por exemplo. Os aplicativos podem ser movidos para MVs que atendem melhor suas necessidades [67].

Existem outras classificações de elasticidade horizontal e vertical tal como citado em [39].

Modelos de Elasticidade

O modelo de elasticidade reativa comporta-se de acordo com a carga de trabalho. São colocados limites na utilização dos recursos, de forma que ao acontecer violação desses

limites ou em caso de violação de SLA a elasticidade é acionada. Nesse modelo, o sistema não antecipa a necessidade, ele apenas reage.

No modelo de elasticidade proativa/preditiva são usadas técnicas de previsão para determinar quando a carga de trabalho excederá a capacidade atualmente provisionada, então é chamado o algoritmo para alocar servidores ou MVs adicionais antes que essa capacidade seja excedida. As técnicas pró-ativas, geralmente, baseiam sua decisão em uma base histórica.

No Capítulo 3 serão apresentados diversos trabalhos correlatos antes de apresentar a proposta de solução.

Capítulo 3

Trabalhos Correlatos

O uso de SMA em nuvem pode ser feito para diversos fins, seja nas aplicações em execução, seja na própria arquitetura do provedor de nuvem. Exitem diversas ferramentas com abordagem multiagentes para ambiente de nuvem computacional, ferramentas estas que se diferenciam pela metodologia, técnica e a quais serviços da nuvem estão aplicadas. Para todas, é possível citar vantagens e desvantagens, de acordo com o objetivo da pesquisa.

Nesta seção são apresentados trabalhos que possuem ligação com o atual trabalho. As ferramentas relacionadas aos trabalhos correlatos foram escolhidas conforme citadas na literatura, priorizando a recencia das publicações. A Tabela 3.1 apresenta uma comparação entre os trabalhos apresentados neste capítulo.

+Cloud

+Cloud [2] é uma plataforma multiagente desenvolvida pelo grupo de pesquisa BISITE[1] baseada no paradigma de nuvem computacional. Esta plataforma oferece serviços nos níveis de PaaS e SaaS. A plataforma não oferece serviço em nível de IaaS aos usuários finais, apesar de sua arquitetura ter um controle sobre esse nível de serviço. Os usuários possuem uma área de trabalho virtual personalizada, na qual têm acesso as ferramentas para desenvolver aplicações no ambiente de nuvem em sua área de trabalho e o usuário pode personalizá-la como preferir.

Na camada de SaaS estão as aplicações de gestão do ambiente, além de outros aplicativos de terceiros, que também utilizam os recursos da camada PaaS. A camada PaaS oferece serviços de identificação de usuário e aplicações, também são oferecidos banco de dados e armazenamento de arquivos. Dentro da arquitetura desenvolvida, a camada interna é composta do meio físico que permite a obtenção, monitoramento e gerenciamento de recursos em forma de MV. No entanto, o serviço de IaaS é interno à arquitetura ficando transparente aos usuários. O +Cloud utiliza agentes em sua arquitetura para gerenciar os recursos disponíveis na infraestrutura física.

O agente *Local Resource Monitor* é responsável pelo monitoramento de recursos de *hardware* e guarda as informações capturadas em vetores. As Informações coletadas são atributos do servidor, como capacidade computacional, memória, estado da MV e capacidade de processamento. Além disso, armazena as informações dos servidores físicos. O agente *Local Manager* é responsável por alocar os recursos de um servidor físico entre

[1] https://bisite.usal.es/es

MVs. Cada servidor físico tem um *Local Manager*, que pode garantir a elasticidade (horizontal) modificando dos recursos em uso. A redistribuição de recursos é feita de forma a otimizar a utilização o uso de recursos físicos entre as MVs e suas tarefas. O agente *Local Resource Monitor* agrupa as informações das MVs e repassa ao *Local Manager*. Já o *Local Manager* identifica a superutilização de algum recurso ou o agente *Service Supervisor* identifica a diminuição na qualidade de serviço, sendo iniciado um processo para adaptação dos recursos nos servidores físicos e redistribuição entre as MVs.

Nos experimentos realizados em sua própria infraestrutura de nuvem, foram feitas sobrecargas para testes de mudanças nas demanda dos serviços. Os resultados mostraram a adaptação do modelo para distribuição dos recursos. Os experimentos tiveram seu foco no uso de CPU e no uso de memória. Em seus resultados também foi notório que o processo de redistribuição de recursos elevou o QoS.

A principal diferença entre o trabalho de [2] e este trabalho é que neste a elasticidade horizontal visando o menor custo para uma aplicação especifica, ou seja, a arquitetura aqui proposta foca na utilização dos recursos pela aplicação do usuário.

Elastic JADE

O trabalho de [68] propõe um modelo que não envolve apenas MV em nuvem, mas também máquinas físicas locais que podem ser estendidas com MV na nuvem. E utilizada uma arquitetura de SMA implementada com o *framework* JADE. Quando o usuário inicia a aplicação, é iniciada uma plataforma JADE na máquina local, os agentes são iniciados nessa máquina. Os agentes, após criados, monitoram continuamente e controlam os recursos disponíveis, até que sejam finalizados. Quando os recursos excedem os limites determinados na máquina local, o agente de gerente na máquina local cria uma nova instância de MV na nuvem pública da Amazon EC2, dessa forma a arquitetura do modelo trabalha em nível de IaaS.

A estratégia abordada por [68] é iniciar um agente de gerenciamento nas instâncias locais e outra na nuvem. O agente de gerenciamento local controla todas as instâncias remotas e coordena os agentes entre as instâncias, propiciando comunicação por meio de mensagens usando os protocolos de padrão FIPA.

Da mesma forma que ocorre na instância local, quando os recursos da MV em nuvem estão com superutilizações, o agente de gerenciamento localizado na MV com limite extrapolado inicia uma nova máquina e mantém o controle dessa nova instância. Assim o agente de gerenciamento deve manter a coordenação das máquinas que tenham sido criados. Em outras palavras, devem ser gerenciadas as plataformas que foram iniciadas antes e depois daquela instância. As configurações e quantidades de recursos para cada nova instância são criadas igualmente, mantendo as mesmas configurações, pois o modelo trabalha em uma única plataforma.

Para avaliar a arquitetura proposta por [68] foi utilizado um estudo de caso do domínio de processamento paralelo de imagens e o ambiente de IaaS da Amazon. O processamento das imagens é feito por agentes. Assim, cada agente processa uma imagem paralelamente. Nos experimentos foi utilizado uma base de dados com milhares de imagens, desta forma os recursos da máquina local são exauridos rapidamente, fazendo-se necessário a elasticidade da MV na nuvem. Com a arquitetura proposta, os agentes são manipulados entre as

instâncias criadas, mostrando que outros problemas podem também ser beneficiados por esse modelo de computação.

Divergindo de [68] este trabalho não tem foco em estender uma aplicação, uma vez que é proposto um modelo arquitetural, no qual as características permitem o uso de diferentes aplicações. A similaridade entre o trabalho de [68] e este está no monitoramento e elasticidade horizontal, feito por meio de um agente gerenciador.

DRPMC

No trabalho de [46] é apresentado o *Dynamic Resource Provisioning and Monitoring In Cloud* (DRPMC). O DRPMC é um modelo multiagente para gerenciar recursos de provedores de nuvem (IaaS), considerando a qualidade de serviço (QoS) de acordo com o nível de serviço (Service Level Agreement - SLA), além de satisfação dos usuários. É apresentado um sistema com três principais fases: monitoramento, análise e execução. O monitoramento é responsável por supervisionar recursos físicos das MVs como poder de processamento (CPU), memória (RAM), armazenamento permanente (HD), largura de banda (redes) etc, além de acompanhar as tarefas e demandas dos usuários. A análise é responsável por gerir as tomadas de decisões otimizadas em relação as MVs, que serão alocadas para cada usuário. A fase de execução realiza as alocações.

A arquitetura é dividida em duas camadas: camada de aplicação e camada de recursos. A camada de aplicação consiste em um conjunto de clientes, que podem consumir uma ou várias MVs. Os recursos são conjuntos de *datacenters* e suas máquinas físicas (*hosts*). Cada máquina física hospeda várias MVs, que consomem os recursos disponíveis. Na arquitetura de SMA proposta se inclui a classe de agente *Local Utility* e o agente *Global Utility*. Para cada cliente é atribuído um agente *Local Utility*, responsável por monitorar os recursos e reformular suas solicitações com base em uma análise de RLS do histórico para prever a quantidade de recursos, que serão realmente usados, sem causar uma violação de SLA. O agente *Local Utility* envia as solicitações reformuladas para o agente *Global Utility*, que é responsável pelo o provisionamento real dos recursos, verificando a disponibilidade em todos os *datacenters* visualizados pelo agente.

O modelo de provisionamento dinâmico estabelecido por [46] usa de RLS para reformular as solicitações dos usuários. Na equação a variável dependente é o número estimado de recursos, e as variáveis independentes são recursos atualmente utilizado e o histórico de desperdício. Além disso, é considerada uma variável (B) para agir como corretora dos erros da regressão. No modelo ainda é considerado sempre um valor super-estimado, a fim de evitar violações de SLA. O autor ainda apresenta um novo algoritmo de seleção de MV chamado algoritmo de detecção de falha no Host (Host Fault Detection - HFD). Esse algoritmo move as MV entre as máquinas físicas, quando ocorre alguma extrapolação de recursos.

Para validação do DRPMC foram realizados experimentos em um ambiente de nuvem simulada, o CloudSim [69]. Foram simuladas cargas de trabalho dinâmicas por um período de 24 horas em um ambiente com 50 MVs de 4 diferentes configurações e 50 *hosts*. Nos experimentos os autores identificaram que realmente ocorreram melhorias, uma vez que mesmo quando o usuário solicitava mais que o necessário para uma dada tarefa, a predição ajustava os recursos de acordo com seu histórico. O HFD também foi avaliado nos experimentos, mostrando-se superior, quando comparado a outros algoritmos de

balanceamento entre máquinas físicas. Os resultados mostram que o sistema DRPMC permite que o provedor de nuvem aumente a utilização de recursos e diminua o consumo de energia, evitando violações de SLA.

Os agentes locais do trabalho de [46] realizam o monitoramento dos clientes e constroem um histórico de utilização, possuindo também um agente para gerenciar as novas instâncias, o que faz se aproximar da estratégica do atual trabalho. O trabalho de [46] difere do atual, pois possui foco na violação do SLA e QoS por parte dos provedores, e o atual concentra-se no monitoramento, provisionamento dinâmico e elasticidade para aplicações submetidas ao modelo.

A Tabela 3.1 é um resumo dos trabalhos correlatos apresentados anteriormente comparado com o modelo proposto nessa dissertação. Note que foram comparados o modelo de serviço das propostas, as técnicas de raciocínio quando utilizadas (e.g., CBR – Case Based Reasoning no CBR-BDI em +Cloud), serviços garantidos (monitoramento, predição de recursos, provisionamento, elasticidade), e o foco do modelo (usuário, provedor).

É importante ressaltar que nenhum dos trabalhos tem o foco no interesse do usuário, como por exemplo, tempo de execução de uma aplicação submetida na nuvem, custo financeiro e menor desperdício de recursos alocados. Tais características dão a esse trabalho motivos para justificar sua originalidade. Ressalta-se ainda que o raciocínio dos agentes nessa proposta utiliza dedução lógica com raciocínio baseado em regras de inferência o que viabiliza a autonomia na tomada de decisão dos agentes para o provisionamento de serviços de forma dinâmica (monitoramento, predição e provisionamento de recursos, elasticidade).

Tabela 3.1: Trabalhos Correlatos

		+Cloud	Elastic JADE	DRPMC	Esta Proposta
Modelo de serviço		IaaS, PaaS	IaaS	IaaS	IaaS
Técnica de Raciocínio		CBR-BDI	-	-	Regras de inferência
Serviços garantidos	**Monitoramento**	Distribuído	Local	-	Local
	Predição de recursos	Análise combinatória	-	RLS	RLM
	Provisionamento	Dinâmico	Estático	Dinâmico	Dinâmico
	Elasticidade	Vertical proativa	Horizontal reativa	Vertical proativa	Horizontal reativa
Foco do modelo		Provedores	Usuário	Provedores	Usuário

No Capítulo 4 será apresentado a proposta de SMA que monitora e prediz o uso de recursos de um determinado sistema na nuvem computacional.

Capítulo 4

Desenvolvimento da Solução

Foi desenvolvido um SMA que monitora e prediz o uso de recursos de um determinado sistema em ambiente de nuvem computacional. Para alcançar esse objetivo faz-se necessário analisar o sistema em execução, além de executar testes de seu comportamento em instâncias de MVs com diferentes configurações para predição de uso de recursos. É importante ainda observar não apenas a variação do ambiente, mas também a variação na própria aplicação submetido, de forma a identificar padrões de seu comportamento e criar uma base de dados com suas características. A aplicação submetida deve ter pelo menos um parâmetro significativo definido pelo usuário, de modo que este parâmetro dite a variação no comportamento da aplicação.

A captura dos dados é realizada por meio do monitoramento dos recursos utilizados, os resultados são armazenados para serem analisados posteriormente. Utilizando-se de RLM nos dados capturados é possível predizer o uso dos recursos e através de regras em um motor de inferência escolher a instância de MV, que melhor executa aquela aplicação.

Na Seção 4.1, é descrito a arquitetura proposta, na Seção 4.2 são apresentadas as características dos agentes, na Seção 4.3 são descritas as regras de racionalidade dos agentes e, finalmente na Seção 4.4 é exposto o MASE-BDI a aplicação que será submetida para validação do modelo.

4.1 Arquitetura Geral

A arquitetura do SMA foi projetada de forma modular, para que cada serviço oferecido seja implementado e utilizado separadamente, o que permite a modificação de cada módulo sem grandes impactos na arquitetura geral. A Figura 4.1 mostra (a) os serviços garantidos na nuvem: provisionamento dinâmico, monitoramento de recursos, elasticidade e recuperação da aplicação; (b) os módulos da arquitetura: Interação com usuário, gerentes e agentes, raciocínio/inferência e as instâncias de MV.

A Figura 4.2 apresenta a arquitetura geral do sistema proposto dividida em três camadas (de cima para baixo): usuário, *core* (contendo agentes e regras de inferência) e MVs. A Figura 4.2 também apresenta o fluxo de execução dos módulos arquiteturais.

Na arquitetura apresentada na Figura 4.2, a camada de interação com o usuário é feita por meio de uma interface *web*, podendo ser também executada em um terminal com acesso direto à instância *master*, a qual passará os parâmetros para início da execução da aplicação. Nesta camada as Tarefas 1 e 14 representam a interação do usuário

com a camada *core*, através da submissão da aplicação e a entrega dos resultados, respectivamente.

