UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

Desenvolvimento e avaliação de algoritmos de otimização para o cumprimento de acordos de níveis de serviços em nuvem

Leonildo José de Melo de Azevedo

Dissertação de Mestrado do Programa de Pós-Graduação em Ciências
de Computação e Matemática Computacional (PPG-CCMC)

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Leonildo José de Melo de Azevedo

Desenvolvimento e avaliação de algoritmos de otimização para o cumprimento de acordos de níveis de serviços em nuvem

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

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Júlio Cezar Estrella

Coorientador: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
Maio de 2018

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

d278d de Melo de Azevedo, Leonildo José
Desenvolvimento e avaliação de algoritmos de
otimização para o cumprimento de acordos de níveis de
serviços em nuvem / Leonildo José de Melo de
Azevedo; orientador Júlio César Estrella;
coorientador Claudio Fabiano Motta Toledo. -- São
Carlos, 2018.
107 p.

Dissertação (Mestrado - Programa de Pós-Graduação
em Ciências de Computação e Matemática
Computacional) -- Instituto de Ciências Matemáticas
e de Computação, Universidade de São Paulo, 2018.

1. Computação em nuvem. 2. otimização. 3.
provisionamento de recursos. 4. metaheurística. I.
Estrella, Júlio César, orient. II. Motta Toledo,
Claudio Fabiano, coorient. III. Título.

Leonildo José de Melo de Azevedo

**Development and evaluation of optimization algorithms for
the fulfillment of cloud service level agreements**

Master dissertation submitted to the Institute of
Mathematics and Computer Sciences – ICMC-USP,
in partial fulfillment of the requirements for the
degree of the Master Program in Computer Science
and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and
Computational Mathematics

Advisor: Prof. Dr. Júlio Cezar Estrella

Co-advisor: Prof. Dr. Claudio Fabiano Motta Toledo

USP – São Carlos
May 2018

*Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.
Em especial, ao pesquisadores do Instituto de Ciências Matemáticas e de Computação (ICMC).*

AGRADECIMENTOS

Como diz um provérbio africano, “se quer ir rápido vá sozinho, se quer ir longe vá em grupo”. Assim foi com este trabalho de mestrado, que certamente não seria possível sem a contribuição de um grupo. Meus sinceros agradecimentos a este grupo, que nele se incluem (mas não se restringem): aos colegas do grupo de pesquisa Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC), em especial ao meu orientador Júlio Cezar Estrella, meu co-orientador Cláudio Fabiano Motta Toledo, ao Stephan Reiff-Marganiec meu supervisor do estágio em pesquisa no exterior, que me auxiliou em toda minha estadia no Reino Unido.

Durante minha estadia no Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, duas pessoas que também foram importantes para minha formação foram: o professor Paulo Sérgio Lopes de Souza e o professor Adenilso da Silva Simão, que além de professores foram excelentes amigos com conselhos de grande valor.

Agradeço também a toda minha família e a Deus, pelo apoio, confiança e oportunidades a mim depositados. Em especial a minha mãe Maria Cristina de Melo que me apoiou desde sempre e almejou pelo meu sucesso e, a minha mulher Patrícia e minha futura filha Alice, que foram fundamentais para meu foco nas etapas finais.

Além desses, agradeço também ao CNPq, Capes e principalmente o apoio da FAPESP, através dos processos 2015/11623-4 e 2016/14219-2, que contribuiu de forma crucial para o êxito deste trabalho.

*“O valor real de qualquer coisa,
é baseado no tempo e esforço gasto para obtê-la.”
(Texto adaptado do livro A Tradição da Liberdade)*

RESUMO

AZEVEDO, L. J. M. **Desenvolvimento e avaliação de algoritmos de otimização para o cumprimento de acordos de níveis de serviços em nuvem.** 2018. 107 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