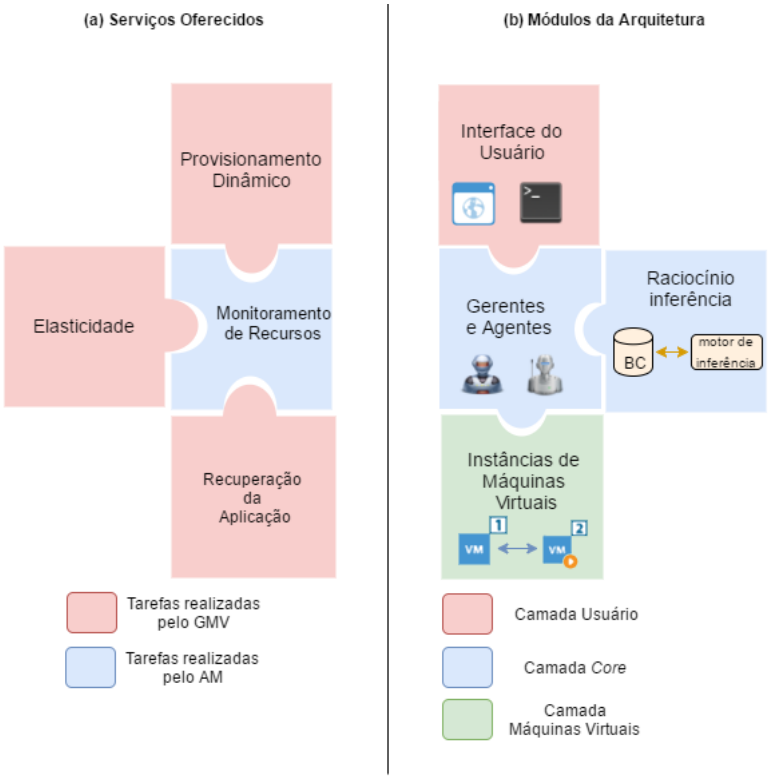

Figura 4.1: Esquema geral da arquitetura contendo: (a) arquitetura modular dos serviços oferecido; e (b) módulos da arquitetura.

A camada *core* (agentes e regras) é o núcleo da arquitetura, sendo formado pelos agentes: gerente de máquinas virtuais (GMV), gerente de monitoramento (GM) e agente de monitoramento (AM). O GMV é a entidade principal da arquitetura, sendo este responsável por receber os parâmetros do usuário e iniciar todas as etapas seguintes, escolher as novas instâncias de MV baseado nas regras de predição, além de iniciar os outros agentes. O GM é responsável por gerenciar os recursos monitorados junto às regras de monitoramento. Note que o GMV é executado na MV *master* enquanto os agentes GM e AM são executados na MV *slave*. Para ressaltar essa diferença está em verte a MV *master* e em vermelho a MV *slave*. Nesta camada as Taregas de 2-6 e 8-13 executam atividades relacionadas aos serviços dos agentes AM, GM e GMV relacionando-os aos bancos de conhecimento de regras de inferência de monitoramento e predição.

Na camada de MVs localizam-se as instâncias em que são feitas as execuções, possuindo duas principais: a MV *master* onde é executado o GMV e a MV *slave* onde é executada a aplicação submetida pelo usuário e os agentes AM e GM. As Tarefas 7 e 2 (indiretamente) são executadas pelos agentes AM e GMV, respectivamente. Uma descrição mais detalhada de cada camada da arquitetura é apresentada na sequência.

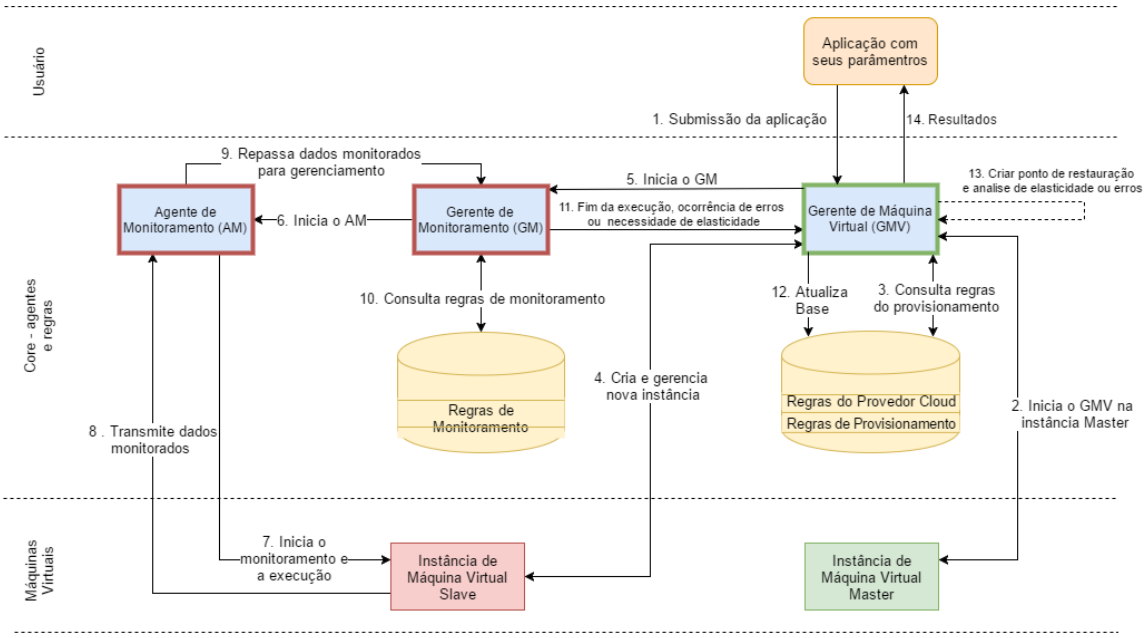

Figura 4.2: Arquitetura geral com o *workflow* de execução dos módulos.

Camada de Usuário

A camada de interação com o usuário recebe as requisições e retorna os resultados por meio da interface *web*. A requisição do usuário é composta por três parâmetros de entrada: *QuantidadeExecução* - representa a quantidade de vezes que será executada a aplicação do usuário; *ParametroSignificativo* - informa qual parâmetro da aplicação é o mais significativo, ou seja, o parâmetro que tenha maior influência no desempenho da aplicação; e o *PassoParametroSignificativo* - qual o valor de incremento do parâmetro significativo na aplicação do usuário.

Nessa camada serão apresentados os resultados da execução da aplicação do usuário submetida, contendo: os dados da aplicação e do respectivo monitoramento, além dos resultados gerados pela execução da aplicação do usuário.

Camada *Core* – Agentes e Regras

Na camada principal da arquitetura estão os principais módulos, em que todas as ações dos agentes são realizadas. Esta camada possui duas partes importantes: os agentes - GMV, GM e AM - e as bases de regras de inferência utilizadas por esses agentes. Nas Seções 4.2 e 4.3 serão detalhados os aspectos de funcionamento desses componentes arquiteturais.

Camada de MVs

A última camada do modelo arquitetural é composta pelas instâncias de MVs. Essa camada foi projetada para ter n MVs, existindo sempre uma única máquina *master* e $n-1$ máquinas *slave*. Dessa forma, a execução da aplicação do usuário é submetida nas MVs *slave*.

4.2 Descrição dos Agentes

Na arquitetura apresentada na Figura 4.2) existem três tipos de agentes - GMV, GM e AM - conforme descrito na Seção 4.1, os quais serão detalhados na sequência.

Um agente AM só se comunica com o gerente imediatamente superior na hierarquia - o agente GM. O agente gerente GM só se comunica com o agente superior - o agente GVM. A comunicação ocorre com os protocolos de interação da FIPA ACL apresentado na Seção 2.1.3. O protocolo de interação utilizado é cooperativo hierárquico conforme as atribuições dos tipos de agentes. Pela essência do projeto, a interação dos agentes ocorrerá por cooperação, de forma a obter um maior desempenho no modelo com a mínima intervenção de humanos.

4.2.1 Gerente de Máquinas Virtuais - GMV

O GMV tem o papel de coordenador geral na arquitetura, uma vez que ele faz a comunicação entre a camada de interação com o usuário e as MVs. O agente GMV é baseado em objetivos e sua função principal é predizer o uso dos recursos na nuvem computacional para que a aplicação submetida pelo usuário seja executada a contento, utilizando para isso regras de inferência de provisionamento. Além disso, o GMV instancia as MVs com as características escolhidos pelo usuário, inicia o agente GM e mantém atualizada a base de dados ao final de cada execução. Faz parte também de suas funções realizar a elasticidade, quando o GM solicitar. Além disso, o GMV garante a continuidade da execução da aplicação, identificando a ocorrência de falhas de execução da aplicação nas MVs *slave* e reiniciando a execução da mesma. O GMV ainda trata ocorrências de falhas na MV, identificando-as e executando a aplicação a partir de ponto de falha.

Provisionamento Dinâmico

No provisionamento de recurso, o agente GMV utiliza técnicas de RLM para predizer o tempo de execução e a média de uso de CPU. A base de cálculo utilizada é o parâmetro significativo da aplicação informado pelo usuário e a quantidade de CPU disponibilizada pelo provedor. A forma de cálculo poderá ser utilizada para todos os demais recursos de uma MV na nuvem. Neste trabalho, o foco do provisionamento de recursos é a CPU tendo em vista que a aplicação utilizada como exemplo (MASE-BDI) é *CPU bound*. *CPU bound* refere-se as aplicações em que o tempo de execução é diretamente relacionado ao poder de processamento da CPU [70].

Para essa dissertação foi necessária a escolha de um provedor de nuvem pública, no qual pudessem ser realizados os experimentos. Um fator importante na arquitetura do modelo é a criação de novas instâncias de MV em tempo de execução, que leva em consideração a liberdade na escolha dos recursos ao criar uma MV. Dessa forma, para realização dos experimentos foi utilizado o ambiente de IaaS do Google Cloud[1].

Com a escolha do provedor Google Cloud, a quantidade de memória de cada MV foi configurada automaticamente com o mínimo para aquela quantidade de CPU, conforme apresentado na Tabela 4.1. Os valores foram ajustados de acordo com os limites estabelecidos pela plataforma do provedor[2]. Cada *virtual* CPU (vCPU) é um único em um

[1]Google Cloud - http://cloud.google.com

[2]Google Cloud Platform Pricing Calculator - https://cloud.google.com/products/calculator/

processador Intel Xeon de 2,6GHz.

Tabela 4.1: Relação entre a quantidade de CPU e quantidade mínima de memória fornecida pelo *Google Cloud Platform*

Quantidade de CPUs	**Quantidade Mínima de Memória**
1	0.9 GB
2	1.9 GB
4	3.6 GB
6	5.4 GB
8	7.2 GB

Nas Equações 4.1 e 4.2 os parâmetros são ajustados a partir dos dados experimentais na base de dados relativas ao tempo ao uso de CPU. Nessas equações o tempo é representado por Tx e a CPU usada é representada por Cx com os coeficientes de RLM (e.g., T1, T2, T3, C1, C2 e C3). O parâmetro significativo é representado por PS e a quantidade de CPUs é representada por QC. O uso dessas equações é realizado para todas as combinações de parâmetro significativo e quantidade de CPUs do provedor.

$$Tempo = T1 + T2 * PS + T3 * QC \tag{4.1}$$

$$CPUUsed = C1 + C2 * PS + C3 * QC \tag{4.2}$$

Depois de realizados os cálculos de predição, o GMV usa as regras de inferência para selecionar a configuração de CPU da MV, que irá executar a aplicação do usuário. No projeto proposto foi necessário escolher um motor de inferência que fornecesse integração ao JADE, além de manter o desempenho. Foi buscado no Capítulo 2.2.3 uma alternativa de software livre que integra facilmente e de forma unificada as cargas de trabalho, a BC e o mecanismo de regras. Pelas características citadas, as regras foram implementadas no motor de inferência DROOLS. O motor de inferência DROOLS ainda permitiu que as regras fossem ajustadas dinamicamente de acordo com a base de dados [29].

As regras de provisionamento estão divididas em duas categorias: as regras de seleção de MVs candidatas; e regras para escolher entre as candidatas a que tenha o melhor custo benefício, baseado na forma de cobrança do provedor da nuvem, apresentadas na Seção 4.3.

As regras foram definidas para escolher as CPUs que tivessem o menor desperdício de recursos, que executassem no menor tempo possível e apresentassem o melhor custo benefício. Em outras palavras, se buscou escolher equilíbrio entre o maior uso de CPU, o melhor tempo e o menor preço.

Elasticidade

A elasticidade é feita de modo horizontal, em que são instanciadas novas MVs no provedor da nuvem. Assim, quando a instância da MV atual está fora dos limites definidos nas regras de monitoramento, uma nova instância é iniciada [68]. O módulo de elasticidade utiliza uma abordagem de ponto de restauração do modelo. A elasticidade ocorre por um conjunto de regras no DROOLS, apresentadas na Seção 4.3. O GM recebe os dados

do AM e faz a verificação no conjunto de regras, conforme apresentado no *workflow* da Figura 4.2.

A regra para executar a elasticidade deve garantir que a transição de uma máquina para outra não seja onerosa para o tempo e custo. Para garantir que a elasticidade ocorra sempre no momento apropriado, a regra implementada verifica se está ocorrendo desperdício de recurso, ou seja, se o usuário está pagando por recursos e não está utilizando os mesmos. Se acionada a regra, o agente GM informa ao agente GMV os dados da execução. Neste caso, primeiramente o agente GMV finaliza a MV em que está aquela execução da aplicação e inicia uma nova MV com os recursos ajustados, conforme apresentado no diagrama da Figura 4.3.

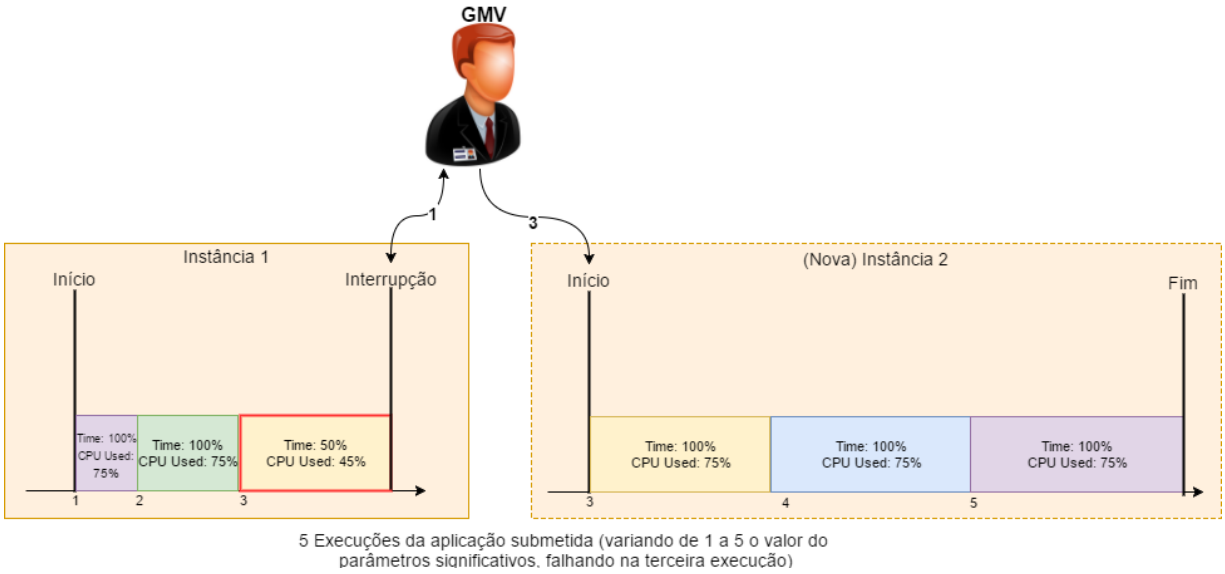

Figura 4.3: Módulo de elasticidade horizontal.

Para justificar a necessidade de realizar a elasticidade, as regras que serão acionadas devem garantir que na maioria das vezes, o tempo de execução não será drasticamente prejudicado, que não haverá um aumento de custo e que não será feito em um momento impróprio, por exemplo, uma parada da execução quando a aplicação já estiver muito próxima da sua finalização. Para tal, a regra de elasticidade foi implementado com as variáveis de tempo da execução, uso de CPU e o valor de custo mínimo do provedor da nuvem, ou seja, é necessário que a aplicação submetida informe ao GMV o quanto de sua execução já foi concluído e o quanto ainda falta executar. O uso de CPU é capturado em tempo de execução pelo AM e o valor de custo mínimo do provedor é capturado diretamente com o provedor da nuvem, como apresentado na Equação 5.2.

$$PrecosProvedor : \left[\{local : local1, preco : valor1\}, \quad \{local : local2, preco : valor2\}\right] \tag{5.2}$$

$$menorValor = \min\{PrecosProvedor.preco\} \tag{5.3}$$

Recuperação da Aplicação

Para garantir que a aplicação submetida pelo usuário seja executada com sucesso, o modelo possui um módulo de recuperação da aplicação. Esse módulo foi implementado com a abordagem de controle de execução, onde cada início de execução é marcado, podendo ser posteriormente recuperado [71].

Na Seção 4.1 é apresentado que o usuário fornece quantas vezes deseja que seja executada a aplicação submetida. Dessa forma, a cada início de uma das execuções da aplicação submetida é feito um ponto de controle de restauração para o modelo. Dessa forma, se durante a execução houver falha (vide execuções 3 e 5 no Eixo X da Figura 4.4) é restaurada a execução a partir do ponto de controle onde ocorreu o erro (vide barras de recuperação acima das execuções 3 e 5 marcadas em vermelho na Figura 4.4), não sendo necessário executar a aplicação antes do seu ponto de falha.

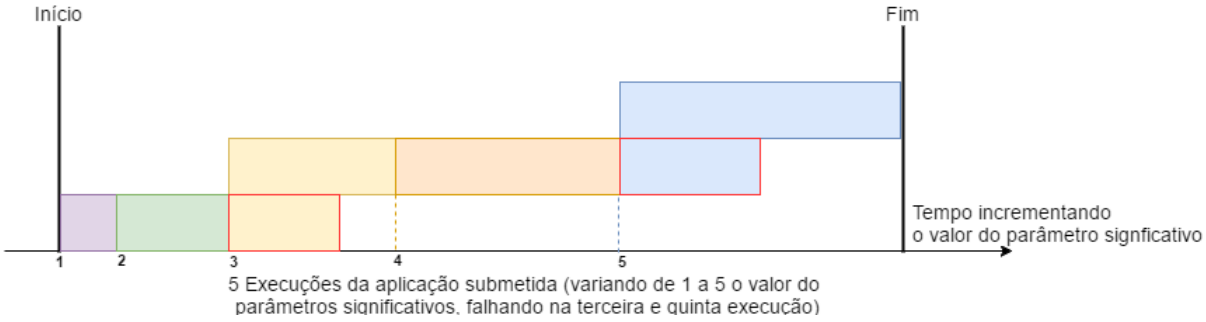

Figura 4.4: Esquema de recuperação da aplicação.

Pode-se citar como falhas tratadas pelo módulo de recuperação da aplicação: execução da aplicação submetida não concluída com sucesso; perda de comunicação com a MV (*slave*), que está executando a aplicação submetida; e falha na MV (*master*) que está executando a atual arquitetura.

Nos casos de erros em que a aplicação não é concluída com sucesso ou há perda de comunicação com a MV *slave* (como apresentado na Figura 4.4), a estratégia do GMV é excluir a MV em que estava sendo executado e criar uma nova MV, reiniciando a execução a partir do ponto onde ocorreu a falha. É mantida uma base de dados com os estados de cada execução, permitindo que o agente GMV identifique a falha e possa reiniciar a aplicação a partir do ponto de restauração necessário.

Quando ocorre falha no próprio modelo ou na MV *master*, a estratégia adotada é executar um *script shell*[3] que reinicia toda MV verificando na base a última execução concluída com sucesso. Neste caso é reinicializado a execução a partir do ponto de falha, não afetando as outras execuções.

4.2.2 Gerente de Monitoramento - GM

O GM é orientado a objetivos (vide Seção 2.1.4) e existe um por MV. Suas principais funções são gerenciar o monitoramento de recursos, recebendo os dados do AM e verificando

[3] *Script Shell* é um algoritmo projetado para realizar uma determinada tarefa, utilizando os comandos específicos e os executáveis do sistema operacional.

na base de regras para identificar problemas na execução ou nos dados monitorados. É parte das funções do GM manter a comunicação com o GMV. Através desta comunicação, o GM informa ao GMV quando existe a necessidade de realizar elasticidade.

A principal finalidade da arquitetura é deixar o AM responsável apenas pela captura dos dados, ficando o GM responsável por analisar, consultar a base de regras e comunicar ao GMV as informações da execução da aplicação, que estão em sua MV.

4.2.3 Agente de Monitoramento - AM

O AM é do tipo reflexivo simples, um para cada recurso de cada MV. A principal função é capturar os dados referentes aos recursos alocados, tratá-los e repassar ao agente GM.

Monitoramento

O agente AM utiliza a ferramenta Dstat [4] para realizar o monitoramento e obter as informações da MV. A escolha do Dstat ocorre pelo fato de fornecer as informações de forma clara e organizada, além de ser facilmente customizável para capturar apenas informações desejadas. O Dstat ainda permite exportar os dados para CSV, apresentando estatísticas em tempo de execução.

O comando utilizado para captura dos dados é "*dstat -cmnd –noheaders –output stats.csv*"(Figura 4.5). O parâmetro "cmnd"informa para o Dstat que a captura terá as colunas dos dados de CPU, Memória, Rede (*network*) e disco. Além disso, os dados são salvos em um arquivo CSV externo (stats.csv). Os dados da execução ainda ficam salvos em estruturas de dados na memória da MV para serem lidos e processados com maior agilidade durando a própria execução da aplicação submetida.

```
~ $ dstat   -cmnd --noheaders --output stats.csv
----total-cpu-usage---- ------memory-usage----- -net/total- -dsk/total-
usr sys idl wai hiq siq| used  buff  cach  free| recv  send| read  writ
 13   3  82   2   0   0|4227M 15.4M 1204M  408M|   0     0 |  42k   81k
 20   4  73   3   0   0|4228M 15.4M 1204M  407M| 115k 5596B|   0     0
 18   4  75   3   0   0|4229M 15.4M 1204M  406M| 102k 5633B|  88k    0
 25   4  69   3   0   0|4230M 15.4M 1204M  406M|  11k 1155B|   0     0
 21   4  72   4   0   0|4230M 15.4M 1204M  406M|  11k 2242B|   0   260k
 25   3  70   2   0   0|4229M 15.4M 1200M  411M|  18k  536B|   0     0
 22   5  70   3   0   0|4227M 15.4M 1198M  414M| 159k 2174B|   0    36k
 23   6  65   5   0   0|4230M 15.5M 1199M  410M| 295k  986B| 144k   52k
 22   3  72   3   0   0|4232M 15.5M 1198M  409M| 399k 1861B|   0   180k
 24   5  69   3   0   0|4232M 15.5M 1198M  409M| 249k 1116B|   0   512k
 20   3  74   3   0   0|4232M 15.5M 1198M  409M| 490B 1087B|   0     0
 19   4  75   3   0   0|4232M 15.5M 1199M  409M| 188B  106B|   0     0
 23   4  70   3   0   0|4231M 15.5M 1199M  409M|7793B 5104B|   0     0
```

Figura 4.5: Colunas dos dados capturados pelo Dstat.

O agente AM registra os dados capturados em uma estrutura de dados (vetor) para repassar para o agente GM, de forma rápida, em tempo de execução e ao mesmo tempo salva os dados em uma base de dados (arquivo de texto) para serem incorporados na BC

[4] http://dag.wiee.rs/home-made/dstat/

ao fim daquela execução. Com isso, a BC (veja Figura 4.2) é sempre atualizada com todas as informações de execuções e com as informações de uso de recursos (atualmente está sendo capturado no tempo *default* do Dstat, por esse motivo não consta o tempo na Figura 4.5).

4.3 Regras de Inferência

Com o uso do MASE-BDI, como estudo de caso escolhido, as regras de racionalidade foram implementadas e ajustadas. As regras possuem um grande impacto sobre o modelo, uma vez que as ações dos agentes serão baseadas na modelagem das mesmas. Dessa forma, é primordial analisar o comportamento da aplicação submetida para que as regras sejam ajustadas corretamente.

As regras foram implementadas no motor de inferência DROOLS, e seu código fonte é independente do código fonte da aplicação que a utiliza. Desta forma, com a arquitetura implementada é possível submeter uma aplicação diferente e ajustar as regras de forma desassociada, apesar de ter sido somente testado nessa dissertação com a aplicação MASE-BDI.

Os agentes GMV e GM apresentam funções específicas, ou seja, possuem regras de inferência específicas. O agente GMV utiliza regras de provisionamento dinâmico de recursos e recuperação da aplicação. Enquanto o agente GM utiliza regras de inferência para realizar a elasticidade horizontal de MV. Os agentes AM não utilizam regras de inferência, pois são agentes reflexivos, apenas capturam dados monitorados armazenando-os. Nas Seções 4.3 e 4.3.2 serão detalhadas as regras de inferência definidas para os agentes GMV e GM.

4.3.1 Regras de Provisionamento

O agente GMV para escolher e provisionar a quantidade de CPUs para executar a aplicação submetida (MASE-BDI) precisa levar em consideração dois principais fatores: (1) o comportamento da aplicação em diferentes cenários de MVs; e (2) as regras de cobranças mínimas do provedor da nuvem.

A regra para provisionar uma nova MV para executar o MASE-BDI leva em consideração o parâmetro quantidade de agentes transformadores (o parâmetro significativo), como pode ser visto na Seção 4.4. Para identificar a significância desse parâmetro foram feitos diversos testes com o MASE-BDI em ambiente de nuvem real. Os resultados e discussões sobre esses testes estão apresentados na Seção 4.5.

Na Figura 4.6 é apresentada uma das regras para provisionamento, referente à primeira etapa para o processo de escolha da quantidade de CPU que serão alocadas. Essa regra tem como dados de entrada a quantidade de agentes transformadores e os dados obtidos com a RLM apresentados nas Equações 4.1 e 4.2, as quais calculam a média de uso de CPU e o tempo de execução. Esta regra combina a quantidade de agentes para todas as quantidades de CPU virtual (vCPU) disponibilizadas pelo provedor, como apresentado no exemplo da Tabela 4.2 (a qual foi calculada utilizando as Equações 4.1 e 4.2).

Para cada teste feito com a regra de provisionamento, a CPU que está sendo testada pode ou não entrar para a lista de CPUs candidatas. A regra foi modelada com base nos testes realizados com a aplicação em estudo do MASE-BDI. Dessa forma, é primeiramente

testada a quantidade com um limite de 100 agentes. Dado o comportamento do MASE-BDI pode haver quatro casos: quantidade de agentes até 30; quantidade de agentes até 70; quantidade de agentes até 85; e quantidade de agentes maior que 85. Os valores foram ajustados por meio de testes com o MASE-BDI na nuvem, mas estes intervalos podem ser ajustados de acordo com a aplicação submetida e com a influência que seus parâmetros têm em seu comportamento.

Tabela 4.2: Exemplo de entradas para regra de provisionamento do agente GMV.

Quantidade de Agentes	**Quantidade de vCPU**	**Média de uso de CPU**	**Tempo de Execução**
10	2	89.9	69.17 segundos
10	4	47.33	20.54 segundos
10	8	12.25	17.57 segundos
50	2	97.66	189.47 segundos
50	4	89.75	152.34 segundos
50	8	55.05	50.02 segundos
100	2	97.70	187.15 segundos
100	4	92.56	163.16 segundos
100	8	66.52	54.51 segundos

A média de uso da CPU é de extrema importância para escolha da CPU que será alocada para execução da aplicação. Os valores utilizados nas regras são variantes de acordo com aplicação submetida, por isso uma base de dados do comportamento da aplicação resulta em melhores resultados nos ajustes da regra. Para o MASE-BDI, a média de uso de CPU está relacionada diretamente à quantidade de agentes, como expresso na Tabela 4.3. A regra foi implementada de forma que para uma quantidade X de agentes seja escolhido o número de CPUs adequados, mantendo um desperdício mínimo de recursos e que seu tempo de execução esteja em um limite aceitável estipulado pelo experimentos do comportamento do MASE-BDI. Os valores expresso na Tabela 4.3 se baseiam nas alterações de uso de CPU para cada quantidade de agentes do MASE-BDI.

Tabela 4.3: Limites de agentes e uso de CPU

Quantidade de Agentes	**Média de uso de CPU**
<30	80%
>30 E <70	75%
>70 E <85	70%
>85	65%

Dentro da regra para selecionar o numero CPUs, existem ainda outras duas condições que precisam ser validadas: o uso médio da CPU tem de ser maior que a média geral de todas a CPUs disponibilizadas pelo provedor, calculado de acordo com os dados obtidos pelo RLM, e o tempo de execução da CPU a ser escolhida precisa ser menor do que as CPUs já listadas como candidatas (vide entrada Figura 4.6). Note que essa estratégia ajuda garantir que as CPUs candidatas tenham o menor desperdício, sendo que é criado uma lista de CPUs candidatas conforme a Figura 4.6 e o Algoritmo 4.1.

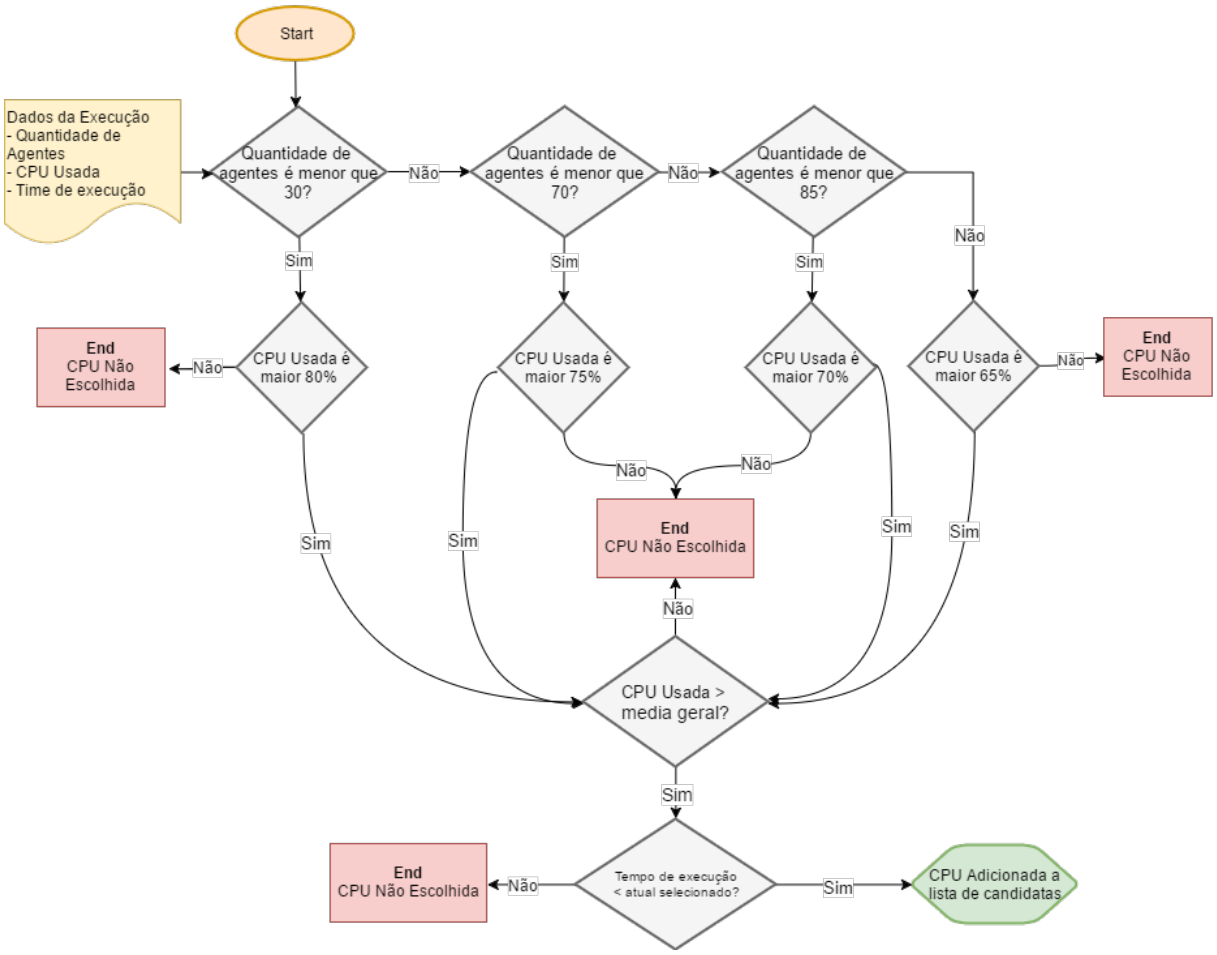

Figura 4.6: Regra de Inferência para o GMV realizar o provisionamento.