Atualmente o acesso a um ambiente de computação em nuvem é fornecido sob demanda, o que permite que provedores ofereçam serviços de forma elástica aos clientes. Embora a nuvem permita uma abstração do comportamento da infraestrutura (lógica e física) dos provedores de serviços, nem sempre é possível oferecer serviços aos clientes de modo que os provedores consigam cumprir adequadamente os acordos de níveis de serviços. Para permitir o cumprimento desses contratos, provedores de serviços precisam de mecanismos que envolvam algoritmos de balanceamento de carga, com o objetivo de fornecer uma distribuição eficiente da carga para os recursos disponíveis. Entretanto, os trabalhos presentes na literatura não abordam de forma adequada o problema de equacionamento de recursos em função das necessidades dos clientes, pois consideram em sua maioria um conjunto limitado de objetivos a serem analisados e cumpridos. Neste contexto, o objetivo deste projeto de mestrado foi desenvolver e avaliar algoritmos disponíveis na literatura que abordem a otimização combinatória para o provisionamento de recursos computacionais, buscando otimizar o uso eficiente da infraestrutura e cumprir os acordos de nível de serviço definidos entre clientes e provedores.

Palavras-chave: Computação em Nuvem, Qualidade de serviço, Provisionamento de recursos, Otimização.

ABSTRACT

AZEVEDO, L. J. M. **Development and evaluation of optimization algorithms for the fulfillment of cloud service level agreements.** 2018. 107 p. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2018.

Currently the access to a cloud computing environment is provided on demand, which allows providers to offer elasticity services to customers. Although the cloud allows an abstraction of the infrastructure behavior of the service providers (logical and physical), the fulfillment of the Service Level Agreements (SLAs) is challenging, because according with the demand and system configuration, the providers cannot ensure the customers requirements. There is a necessity of mechanisms that consider load balancing algorithms to provide an efficient load distribution in the available resources. However, the papers available in the literature do not efficiently address the problem of resource equation considering the customers requirements, because they consider a limited set of objectives to be analysed and fulfilled. Therefore, this master project aims to develop and evaluate algorithms available in the literature that address the combinatorial optimization and the multi-objective approach to handle the computational resources during the execution time, trying to optimize the efficient use of the resources available in the infrastructure and fulfill the service level agreements defined between the clients and providers.

Keywords: Cloud, Quality of Service, Resources provisioning, Optimization.

LISTA DE ILUSTRAÇÕES

Figura 1 – Modelo visual da definição de computação em Nuvem proposta pelo NIST - Adaptado de Alliance (2011).	8
Figura 2 – Gerenciamento do ciclo de vida do SLA (FANIYI; BAHSOON, 2015). . . .	17
Figura 3 – Taxonomia dos métodos de otimização (adaptado de (TALBI, 2009)). . . .	21
Figura 4 – Método de seleção por roleta.	25
Figura 5 – Mutação com uma taxa de 10% e 100% em um indivíduo.	27
Figura 6 – Relação dos objetivos conflitantes Tempo de Custo.	28
Figura 7 – Representação do conceito da relação de Dominância de Pareto.	30
Figura 8 – Ilustração do conjuntos de soluções e da Fronteira de Pareto (BUI; ALAM, 2008).	30
Figura 9 – Modelo ou sistema de identificação do problema.	41
Figura 10 – Ciclo de vida identificado do SLA.	42
Figura 11 – Representação do gerenciamento de carga de uma arquitetura padrão em nuvem.	44
Figura 12 – Representação do gerenciamento de carga de uma arquitetura em nuvem com os algoritmos de otimização.	44
Figura 13 – Comportamento da arquitetura e ativação dos módulos de otimização. . . .	45
Figura 14 – Funcionamento do ReMM (BATISTA, 2016).	49
Figura 15 – Arquitetura proposta em Nakamura (NAKAMURA, 2017).	51
Figura 16 – Representação da codificação da tripla (s, m, l)	52
Figura 17 – Estrutura da população e migração.	54
Figura 18 – Cruzamento uniforme.	55
Figura 19 – Cruzamento Blx- α	55
Figura 20 – Ilustração do NSGA-II (adaptado de Deb <i>et al.</i> (2002)).	60
Figura 21 – Distância entre as funções objetivo (adaptado de Deb <i>et al.</i> (2002)).	62
Figura 22 – Linha do tempo da negociação do SLA entre cliente e provedor.	68
Figura 23 – Distribuição gaussiana.	70
Figura 24 – Distribuição gaussiana com a gaussiana dos valores aberrantes.	70
Figura 25 – Gráfico do tempo de resposta do modelo determinístico e o μ AG em razão de cada configuração aplicada.	75
Figura 26 – Qualidade das soluções dos algoritmos SA, TB e MPGA em relação ao método determinístico com o Apache Benchmark.	76
Figura 27 – Ampliação de um intervalo do gráfico da Figura 26.	77