Algoritmos 4.1: Algoritmo da regra de inferência para o provisionamento.

```
1   funcao Regra de inferência para provisionamento retorna Lista de candidata
2
3   enquanto (quantidade de CPUs verificada < quantidade de CPUs disponibilizadas)
4       se (Uso de CPU > Mé dia geral de uso das CPUs E
5       Tempo de execução < Tempo melhor CPU selecionado)
6           se (Quantidade de agente < 30 E Uso de CPU > 80% )
7               Adicionar CPU a Lista de CPU Candidata
8           senao se (Quantidade de agente < 70 E Uso de CPU > 75% )
9               Adicionar CPU a Lista de CPU Candidata
10          senao se (Quantidade de agente < 85 E Uso de CPU > 70% )
11              Adicionar CPU a Lista de CPU Candidata
12          senao se (Quantidade de agente > 85 E Uso de CPU > 65% )
13              Adicionar CPU a Lista de CPU Candidata
14  retorna Lista de CPU Candidata
```

A segunda regra de provisionamento, apresentada na Figura 4.7 e no Algorítimo 4.2, garante a escolha do numero de CPUs de menor custo para o usuário. Essa escolha é feita

considerando as CPUs elencadas na lista de candidatas geradas pela primeira regra. As condições dessa regra são dependentes das formas de cobranças feitas pelo provedor de nuvem em questão. É verificado o tempo mínimo cobrado, de forma que possa ser levado em consideração o melhor custo benefício.

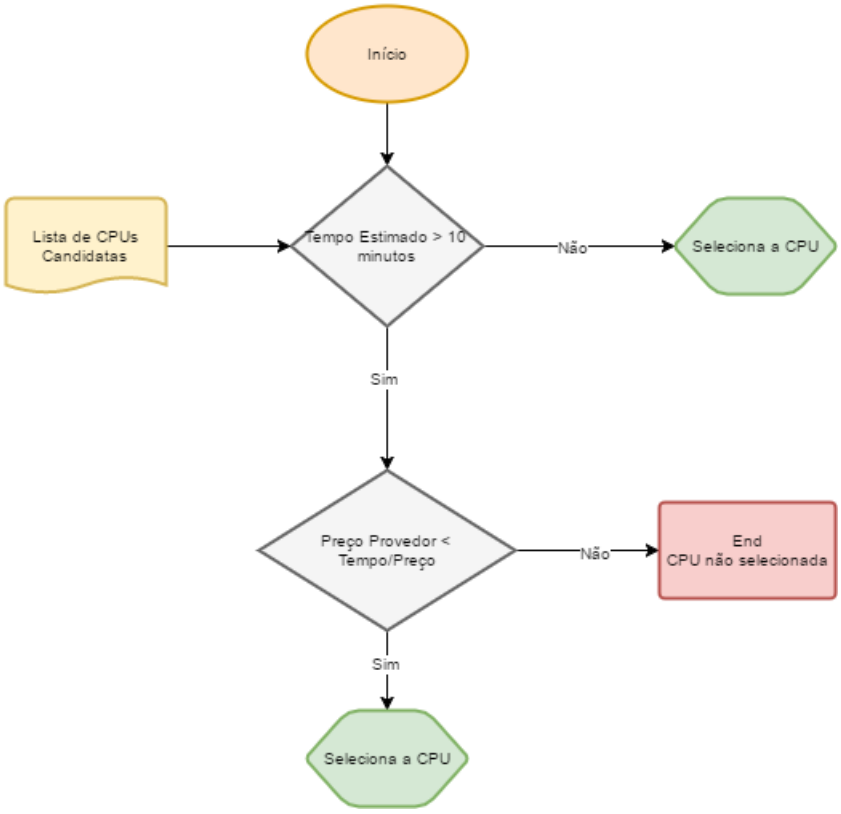

Figura 4.7: A segunda regra de provisionamento com base no dados do provedor.

Algoritmos 4.2: Algoritmo da regra de inferência para o provisionamento com base no dados do provedor.

```
1   funcao Regra de inferência para provisionamento com base no dados do provedor.
2   retorna CPU Selecionada
3
4   enquanto (Percorrer elementos da lista de CPUs Candidadas)
5       se (Tempo estimado > tempo mínimo cobrado pelo provedor)
6           Seleciona a CPU
7       senao se (Verificar custo benefício < Melhor custo benefício da
8       selecionado até  o momento)
9           Seleciona a CPU
10  retorna CPU selecionada
```

Para as condições implementadas na regra foi considerada a forma de cobrança do provedor Google Cloud[5]. Foram presentados os seguintes critérios: todos os tipos de MV são cobrados com um mínimo de 10 minutos. Por exemplo, se executar uma MV por 2

[5]Google Cloud Platform – https://cloud.google.com

minutos, será cobrado por 10 minutos. Após o limite de 10 minutos, as instâncias são carregadas em incrementos de 1 minuto, arredondadas para o minuto maior. Desta forma, uma instância que executa por 11,25 minutos será feito a cobrança por 12 minutos de uso [6]. Considerando as especificidades do provedor Google Cloud foram modeladas as regras de inferência de provisionamento utilizadas pelo GMV.

A regra de inferência das características do provedor tem como entrada a lista de CPUs candidatas definidas na primeira regra, bem como seus respectivos tempos estimados de execução. Para cada CPU na lista é primeiramente verificado se o tempo é maior que o tempo mínimo. Caso não seja, a atual CPU que está sendo avaliada pela regra é escolhida a priori. Nos casos em que o tempo de execução seja maior que o tempo mínimo é calculado o custo benefício para aquela CPU, utilizando a proporção *Tempo/Preço*. Dessa forma é escolhida a CPU com o melhor custo benefício.

Tabela 4.4: Exemplo de entradas para segunda regra de provisionamento do MASE-BDI.

vCPU	**Tempo Estimado**	**Preço estimado**
1	14 minutos	$ 0,008148
2	10 minutos	$ 0,011640
4	8 minutos	$ 0,018624
6	7 minutos	$ 0,024444
8	6 minutos	$ 0,027936

A segunda regra de provisionamento percorre a lista de CPUs candidatas da menor para maior quantidade de CPUs. Caso todos os tempos estimados estejam entre 0 e 10 minutos a regra escolhe a CPU com o melhor tempo de execução, onde 10 minutos é o tempo mínimo cobrado pelo provedor da Google utilizado neste trabalho.

A combinação das duas regras apresentadas nas Figuras 4.6 e 4.7 garantem ou possibilitam um provisionamento dinâmico, que atenda os requisitos de tempo, média de uso de CPU e custo benefício. Com essas regras o GMV tem um modelo de racionalidade dinâmica responsável por garantir um provisionamento direcionado a aplicação submetida.

4.3.2 Regra de Elasticidade

Para garantir a elasticidade no modelo, o agente GM utiliza também regras de inferência. De acordo com as classificações apresentadas na Seção 2.3.8, a elasticidade deste trabalho segue o modelo reativo, uma vez que o agente reage de acordo com a utilização dos recursos. No modelo proposto é feita a elasticidade horizontal, porém não com replicação. Na arquitetura proposta a elasticidade é transparente ao usuário, e acontece quando é identificado um desperdício de recursos (não sendo tratado quando houver sobrecarga nesses recursos). O agente GM instancia uma nova MV e transfere a execução para essa nova máquina (o *overhead* não foi tratado nesse caso).

Quando alocada a quantidade de vCPU em uma MV, a medição dessas CPUs se dá entre 0% e 100% de uso. O agente GMV realiza o cálculo do uso médio da CPU a cada segundo (de acordo com os dados extraídos pelo agente AM através do Dstat – tempo *default* usado pela ferramenta de monitoramento). É considerado desperdício de recurso,

[6]Google Compute Engine Pricing - https://cloud.google.com/compute/pricing

quando o uso médio do CPU está abaixo de 50%. Em outras palavras o usuário não está utilizando metade do que está pagando.

Para realizar o modelo de elasticidade proposto precisa-se considerar os seguintes quesitos: (1) não deve implicar em um grande aumento no tempo geral da execução; (2) o custo para trocar de MV não pode ultrapassar o da atual MV onde está sendo executado a aplicação; e (3) a elasticidade deve ser feita dentro de um período específico da execução, por exemplo, para que não seja realizada quando a aplicação submetida estiver com mais de 95% da execução completa.

Uma fragilidade do modelo de racionalidade dos agentes baseados em regras de inferência é a necessidade de se ter uma base de dados representativa da aplicação para viabilizar a definição de regras adequadas. Nesse caso, uma regra de provisionamento mal definida poderá resultar no procedimento de provisão de recursos inadequados, podendo até levar ao desperdício de recursos. Neste sentido foi definida uma estratégia de elasticidade para viabilizar o provisionamento de recursos nas execuções iniciais da aplicação, que forneça dados para a definição das regras de inferência.

A regra para elasticidade é apresentada na Figura 4.8 e no Algoritmo 4.3. Note que os dados de entrada para a regra de inferência são: o tempo de execução da aplicação já realizada no instante t, tempo mínimo cobrado pelo provedor de nuvem, média de CPU utilizada. Os dados de tempo de execução e média de CPU utilizados são capturados e submetidos a cada segundo (tempo *default* do Dstat, podendo ser ajustado), garantindo assim testes feitos a todo momento da execução da aplicação submetida.

Na primeira condição para a regra de inferência de elasticidade é verificada se a quantidade de CPU alocada para a atual MV é maior que um. Essa condição se justifica pelo fato da estratégia geral da elasticidade ser a diminuição dos desperdício. Dessa forma, se estiver ocorrendo desperdício em apenas uma CPU não é possível diminuir a quantidade. Caso a quantidade de CPUs seja maior que um, e existir a ocorrência de desperdício, é possível diminuir a quantidade de CPUs para aquela execução.

A segunda e a terceira condição da regra garante que a ação de elasticidade não ocorra nos momentos inicias da execução ou quando a execução já estiver sendo concluída. As condições garantem que a aplicação esteja acima de 10% e abaixo de 60% de seu tempo de execução. Para o MASE-BDI é possível capturar o estado da execução, pois trata-se de um simulador baseado em agentes com eventos discretos (*steps* de simulação).

A quarta condição da regra está relacionada a média de utilização da CPU. Como a prioridade da regra é diminuir o desperdício de CPU, essa condição certifica que a média de utilização esteja acima de 50% de uso da CPU alocada. No caso em que a média esteja abaixo dessa porcentagem é entendido que esteja ocorrendo desperdício, uma vez que à média de utilização não está chegando à metade do que o usuário está pagando.

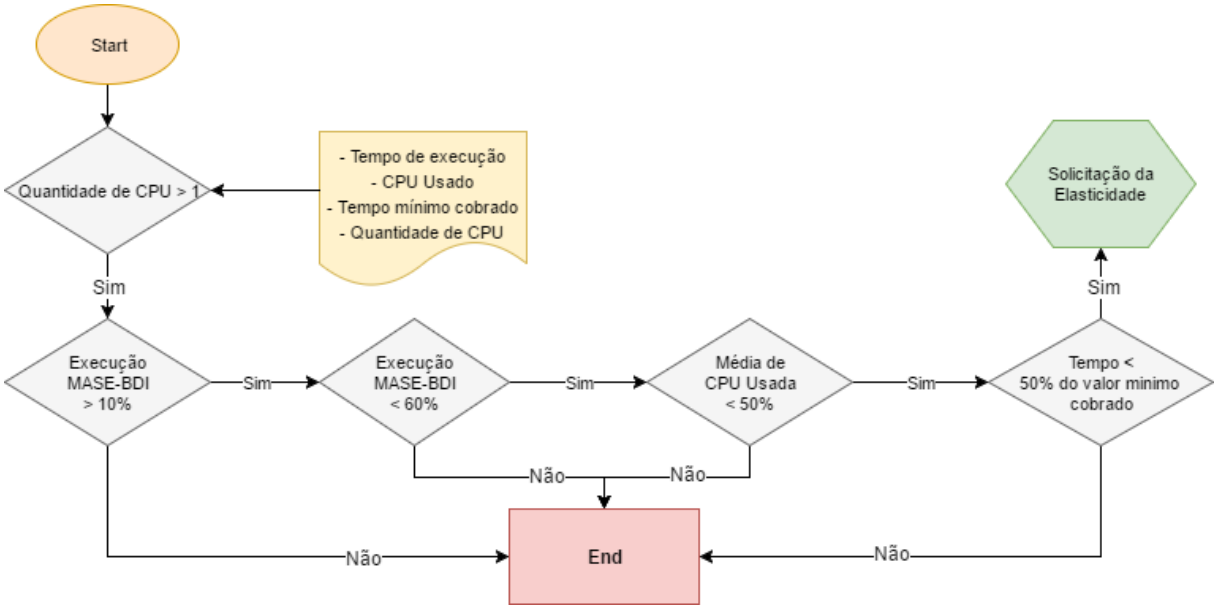

Figura 4.8: Regra de inferência para o GM realizar a elasticidade baseada nos dados monitorados.

Algoritmos 4.3: Algoritmo da regra de inferência para realizar elasticidade.

```
1  ***funcao*** Regra de inferência para elasticidade.
2  ***retorna*** Elasticidade
3
4  ***se*** (Quantidade CPU > 1 ***E*** Execução do MASE−BDI > 10% ***E***
5   Execução do MASE−BDI < 60% ***E*** Mé dia de uso de CPU < 50% ***E***
6   ***E*** Tempo de execução no momento < 50% do valor mínimo cobrado)
7            Solicita elasticidade
8   ***retorna*** Elasticidade
```

A quinta e última condição da regra é para verificar a viabilidade econômica para realizar a elasticidade, sendo levado em consideração o tempo mínimo cobrado pelo provedor da nuvem. Para que essa condição seja aceitável, o tempo de execução da aplicação submetida naquele teste da regra deve estar abaixo da metade do cobrado pelo provedor. Em resumo, a regra verifica se o tempo de execução está abaixo de 50% do tempo mínimo cobrado pelo provedor, e se a aplicação está entre 10% e 60% de sua execução. Deste modo é garantido que não ultrapasse o valor mínimo de cobrança.

Com todas as condições sendo validadas é então realizada a elasticidade, como apresentado no exemplo da Tabela 4.5. É importante salientar que o módulo de provisionamento dinâmico busca evitar que aconteça elasticidade, uma vez que a predição dos recursos é justamente para identificar os recursos necessários para cada execução. Idealmente a elasticidade é utilizada apenas para casos em que a BC e as regras de inferência não estão totalmente precisas.

Tabela 4.5: Exemplo de entradas para regra de elasticidade do MASE-BDI.

Quantidade de CPU	Tempo mínimo cobrado pelo provedor	Tempo de execução	Média de CPU usado	"Steps"de execução Mase	Elasticidade
1	10 minutos	3	40%	30%	Não
2	10 minutos	3	40%	40%	Sim
2	10 minutos	5	45%	50%	Não
4	10 minutos	2	30%	40%	Sim
4	10 minutos	3	35%	50%	Sim
4	10 minutos	4	40%	60%	Não

4.4 MASE-BDI

O MASE-BDI é um simulador de uso da terra baseado em agentes inteligentes com raciocínio BDI. Foi desenvolvido na linguagem JAVA e com *framework* JADEX (conforme apresentado na Seção 2.1.7) para implementação do seu SMA. O MASE-BDI permite a execução de simulações baseadas em agentes com uso de imagens reais de satélite para gestão dos recursos ambientais em médio e longo prazo. A visão geral do modelo de simulação do MASE-BDI é apresentada na Figura 4.9. O MASE-BDI viabiliza a predição de cenários futuros em espaços naturais, para validação de políticas públicas [7]. O MASE-BDI implementou o modelo de racionalidade BDI para melhor representar o raciocínio humano nos agentes da simulação, e garantir que seus agentes tenham tomadas de decisões mais próximas de um humano, uma vez que é dado aos agentes desejos, crenças e intenções. Nele é implementada uma arquitetura de agente com hierarquia, além de parâmetros de entrada ajustáveis fornecidos pelos usuários, como exemplo a quantidade de agentes transformadores individuais e grupais (e.g., cooperativas).

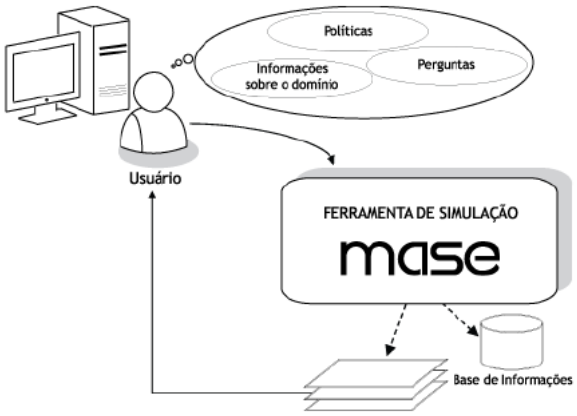

Figura 4.9: Visão geral da simulação MASE-BDI [7].

Na Figura 4.10 é apresentada a arquitetura do MASE-BDI, dividida em três camadas. De cima para baixo, encontra-se a camada de interface com o usuário, a qual inclui o módulo de interface gráfica e o centro de controle do JADEX. A camada de utilidades

que inclui o módulo processamento de imagens usado para o tratamento das imagens de entrada, o módulo para ajustes de parâmetros de entrada e o módulo de validação, responsável por analisar os resultados das simulações.

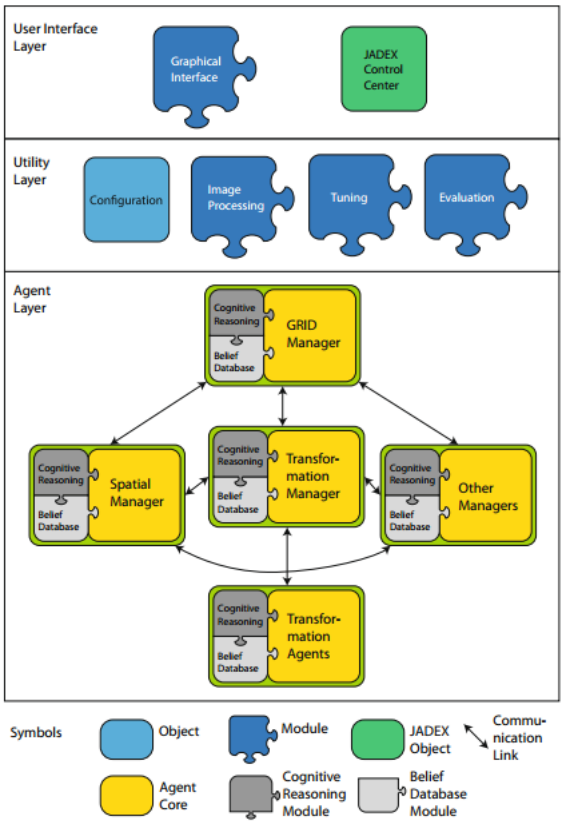

Figura 4.10: Arquitetura MASE-BDI [7].

Na camada principal da arquitetura do MASE-BDI estão os cinco tipos de agentes: *GRID Manager* (GRIDM), *Spatial Manager* (SPM), *Transformation Manager* (TRM), *Transformation Agent* (TRA) e *Other Managers*. GRIDM é responsável por coordenação das simulações, tem a função de instanciar, gerentes, coordenar os passos das simulações e terminar ou reiniciar a simulação. TRM é responsável por instanciar e resolver conflitos entre TRA. SPM é o agente responsável por administrar os recursos espaciais, além de direcionar o TRM para suas localizações inicias e dar diretrizes ao TRM para resolução de conflitos. TRA são os agentes transformadores, sendo estes eles que agem sobre o *grid* de simulação, modificando-a de acordo com regras pré-definidas de exploração. *Other Managers* é uma classe de gerente genérico definido para futuras integrações com o simulador MASE-BDI.

A quantidade de agentes da classe agentes transformadores é um parâmetro de entrada significativo para o resultado de uma simulação. Esse parâmetro é importante para o tempo de execução da simulação, uma vez que quanto maior a quantidade de agentes transformadores maior é o tempo para executar a simulação, além de melhorar a Figura de Mérito, do resultado da simulação, como apresentado na Tabela 4.6. A Figura de Mérito utilizada no MASE-BDI foi proposta por [72], pois apresenta uma forma qualitativa de se avaliar simulações em ambientes naturais, para comparação futura com MASE-BDI.

Tabela 4.6: Resultados simulação MASE-BDI [7].

Quantidade de Agentes Transformadores de cada tipo	Figura de Mérito	Tempo de Simulação
10	14.87036418	0:00:14
20	29.79210311	0:00:27
30	54.04066592	0:00:43
40	51.98603923	0:01:04
50	52.27247897	0:01:50
60	51.75358084	0:02:22
70	50.87789301	0:04:21

Tendo em vista que agentes são entidades computacionais, a quantidade de agentes ativos tem influência na utilização dos recursos da plataforma computacional em que o MASE-BDI está sendo executado. Essa influência sobre o tempo e uso de recursos torna a quantidade de agente de transformação individual e de grupo os mais importantes parâmetros para o modelo atual.

Pelas propriedades citas, o MASE-BDI é utilizado como estudo de caso (aplicação submetida) para validar o modelo proposto nesse trabalho de mestrado. O parâmetro significativo é a quantidade de agentes transformadores.

4.5 Experimentos Iniciais

Modelar o comportamento da aplicação submetida é a principal tarefa ao utilizar o modelo proposto. Essa modelagem é feita monitorando-se os recursos e o comportamento da aplicação. Com os resultados obtidos por meio do monitoramento também foi possível popular a base de dados para garantir o provisionamento dinâmico e modelagem das regras de inferência.

Dadas as características do MASE-BDI apresentadas na Seção 4.4, foram realizados experimentos iniciais a fim de modelar o comportamento em MVs com diferentes configurações. Para análise dos experimentos é importante salientar que o MASE-BDI é uma aplicação com abordagem multiagentes, ou seja, cada agente é uma entidade computacional que realiza tarefas e consome recursos para alcançar seus objetivos. Dessa forma, aumentar a quantidade de agentes significa aumentar a quantidade de entidades consumindo recursos. Para o MASE-BDI aumentar a quantidade de agente significa também aumentar a quantidade de conflitos de interesse entre os agentes. Esses conflitos elevam o tempo da execução, além de interferir na utilização dos recursos por parte dos agentes.

Para realizar os experimentos foi utilizado o provedor *Google Cloud Platform*, uma plataforma de nuvem pública. É importante ressaltar que os testes foram distribuídos em cinco MVs distintas, personalizadas com as configurações:

1. 1 vCPU, 0.9 GB de memória, 60 GB de disco rígido, sistema operacional Ubuntu 14.04 x64.

2. 2 vCPU, 1.9 GB de memória, 60 GB de disco rígido, sistema operacional Ubuntu 14.04 x64.
3. 4 vCPU, 3.6 GB de memória, 60 GB de disco rígido, sistema operacional Ubuntu 14.04 x64.
4. 6 vCPU, 5.4 GB de memória, 60 GB de disco rígido, sistema operacional Ubuntu 14.04 x64.
5. 8 vCPU, 7.2 GB de memória, 60 GB de disco rígido, sistema operacional Ubuntu 14.04 x64.

Nos testes, o parâmetro significativo do MASE-BDI variou entre 1 e 100 agentes transformadores. Com isso, foram realizadas 100 execuções do MASE-BDI para cada configuração de MV.

A Figura 4.11 apresenta o comportamento do MASE-BDI nas cinco MVs utilizadas nos experimentos. Nos gráficos são aprestados o crescimento na quantidade de agentes (de 1 a 100 agentes), o tempo de execução (segundos) e a média de uso de CPU. O foco dos experimentos é capturar o comportamento aplicação MASE-BDI na nuvem, ou seja, não faz parte analisar os motivos internos que levam a aplicação a ter certos comportamentos, essa análise estará fora do escopo desse projeto.

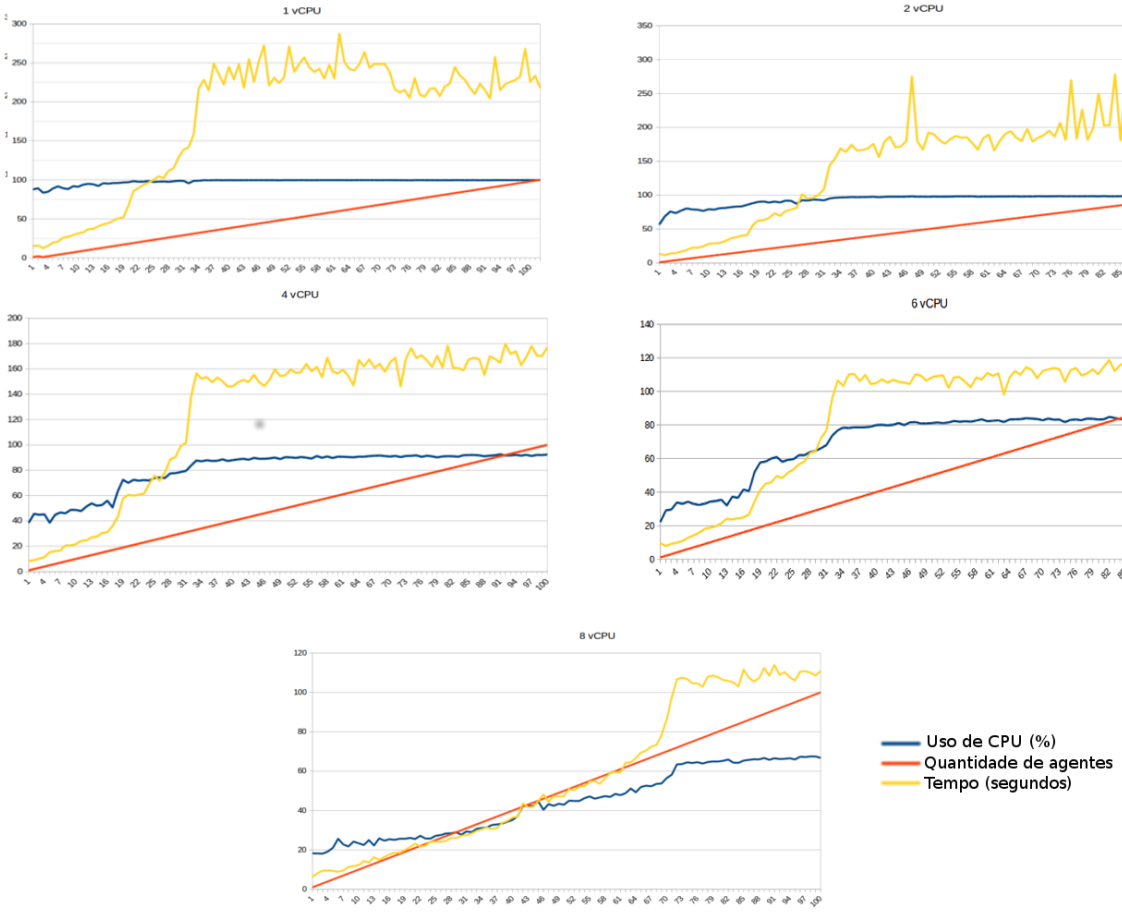

Figura 4.11: Execuções do MASE-BDI em diferentes MVs.

Nas MVs com menos CPUs (1 e 2 vCPUs) é notado que a média de uso da CPU (*CPU Used* - linhas azuis) é muito próxima de 100%, ou seja, o uso da CPU é próximo do total alocado. A medida em que a quantidade de CPUs aumenta (4, 6 e 8 vCPUs) a média de uso da CPU cai, aumentando o desperdício de recursos. Quando feita a comparação entre as MVs com 1 e 2 vCPUs e MVs com 4 e 6 vCPUs tem-se um ganho médio 68.63% no tempo, entretanto nas MVs com menos vCPUs tem a média de uso até 80.41% a mais.

O comportamento do MASE-BDI em uma MV com 8 vCPUs difere das VMs com outras quantidades de CPUs. É notado que para 8 vCPUs o tempo acompanha o crescimento da quantidade de agentes, porém também aumenta o desperdício médio de uso da CPU. Na MV com 8 vCPUs, até 70 agentes o uso médio da CPU fica abaixo de 60% do que foi alocado pelo usuário.

No Capítulo 5 serão apresentados os resultados dos experimentos realizados aplicando as regras de inferência modelada pelos experimentos iniciais discutidos nessa seção.

Capítulo 5

Experimentos e Resultados

Neste capítulo são apresentados os experimentos e resultados obtidos com a realização desse trabalho. O capítulo está divido em duas seções: na Seção 5.1 é apresentado o método estatístico RLM; e a Seção 5.2 apresenta o resultado das escolhas feitas pelos agentes.

5.1 Predição de Recursos

A base de dados formada pelos experimentos da Seção 4.5, permitiu modelar os coeficientes da RLM, o que possibilitou a predição do tempo e uso médio de CPU.

5.1.1 Resultados da Regressão Linear Múltipla

A fim de lapidar e aumentar a relação entre as variáveis do modelo de RLM apresentado na Seção 4.2.1, foi feita a verificação da possibilidade de transformações nas variáveis utilizadas. No modelo de RLM proposto foram feitas transformações em escala logarítmica nas variáveis na predição de tempo expresso no Equação 5.1, essa transformação elevou a relação entre as variáveis e melhorando os resultados do modelo (vide Equação 4.1 e 4.2, onde está definido PS e QC). A transformação não ocorre na RLM do uso de CPU, uma vez que o R^2 apresentado na Tabela 5.1 e a distribuição trazem melhores resultados sem as transformações, como é apresentado nos gráficos das Figuras 5.1 e 5.2. O R^2 é o coeficiente que indica o quanto o modelo consegue explicar os valores observados, sendo melhor os valores próximos de 100.

Note que a Equação 5.1 é a transformação logarítmica da Equação 4.1. Com a transformação o resultado é dado também em logaritmo, por isso é necessário transformar o valor novamente para base decimal, de forma que as regras do agente GMV possam interpretar esses valores conforme apresentado na Equação 5.2.

$$TempoLog = \log_{10}(T1) + \log_{10}(T2) * \log_{10}(PS) + \log_{10}(T3) * \log_{10}(QC) \tag{5.1}$$

$$TempoEstimado = 10^{TempoLog} \tag{5.2}$$

Tabela 5.1: Comparação de R^2 para transformações de variáveis.