Figura 28 – Qualidade das soluções dos algoritmos SA, TB e MPGA em relação ao método determinístico com o Smallpt benchmark.	77
Figura 29 – Análise de confiabilidade dos métodos.	78
Figura 30 – Análise de convergência do MPGA.	79
Figura 31 – Análise de convergência do TS.	80
Figura 32 – Comportamento da arquitetura com recursos limitados.	83
Figura 33 – Comportamento da arquitetura com recursos limitados para o conjunto de VMs [50,50,50].	84
Figura 34 – Comportamento da arquitetura com recursos limitados para o conjunto de VMs [25,25,25].	85
Figura 35 – Curva de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Apache.	89
Figura 36 – Curva de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Smallpt.	91
Figura 37 – Quantidade de fronteiras de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Apache.	92
Figura 38 – Quantidade de fronteiras de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Smallpt.	93

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmos Evolutivos	22
Algoritmo 2 – Seleção por torneio	25
Algoritmo 3 – Gerador de SLA	47
Algoritmo 4 – Algoritmo Determinístico	53
Algoritmo 5 – Algoritmo MPGA	54
Algoritmo 6 – Algoritmo μ GA	57
Algoritmo 7 – SA algorithm	58
Algoritmo 8 – Algoritmo TS	59
Algoritmo 9 – <i>fast-non-dominated-sort</i>	61
Algoritmo 10 – Algoritmo NSGA-II	63

LISTA DE TABELAS

Tabela 1 – Parâmetros do SLA e suas unidades de medida (FANIYI; BAHSOON, 2015)	15
Tabela 2 – Parâmetros do SLA por modelo de serviço (FANIYI; BAHSOON, 2015)	15
Tabela 3 – Modelos de negociação e suas características.	16
Tabela 4 – Contraste com os trabalhos relacionados	37
Tabela 5 – Exemplos de SLAs gerados	48
Tabela 6 – Variação do número de VMs de acordo com a configuração.	67
Tabela 7 – Variação do SLA.	67
Tabela 8 – Variação do número de VMs.	68
Tabela 9 – Configuração dos algoritmos	69
Tabela 10 – Tempo de resposta e número de combinações obtidas pelo método determinístico (cenário 1).	72
Tabela 11 – Tempo de resposta e taxa de sucesso obtido pelo μ AG em 3 minutos (cenário 1).	73
Tabela 12 – Taxa de sucesso obtido pelo μ AG em 3, 5 e 10 minutos.	73
Tabela 13 – Tempo de resposta e número de combinações obtidas pelo método determinístico.	74
Tabela 14 – Tempo de resposta obtido pelo μ AG em todas as configurações (cenário 2).	74
Tabela 15 – Tempo de resposta e taxa de sucesso obtido pelo MPGA em relação ao μ AG em 3 minutos (cenário 1).	76
Tabela 16 – Taxa de sucesso dos métodos.	79
Tabela 17 – Simulação em uma arquitetura sem gerenciamento otimizado de recursos, considerando recursos infinitos.	81
Tabela 18 – Simulação em uma arquitetura aplicando os algoritmos de otimização, considerando recursos infinitos.	81
Tabela 19 – Simulação em uma arquitetura aplicando o MPGA, considerando recursos limitados.	82
Tabela 20 – Simulação em uma arquitetura aplicando o MPGA, considerando o conjunto de VMs [50,50,50].	83
Tabela 21 – Simulação em uma arquitetura aplicando o MPGA, considerando o conjunto de VMs [25,25,25].	84
Tabela 22 – Simulação em uma arquitetura aplicando os algoritmos de otimização na fase de estabelecimento de contrato, considerando recursos ilimitados.	85