RLM	R^2 **com Transformações Logarítmicas**	R^2 **sem Transformações Logarítmicas**
Tempo	90.43%	72.32%
Uso de CPU	67.37%	73.78%

Nas Figuras 5.1 e 5.2 são apresentados os gráficos das variâncias das observações com e sem a transformação logarítmica. É possível observar maior variância das observações analisadas quando comparado a Figura 5.1(a) em relação à Figura 5.1(b) e a 5.2(a) em relação à Figura 5.2(b). Tais transformações matem as propriedades estatísticas do modelo.

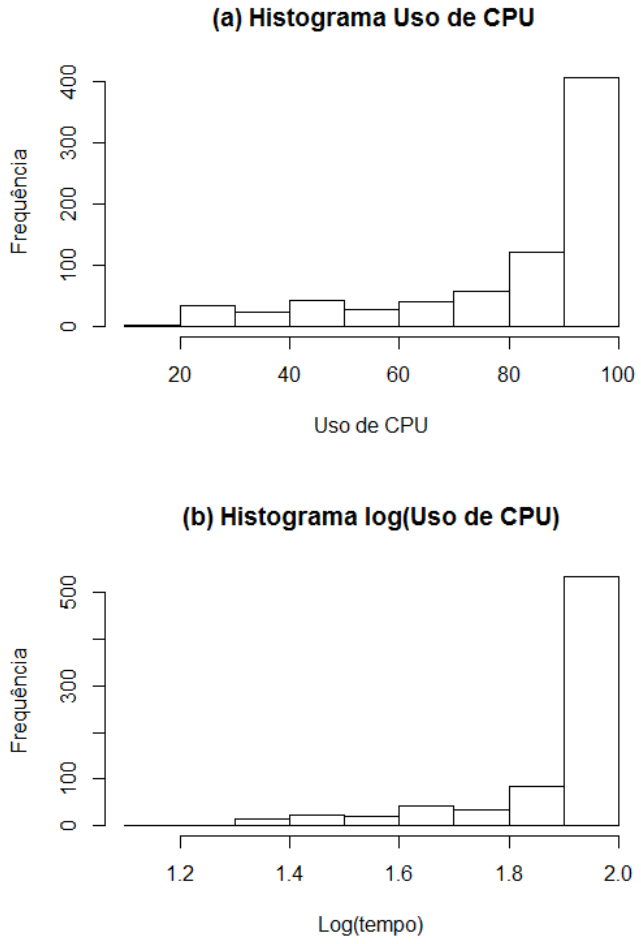

Figura 5.1: Distribuição dos dados da média de uso do CPU.

Para validar e verificar a acurácia da regressão foram feitas as análises de resíduos. Nas Figuras 5.3 e 5.4 são apresentadas técnicas de análise do erro, entre os valores reais e preditos. Nos primeiros gráficos das Figuras 5.3 e 5.4, (a) mostram as não relações entre as observações, de forma que estão distribuídos aleatoriamente. Nos segundos gráficos (b) são mostrados que as observações estão distribuídas sobre a linha da distribuição

normal, com pequenas variações no início e fim das caudas, demonstrando assim que os resíduos estão distribuídos normalmente. Os últimos três gráficos mostram que os resíduos distribuídos aleatoriamente sobre a linha (vermelha) do eixo X, indicando que os resíduos estão distribuídos aleatoriamente entre as variáveis de quantidade de agentes e quantidade de CPU. Ficando validado o modelo de RLM, uma vez que os erros são normais e distribuídos independentemente com média zero.

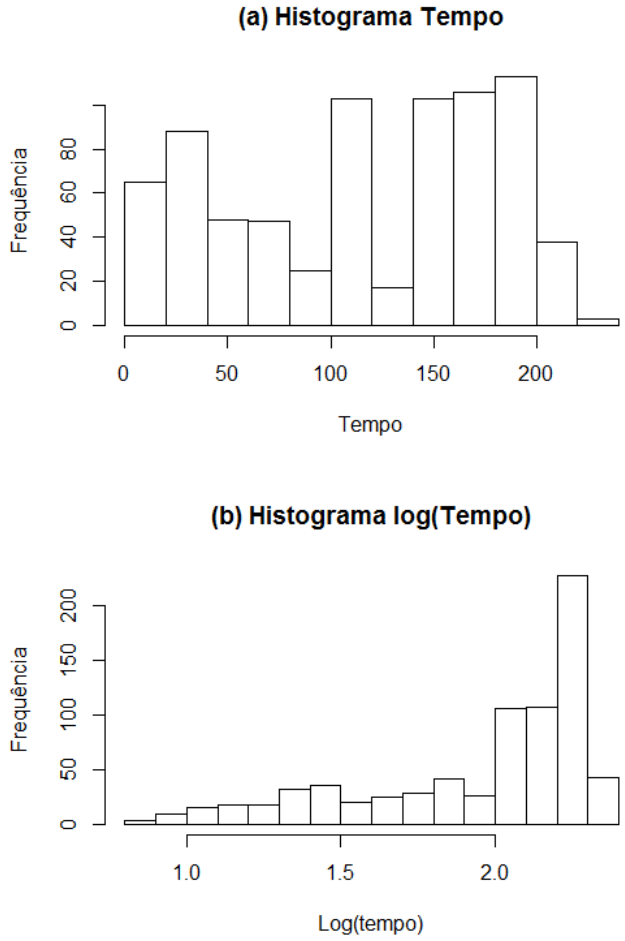

Figura 5.2: Distribuição dos dados da média do tempo.

Os resultados da regressão estão expressos na Tabela 5.2. Note que para o uso do CPU tem-se um MAPE de 3.59%, ou seja, tem 96.41% de acertos na predição do uso de CPU. Para o modelo predizer o tempo gasto para executar o MASE-BDI em média, a previsão está incorreta em 5.28%, com 94.72% de acerto.

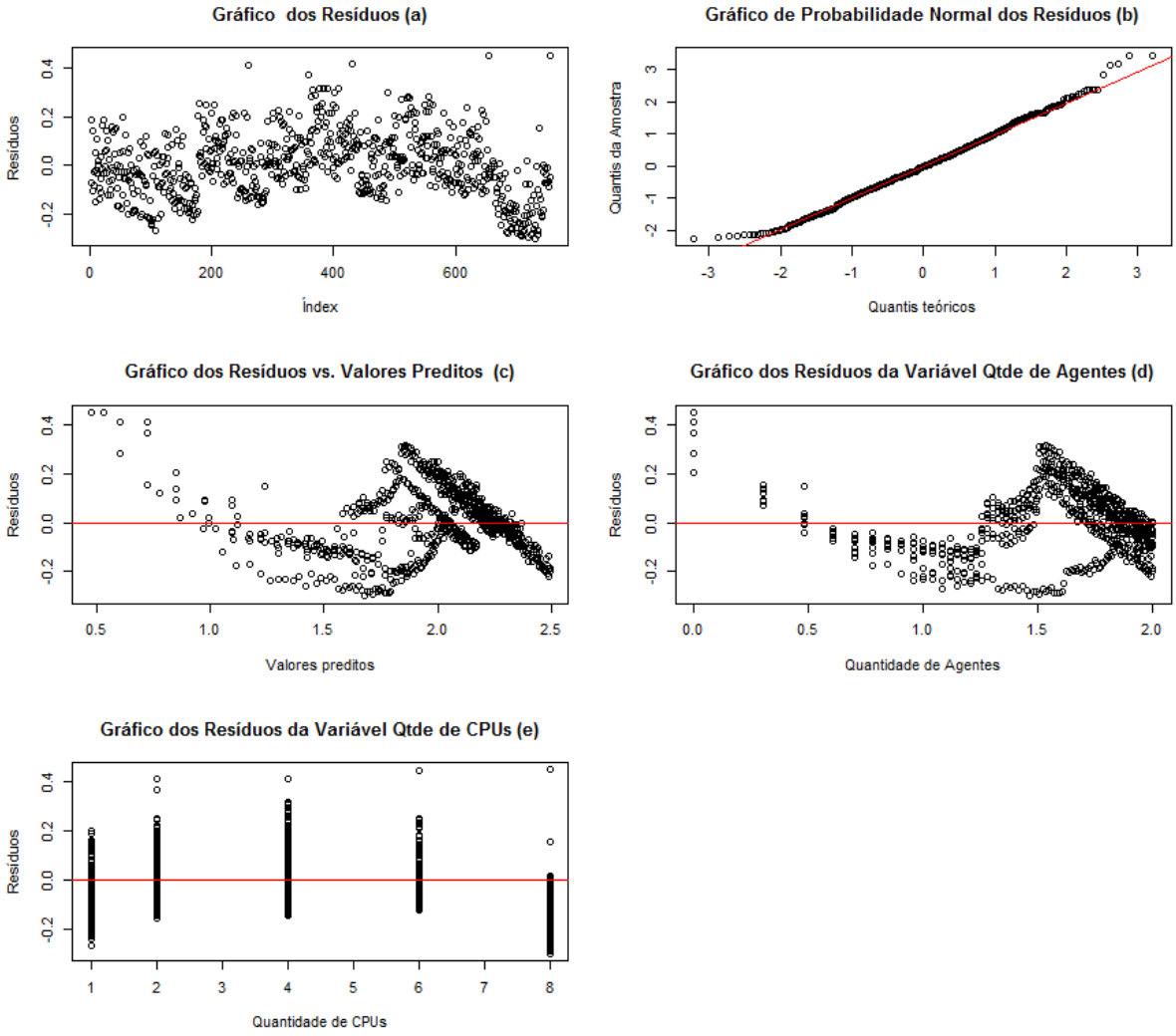

Figura 5.3: Gráficos dos resíduos.

Com os resultados anteriormente apresentados se pode concluir que o modelo de RLM pode ser utilizado como ferramenta para predizer o tempo e uso do CPU para execuções do MASE-BDI.

5.1.2 Estimativas de Tempo e Uso de CPU

Nessa seção são apresentados os resultados obtidos por meio da utilização do modelo de RLM para predizer tempo e uso de CPU, com o dados das execuções do MASE-BDI. Seção 4.5 apresenta as distribuições das MVs e cenários para realização dos experimentos iniciais.

Nos gráficos das Figuras 5.5 e 5.6 são apresentadas as comparações entre os dados reais (azul) e os dados estimados pela regressão (vermelho). Na Figura 5.5 são mostrados os resultados da RLM para o uso de CPU nas cinco configurações de MVS. Dessa forma, têm-se uma aproximação com erro médio de 22 segundos para a predição de tempo e 8.52% na estimativa de uso de CPU.

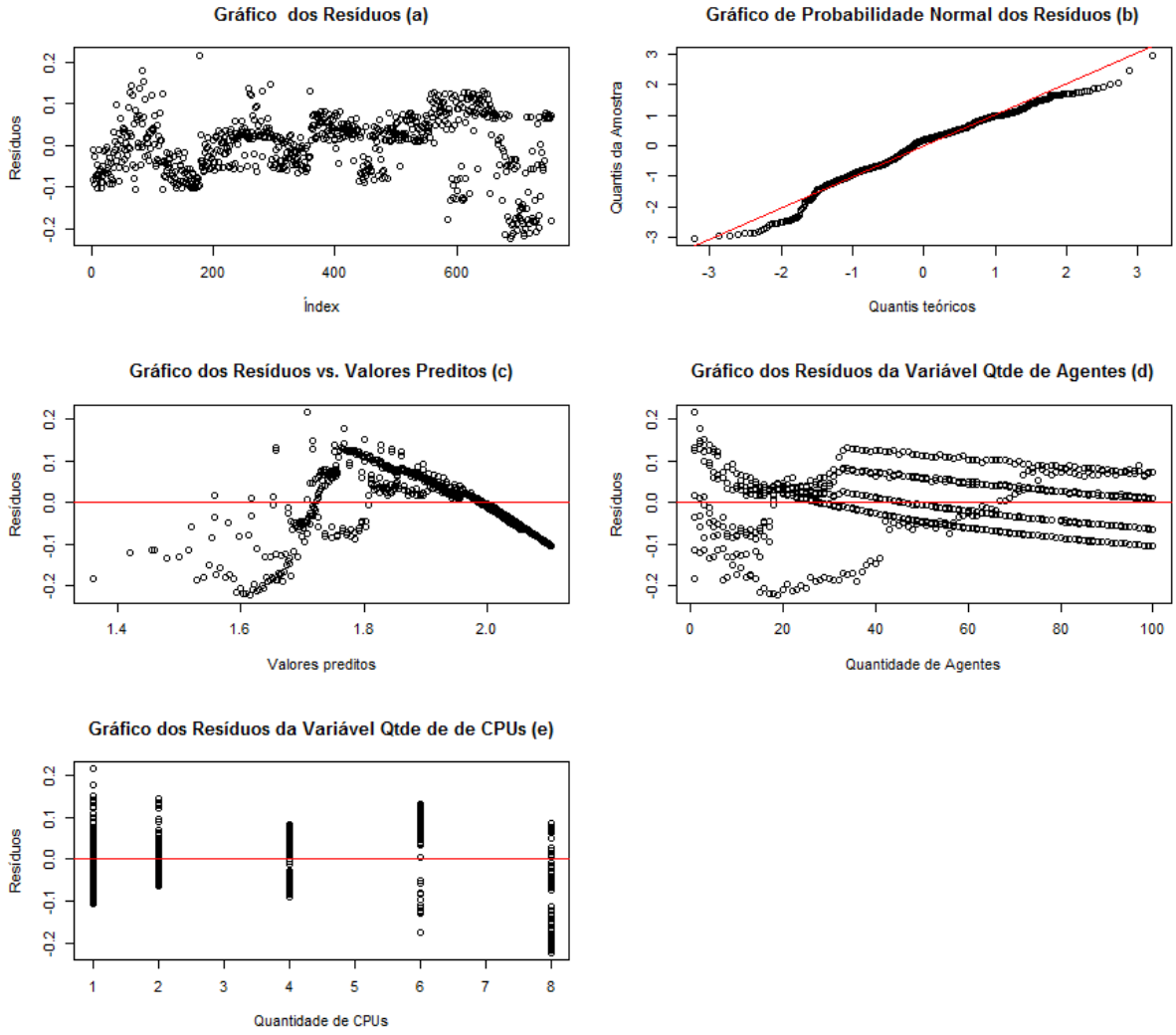

Figura 5.4: Gráficos dos resíduos.

O Comportamento dinâmico da aplicação MASE-BDI pode ser notado no tempo e do uso do CPU, mesmo que os valores sigam um padrão de crescimento existe uma grande variação entre uma execuções.

Nos gráficos de comparações nota-se que a RLM consegue expressar o padrão de crescimento das MVs. Nesse sentido, para os resultados da RLM o erro é aceitável, apresentando os valores da RLM acima de 90% de acerto (vide MAPE com uso de CPU 96.41% e tempo 94.72%). Ou seja, com o uso dos métodos apresentados na da Tabela 5.3 está sendo avaliado o resultado do modelo de RLM para predizer o tempo e o uso médio de CPU.

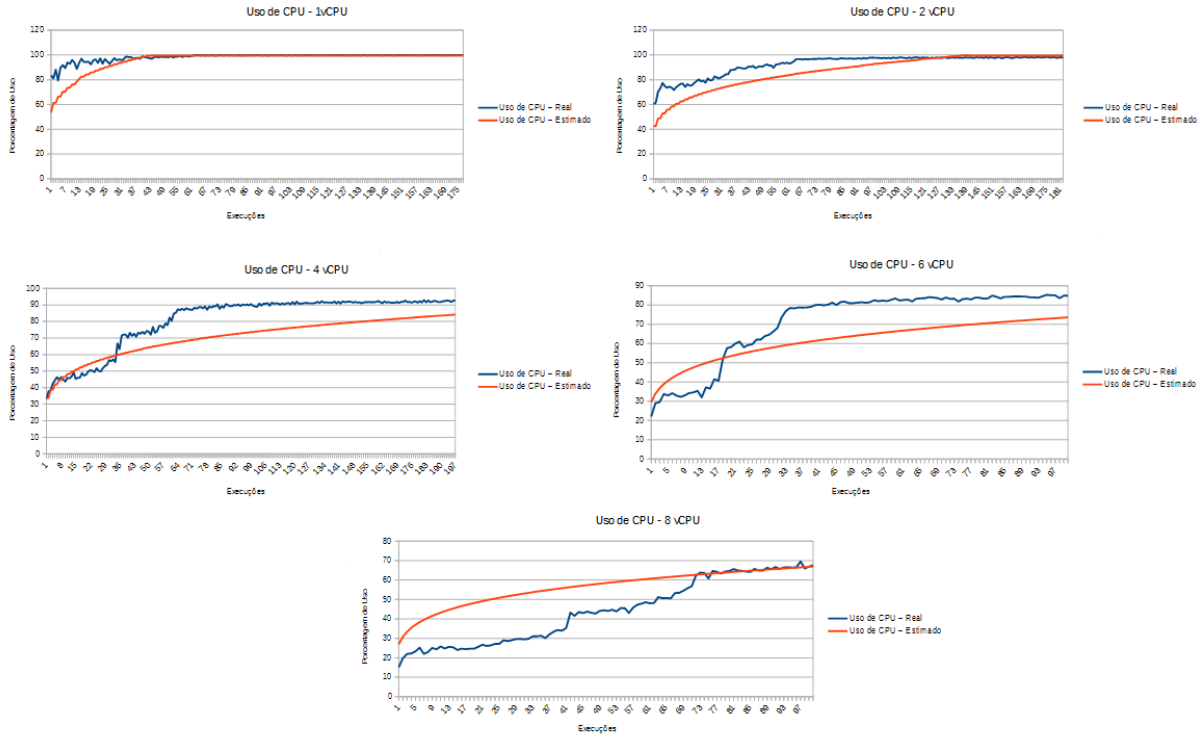

Figura 5.5: Gráficos da regressão do uso de CPU.

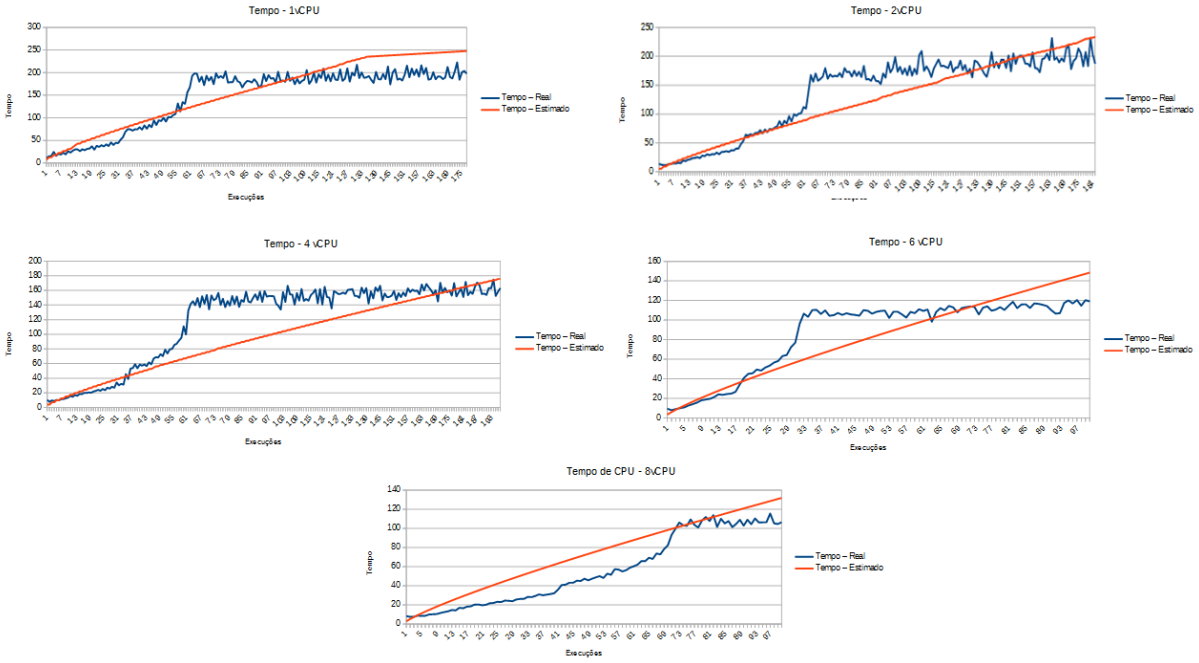

Figura 5.6: Gráficos da regressão do tempo.

Tabela 5.2: Resultado dos índices de avaliação.

Método de Avaliações	**Tempo**	**Uso de CPU**
Bias	-1.872841e-15	-2.678927e-15
MAE	0.09	0.06
RMSE	0.11	0.08
MAPE	5.28%	3.59%

5.2 Provisionamento Dinâmico

Com os resultados da regressão, o agente GMV utiliza-se das regras de inferência apresentadas na Seção 4.3 e escolhe a melhor configuração da MV, para executar no menor tempo, com menor desperdício de CPU e com melhor custo benefício. Igualmente com os parâmetros de teste apresentados na Seção 4.5, foram feitas 100 execuções do MASE-BDI com a quantidade de agentes transformadores variando de 1 a 100 (podendo variar o número de agentes em execuções futuras).

O GMV escolhe entre as cinco configurações de máquinas apresentadas na Seção 4.5, essas MVs são disponibilizadas de forma gratuita no Google Cloud, o que limitou as configurações dos experimentos. Os resultados das escolhas GMV são expressos e apresentados em cinco cenários que propiciam a comparação e validação do modelo, sendo estes divididos conforme as decisões do GMV:

- Cenário 1: intervalo entre 1 e 30 agentes, em que o GMV realizou as execuções na MV com 1 vCPU;
- Cenário 2: intervalo entre 31 e 46 agentes, em que o GMV realizou as execuções na MV com 2 vCPU;
- Cenário 3: intervalo entre 47 e 70 agentes, em que o GMV realizou as execuções na MV com 4 vCPU;
- Cenário 4: intervalo entre 71 e 90 agentes, em que o GMV realizou as execuções na MV com 6 vCPU; e
- Cenário 5: intervalo entre 91 e 100 agentes, em que o GMV realizou as execuções na MV com 8 vCPU.

Na Tabela 5.3 são apresentados os resultados para a execução do modelo nos cinco cenários, estando os valores das execuções selecionadas pelo GMV destacados em vermelho. Em cada linha da tabela estão os resultados das execuções do MASE-BDI, a média de uso de CPU, a média de tempo e a média dos custos para cada cenário em cada MV. É possível comparar o resultado da escolha a partir do GMV com execuções de outras MVs. Por exemplo, na primeira linha (Cenário 1, intervalo entre 1 e 30 agentes), o GMV escolheu a MV com 1 vCPU, uma vez que apresentou o maior uso de CPU e o tempo e custos aceitáveis, quando comparado às demais MVs.

Importante ressaltar que o GMV não leva apenas um dos quesitos em consideração no momento da escolha. Nas regras de inferência do GMV, primeiramente, são selecionadas as MVs com maiores médias de uso de CPU e que tenham os menores tempos de execução, criando assim uma lista de MVs candidatas. Dentre as MVs candidatas se faz escolha da

MV com o menor custo. As colunas tempo (dado em segundos) e custo (em dolar) apresentadas na Tabela 5.3 levam em consideração a adição do *overhead* do tempo, e da criação e exclusão de cada MV (*slave*). O custo da MV (*master*) em que é executado o agente GMV também é adicionado aos custos.

Tabela 5.3: Resultados de execuções das escolhas do GMV.

Intervalo de Agentes	1 vCPU			2 vCPU			4 vCPU			6 vCPU			8 vCPU		
	Uso de CPU(%)	Tempo (s)	Custo ($)	Uso de CPU(%)	Tempo (s)	Custo ($)	Uso de CPU(%)	Tempo (s)	Custo ($)	Uso de CPU(%)	Tempo (s)	Custo ($)	Uso de CPU(%)	Tempo (s)	Custo ($)
1-30	**96,96**	**61,53**	**0,61**	89,61	46,77	0,91	60,41	38,24	1,48	46,16	32,18	1,87	25,08	17,22	1,34
31-46	99,49	180,35	1,73	**96,04**	**138,03**	**2,68**	86,70	122,75	5,54	76,69	102,64	5,97	31,31	39,52	3,07
47-70	99,63	189,34	1,93	97,35	184,91	3,59	**80,13**	**104,31**	**4,05**	81,45	108,08	6,29	44,35	56,98	4,42
71-90	99,69	213,68	2,09	97,76	186,47	3,62	91,36	153,35	5,95	**78,73**	**78,71**	**4,58**	63,06	101,33	7,86
91-100	99,74	224,06	2,17	97,93	222,07	4,31	91,85	158,34	6,14	84,45	115,27	6,71	**76,09**	**79,06**	**6,14**

Por meio da Tabela 5.3 é possível perceber que o uso médio de CPU ficou entre 76.09% e 96.96% do que foi alocado (vide valores ressaltados em vermelho na primeira coluna – uso de CPU). Os resultados quando comparados com os valores dos gráficos apresentados na Figura 4.11 estão em bons, ressaltando o balanceado entre tempo e custo.

Comparando as mesmas 100 execuções feitas em VMs com configurações fixas, o modelo proposto encontrou um ponto de equilíbrio entre o menor tempo e o menor custo. Verifique que para cada execução do MASE-BDI o agente GMV selecionou qual a melhor configuração da MV (*slave*) para aquela execução, ficando pequena a diferença entre o tempo e o custo, quando comparados com as outras colunas.

5.2.1 Resultados da Elasticidade

Foram realizados experimentos do módulo de elasticidade, sendo utilizada uma base de dados com menos casos de execução, neste caso, não levou a uma boa escolha de MV. Em decorrência, houve desperdício de recursos alocados, acionando a regra de elasticidade (Figura 4.8) descrita na Sessão 4.2.1, então foi realocado uma nova MV para aquela execução.

Os testes da regra de elasticidade levam em consideração o *overhead* para criação de uma nova MV, como apresentado na Sessão 4.3.2. A Elasticidade é um módulo que idealmente não será utilizado, uma vez que existe o módulo de predição de recursos. Esse módulo é uma forma de suprir a falta de uma base bem populada para ajustes das regras de predição.

Como citado na Seção 4.1, o modelo recebe como parâmetros de entrada quantidade de agentes, quantidade de execução e o incremento na quantidade de agentes para cada execução. Dessa forma, é feito um ponto de recuperação a cada início de uma execuções do MASE-BDI, com isso a elasticidade ocorre em dados momentos dentro de um conjunto de

execuções, os resultados dessas execuções são salvos para serem levados em consideração nas próximas execuções.

Tabela 5.4: Resultado dos Experimentos com Elasticidade.

Ocorrências	Quantidade de Agentes	vCPU	Média de Uso de CPU	% Concluido	Nova vCPU	Nova Média de Uso de CPU	Diferença de Uso
1	10	4	47,3333333333333	55	1	94,5357142857143	47.202
2	16	6	41,48	52	1	95,0263157894737	53.546
3	17	6	40,6296296296296	42	1	95,7	55.070
4	22	8	25,1904761904762	12	1	98,0945945945946	72.904
5	29	8	28,32	15	2	92,91	64.59
6	33	8	31,2142857142857	37	2	96,577380952381	65.363
7	43	8	42,2558139534884	52	4	87,6376811594203	45.381
8	44	8	43,0238095238095	48	4	88,3241379310345	45.300

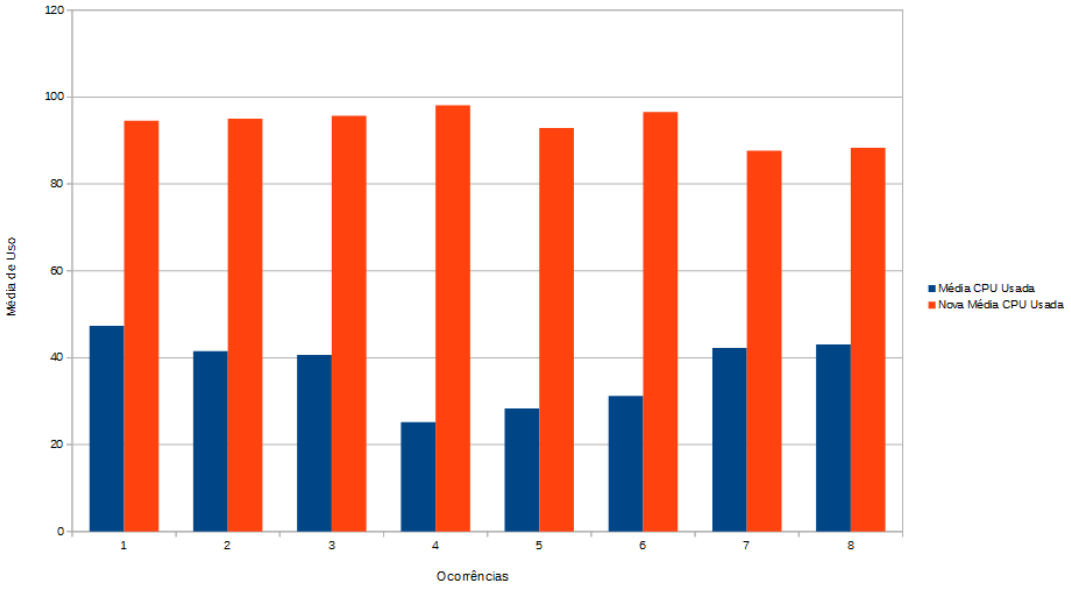

Figura 5.7: Comparação entre o uso de CPU após elasticidade.

Na Tabela 5.4 são apresentadas as ocorrências de elasticidade conforme a diferença de uso na última coluna, a qual expressa a diferença entre a coluna média de uso de CPU e a nova média de uso de CPU. Quando acionado, esse módulo busca sempre diminuir o desperdício médio de uso.Ao serem comparados os resultados (Figura 5.7) se percebe claramente que a escolha da nova MV levou a uma queda no desperdício, justificando assim seu acionamento. Quando realizado a elasticidade, a diferença entre a MV inicial-

mente alocada e a nova MV foi uma diminuição média de 40% de desperdício, conforme apresentado na Tabela 5.4.

5.2.2 Resultados da Recuperação da Aplicação

Para os experimentos no módulo de recuperação da aplicação, erros foram induzidos durante a execução do MASE-BDI. Foram introduzidos erros na MV *slave*, em que está sendo executada a aplicação e erros na VM *master* em que os agentes estão gerenciando as execuções.

A recuperação da aplicação implementada busca garantir que todas as execuções solicitadas pelo usuário sejam finalizadas e concluídas com sucesso. Nos testes, os resultados foram satisfatórios, uma vez que o modelo conseguiu restabelecer normalmente a execução. Assim como na elasticidade, na recuperação dos erros (ver Sessão 4.2.1) o modelo verifica os pontos de recuperação feitos a cada início de uma execução, de modo que as execuções já concluídas não sejam afetadas e que sejam executados apenas as faltantes.

Na Tabela 5.5 são apresentados os erros ocorridos nas execuções e seus métodos de recuperação, também é mostrado o tempo médio de cada etapa assim como uma média total do tempo para recuperação dos tipos de erros. Na recuperação dos erros nas MVs *slaves* o tempo é relativamente maior (aproximadamente 132 segundos, tempo em que o provedor cria e deleta MVs) pelo fato da abordagem da recuperação deletar a MV e criar uma nova. Esse processo eleva o tempo total. Os erros na MV *master* são restaurados utilizando o *script shell*, que monitora se as execuções estão sendo finalizadas com sucesso, o tempo médio de recuperação da falha é de 62 segundos. Note que o tempo total de execução em geral é menor que 132 segundos conforme apresentado na Tabela 5.3, na coluna tempo.

Tabela 5.5: Resultado dos Experimentos da Recuperação da Aplicação.

Local do Erro	Tipo de Erro	Método de Recuperação	1ª Etapa	2ª Etapa	3ª Etapa	Total
MV Slave	Erro na execução do MASE-BDI	Recriar MV slave com as mesmas configurações	8's para deletar a MV slave	42s para recriar a MV slave	5s para iniciar a execução MASE-BDI	132s para total recuperação
	MV slave para de responder	Recriar MV slave com as mesmas configurações				
MV Master	MV master para de responder	Executa o script shell para reiniciar a execução onde ocorreu o erro	15s para reiniciar a execução do modelo	42s para criar MV slave	5s Para iniciar a execução do MASE-BDI	62s para total recuperação
	MV master é reiniciada	Executa o script shell para reiniciar a execução onde ocorreu o erro				

Os testes de recuperação da aplicação tem foco na garantia das execuções do MASE-BDI. É notório que o tempo de recuperação é elevado quando comparado com o tempo de execução do MASE-BDI, porém o foco foi garantir apenas que seja executado a quantidade

de vezes solicitada pelo usuário. Os testes realizados não consideram erros internos da aplicação, uma vez que o atual modelo trata o MASE-BDI como uma caixa preta.

No Capítulo 6 são apresentados os resultados alcançados durante a realização desse trabalho de mestrado. São ainda apresentados alguns trabalhos futuros.

Capítulo 6

Conclusões e Trabalhos Futuros

A presente dissertação de mestrado propôs o desenvolvimento de um modelo computacional com abordagem de SMA, a fim de garantir características de nuvem computacional para aplicações submetidas nesse ambiente. Assim, foi proposto o uso de um modelo de racionalidade baseado em regras de inferência, tendo como estudo de caso a aplicação MASE-BDI. O SMA desenvolvido tem a finalidade de proporcionar ao MASE-BDI o melhor uso dos recursos, diminuindo o tempo de execução e de custos.

Com relação às características mais encontradas em ferramentas de monitoramento, provisionamento dinâmico de recursos e elasticidade o foco dessas ferramentas é encontrar melhorias nas arquiteturas de provedores, além de auxiliá-los no gerenciamento de recursos. O modelo proposto explora justamente a lacuna encontrada, combinando recursos de IaaS e a necessidade de usuários, que desejam executar aplicação em nuvem. O foco é fornecer aos usuários a possibilidade de um uso otimizado de recursos. É importante ressaltar que com base nos estudos efetuados não existe hoje ferramenta de abordagem SMA que garanta que aplicações desenvolvidas sem o foco no ambiente nuvem possam ser submetidas e executadas garantindo monitoramento, provisionamento dinâmico e elasticidade. Foi então respondida a questão de pesquisa inicialmente proposta, mostrando que é possível definir um modelo de monitoramento e alocação dinâmica de recursos para aplicações em nuvem computacional utilizando agentes inteligentes.

O resultado do trabalho de pesquisa apresentado tem como principais contribuições: a definição de um modelo, o detalhamento arquitetural, a implementação do protótipo e, a realização de experimentos com o estudo de caso MASE-BDI. O modelo de predição com RLM conseguiu 96.41% de acerto na predição do uso de CPU e 94.72% na predição do tempo de execução. As escolhas do agente GMV também apresentaram bons resultados, uma vez que quando comparadas às execuções feitas em outras MVs, as escolhas encontraram um equilíbrio entre o uso de CPU, tempo e custo, garantindo que o uso médio de CPU ficasse acima de 76%. O módulo de elasticidade conseguiu diminuir em média 40% o desperdício para execução do modelo em MVs preditas erroneamente. Os resultados preliminares mostram que a arquitetura do modelo implementado é robusta o suficiente para executar uma aplicação dinâmica com comportamento não determinístico e em ambiente de nuvem com tempo e custo reduzido.

Algumas melhorias podem ser vislumbradas para complementar o modelo aqui apresentado, principalmente ao modelo de racionalidade dos agentes, distribuição da aplicação e no módulo de predição. Considera-se que os trabalhos futuros possam ser estabelecidos

nos seguintes pontos:

- Desempenho - implementar a arquitetura de maneira distribuída, buscando um melhor desempenho do modelo. Essa alteração na arquitetura possibilitará que o gerenciamento de recursos e a execuções sejam feitos de forma robusta e escalável;
- Racionalidade dos agentes - implementar na racionalidade dos agentes, modelos ainda mais dinâmicos, de forma que possam aprender e ajustar as regras do motor de inferência de forma automática. Uma outra abordagem para racionalidade dos agentes seria a implantação de um modelo BDI, em que as tomadas de decisões seriam baseadas em crenças, desejos e intenções para cada ação;
- Interface - para aumentar a acessibilidade e garantir que outros usuários possam utilizar o modelo é necessário uma interface amigável e acessível, garantindo que usuários que não tenham conhecimentos técnicos em desenvolvimento possam utilizá-la;
- Predição de Recursos - com o aumento do uso do modelo e com uma base de dados massiva, a incorporação de outros modelos de aprendizado de máquina como MVS e RN também trará benefícios ao modelo.

Referências

[1] Domenico Talia. Cloud computing and software agents: Towards cloud intelligent services. In *WOA*, volume 11, pages 2–6. Citeseer, 2011. 1

[2] Fernando De la Prieta, Sara Rodríguez, Javier Bajo, and Juan M Corchado. +cloud: A virtual organization of multiagent system for resource allocation into a cloud computing environment. In *Transactions on Computational Collective Intelligence XV*, pages 164–181. Springer, 2014. 1, 35, 36

[3] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. pages 1–10, 2008. 1, 22

[4] Peter Mell and Tim Grance. The nist definition of cloud computing. *National Institute of Standards and Technology*, 53(6):50, 2009. 1, 22, 23, 25, 32

[5] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010. 1

[6] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009. 2, 5, 6, 15

[7] Cássio G. C. Coelho, Carolina G. Abreu, Rafael M. Ramos, Aldo H. D. Mendes, George Teodoro, and Célia G. Ralha. Mase-bdi: agent-based simulator for environmental land change with efficient and parallel auto-tuning. *Applied Intelligence*, 45(3):904–922, 2016. xiv, xv, 3, 54, 55, 56

[8] Célia G. Ralha, Carolina G. Abreu, Cássio G.C. Coelho, Alexandre Zaghetto, Bruno Macchiavello, and Ricardo B. Machado. A multi-agent model system for land-use change simulation. *Environmental Modelling Software*, 42:30 – 46, 2013. 3

[9] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall series in artificial intelligence. Prentice Hall, 2010. xiv, 5, 6, 7, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 28, 29

[10] Stefan Bussmann, Nicolas R Jennings, and Michael J Wooldridge. *Multiagent systems for manufacturing control: A design methodology*. Springer Science & Business Media, 2004. 5

[11] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons, 2007. 8, 14

[12] James Mayfield, Yannis Labrou, and Tim Finin. Evaluation of kqml as an agent communication language. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 347–360. Springer, 1995. 8

[13] John R Searle. *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press, 1969. 8

[14] Katia P. Sycara. The many faces of agents. *AI magazine*, 19(2):11–12, 1998. 10

[15] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005. 13

[16] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade: a fipa2000 compliant agent development environment. In *Proceedings of the fifth international conference on Autonomous agents*, pages 216–217. ACM, 2001. 14

[17] Craig Boutilier. Sequential optimality and coordination in multiagent systems. In *IJCAI*, volume 99, pages 478–485, 1999. 14

[18] Nicholas R Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38, 1998. 14

[19] Katia P Sycara. Multiagent systems. *AI magazine*, 19(2):79, 1998. 14

[20] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A bdi reasoning engine. In *Multi-agent programming*, pages 149–174. Springer, 2005. 15

[21] Michael Bratman. *Intention, Plans, and Practical Reason*. Center for the Study of Language and Information, 1987. 15

[22] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming multi-agent systems in AgentSpeak using Jason*, volume 8. John Wiley & Sons, 2007. 16

[23] Hyacinth S Nwana, Divine T Ndumu, Lyndon C Lee, and Jaron C Collis. Zeus: a toolkit for building distributed multiagent systems. *Applied Artificial Intelligence*, 13(1-2):129–185, 1999. 16

[24] Ronald J Brachman and Brian C Smith. Special issue on knowledge representation. *ACM SIGART Bulletin*, (70):1–138, 1980. 18

[25] Robert Kowalski. Predicate logic as programming language. In *IFIP congress*, volume 74, pages 569–544, 1974. 19

[26] Robert Kowalski. *Computational logic and human thinking: how to be artificially intelligent*. Cambridge University Press, 2011. 19

[27] E Friedman Hill. Jess in action: rule-based systems in java, 2003. 20

[28] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005. 20

[29] Paul Browne. *JBoss Drools business rules*. Packt Publishing Ltd, 2009. 21, 43

[30] Robert B Doorenbos. *Production matching for large learning systems*. PhD thesis, University of Southern California, 1995. 21

[31] Don Batory. The leaps algorithm. Technical report, Austin, TX, USA, 1994. 21

[32] Ernest Friedman. *Jess in action: rule-based systems in java*. Manning Publications Co., 2003. 21

[33] Carlos Santos da Figueira Filho and Geber Lisboa Ramalho. Jeops—the java embedded object production system. In *Advances in Artificial Intelligence*, pages 53–62. Springer, 2000. 21

[34] William Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003. 21

[35] Dennis Merritt. *Building expert systems in Prolog*. Springer Science & Business Media, 2012. 21

[36] Robert Kowalski and Donald Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3-4):227–260, 1971. 21

[37] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.*, 25(6):599–616, jun. 2009. 22

[38] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. 22, 27

[39] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008. xiv, 22, 23, 24, 25, 33

[40] Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr., Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens. {SLA} guarantees for cloud services. *Future Generation Computer Systems*, 54:233 – 246, 2016. 22

[41] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008. 23

[42] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009. 23

[43] Khalid Alhamazani, Rajiv Ranjan, Karan Mitra, Fethi Rabhi, Prem Prakash Jayaraman, Samee Ullah Khan, Adnene Guabtni, and Vasudha Bhatnagar. An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art. *Spring Computing*, 97(4):357–377, 2015. 26

[44] Eddy Caron, Luis Rodero-Merino, Frédéric Desprez, and Adrian Muresan. Auto-Scaling, Load Balancing and Monitoring in Commercial and Open-Source Clouds. Research Report RR-7857, INRIA, February 2012. 26

[45] Afef Mdhaffar, Riadh Ben Halima, Mohamed Jmaiel, and Bernd Freisleben. Cep4cloud: Complex event processing for self-healing clouds. In *2014 IEEE 23rd International WETICE Conference*, pages 62–67. IEEE, 2014. 26

[46] Mahmoud Al-Ayyoub, Yaser Jararweh, Mustafa Daraghmeh, and Qutaibah Althebyan. Multi-agent based dynamic resource provisioning and monitoring for cloud computing systems infrastructure. *Cluster Computing*, 18(2):919–932, 2015. 27, 37, 38

[47] Jacques Sauvé, Filipe Marques, Antão Moura, Marcus Sampaio, João Jornada, and Eduardo Radziuk. Sla design from a business perspective. In *International Workshop on Distributed Systems: Operations and Management*, pages 72–83. Springer, 2005. 27

[48] Michael Factor, Kalman Meth, Dalit Naor, Ohad Rodeh, and Julian Satran. Object storage: The future building block for storage systems. In *Local to Global Data Interoperability-Challenges and Technologies, 2005*, pages 119–123. IEEE, 2005. 27

[49] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012. 28, 30

[50] R Buyya, D Abramson, and J Nimrod-G Giddy. An architecture for a resource management and scheduling system in a global computational grid ii proc. of the 4th international conference and exhibition on high performance computing in asia-pacific region. *Beijing, China*, 2000. 28

[51] Daniel de Oliveira, Vitor Viana, Eduardo Ogasawara, Kary Ocana, and Marta Mattoso. Dimensioning the virtual cluster for parallel scientific workflows in clouds. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 5–12. ACM, 2013. 28

[52] Sunirmal Khatua, Moumita Mitra Manna, and Nandini Mukherjee. Prediction-based instant resource provisioning for cloud applications. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 597–602. IEEE, 2014. 28

[53] Tom M Mitchell et al. Machine learning. wcb, 1997. 28

[54] Palden Lama and Xiaobo Zhou. Aroma: Automated resource allocation and configuration of mapreduce environment in the cloud. In *Proceedings of the 9th international conference on Autonomic computing*, pages 63–72. ACM, 2012. xiv, 28, 29

[55] Alex M Andrew. An introduction to support vector machines and other kernel-based learning methods. *Kybernetes*, 2013. 28

[56] Ronald MacGregor. *Neural and brain modeling*. Elsevier, 2012. 29

[57] Michael A Arbib. *The handbook of brain theory and neural networks*. MIT press, 2003. xiv, 29, 30

[58] Frank Harrell. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer, 2015. xiv, 30

[59] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2015. 30

[60] Abdul Rashid and Shabbir Ahmad. Predicting stock returns volatility: An evaluation of linear vs. nonlinear methods. *International Research Journal of Finance and Economics*, 20:141–150, 2008. 31

[61] Naihua Duan. Smearing estimate: a nonparametric retransformation method. *Journal of the American Statistical Association*, 78(383):605–610, 1983. 31

[62] Peter C Austin, William A Ghali, and Jack V Tu. A comparison of several regression models for analysing cost of cabg surgery. *Statistics in medicine*, 22(17):2799–2815, 2003. 31

[63] T Hashino, AA Bradley, and SS Schwartz. Evaluation of bias-correction methods for ensemble streamflow volume forecasts. *Hydrology and Earth System Sciences Discussions*, 3(2):561–594, 2006. 31

[64] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005. 31, 32

[65] David A Swanson, Jeff Tayman, and TM Bryan. Mape-r: a rescaled measure of accuracy for cross-sectional subnational population forecasts. *Journal of Population Research*, 28(2-3):225–243, 2011. 32

[66] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 85–96. ACM, 2012. 32

[67] Emanuel Ferreira Coutinho, Flávio Rubens de Carvalho Sousa, Paulo Antonio Leal Rego, Danielo Gonçalves Gomes, and José Neuman de Souza. Elasticity in cloud computing: a survey. *annals of telecommunications-annales des télécommunications*, 70(7-8):289–309, 2015. xiv, 33

[68] U. Siddiqui, G. A. Tahir, A. U. Rehman, Z. Ali, R. U. Rasool, and P. Bloodsworth. Elastic jade: Dynamically scalable multi agents using cloud resources. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 167–172. IEEE, Nov 2012. 36, 37, 43

[69] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011. 37

[70] Barry Rountree, David K Lowenthal, Martin Schulz, and Bronis R de Supinski. Practical performance prediction under dynamic voltage frequency scaling. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8. IEEE, 2011. 42

[71] Nawel Bayar, Saber Darmoul, Sonia Hajri-Gabouj, and Henri Pierreval. Fault detection, diagnosis and recovery using artificial immune systems: A review. *Engineering Applications of Artificial Intelligence*, 46:43–57, 2015. 45

[72] Robert Gilmore Pontius, Diana Huffaker, and Kevin Denman. Useful techniques of validation for spatially explicit land-change models. *Ecological Modelling*, 179(4):445–461, 2004. 55