Tabela 23 – Simulação aplicando os algoritmos de otimização na fase de monitoramento dos SLAs, considerando recursos ilimitados.	86
Tabela 24 – Simulação em uma arquitetura aplicando os algoritmos de otimização na fase de estabelecimento de contrato, com o conjunto de VMs [200,200,200]. . .	86
Tabela 25 – Simulação aplicando os algoritmos de otimização na fase de monitoramento dos SLAs, com o conjunto de VMs [200,200,200].	87
Tabela 26 – Fluxo de execução e negociação de cada cliente.	87
Tabela 27 – Estado de cada cliente durante o fluxo de execução.	87
Tabela 28 – Simulação em uma arquitetura aplicando os algoritmos de otimização na fase de estabelecimento de contrato, com o conjunto de VMs [100,100,100]. . .	88
Tabela 29 – Simulação aplicando os algoritmos de otimização na fase de monitoramento dos SLAs, com o conjunto de VMs [100,100,100].	88
Tabela 30 – Fluxo de execução e negociação de cada cliente para um conjunto de [100,100,100] VMs.	88
Tabela 31 – Estado de cada cliente durante o fluxo de execução para um conjunto de [100,100,100] VMs.	89
Tabela 32 – Soluções geradas alternando a prioridade de cada parâmetro para os clientes W, X, Y, Z com benchmark Apache.	90
Tabela 33 – Soluções geradas alternando a prioridade de cada parâmetro para os cliente W, X, Y, Z com benchmark Smallpt.	90

LISTA DE ABREVIATURAS E SIGLAS

μ GA	<i>micro-Genetic Algorithm</i>
blx- α	<i>blend alpha crossover</i>
CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
EMO	<i>Evolutionary Multiobjective Optimization</i>
FIFO	<i>First In, First out</i>
IaaS	<i>Infrastructure as a Service</i>
LaSDPC	Laboratório de Sistemas Distribuídos e Programação Concorrente
LaSDPC	Laboratório de Sistemas Distribuídos e Programação Concorrente
MCDM	<i>Multi-Criteria Decision Making</i>
MOP	<i>Multiobjective Optimization Problem</i>
MPGA	<i>Multi-Population Genetic Algorithm</i>
PaaS	<i>Platform as a Service</i>
ProOF	<i>Professional Optimization Framework</i>
QoS	<i>Quality of Service</i>
ReMan	<i>Resource Manager</i>
ReMM	<i>Resource Management Module</i>
ReMM	<i>Resource Management Module</i>
SA	<i>Simulated Annealing</i>
SaaS	<i>Software as a Service</i>
SLA	<i>Service Level Agreement</i>
TS	<i>Tabu Search</i>
VM	<i>Virtual Machine</i>
VMM	<i>Virtual Machine Monitor</i>

SUMÁRIO

1	INTRODUÇÃO	1
1.1	Motivação e Objetivos	2
1.2	Estrutura do Documento	2
2	COMPUTAÇÃO EM NUVEM	5
2.1	Considerações Iniciais	5
2.2	Definição e arquitetura	5
2.3	Conceitos fundamentais	9
2.4	QoS	9
2.4.1	<i>Provisionamento de Recursos</i>	11
2.4.2	<i>Planejamento de Capacidade</i>	13
2.4.3	<i>Acordos de níveis de serviços</i>	14
2.5	Considerações Finais	17
3	ALGORITMOS DE OTIMIZAÇÃO MONO-OBJETIVO E MULTI-OBJETO	19
3.1	Considerações Iniciais	19
3.2	Conceitos Fundamentais	19
3.3	Metaheurísticas baseadas em solução única	21
3.4	Algoritmos evolutivos	21
3.4.1	<i>Motivação</i>	22
3.4.2	<i>Terminologias Básicas</i>	22
3.4.3	<i>Operadores Genéticos</i>	24
3.4.4	<i>Representação Real</i>	26
3.5	Problemas multi-objetivo	28
3.6	Considerações Finais	31
4	REVISÃO BIBLIOGRÁFICA	33
4.1	Considerações iniciais	33
4.2	Trabalhos relacionados no contexto geral de computação em nuvem	33
4.3	Trabalhos relacionados no contexto de computação em nuvem que abordam SLA	35
4.4	Principais características dos trabalhos relacionados	37
4.5	Considerações finais	38

5	DESENVOLVIMENTO DA ABORDAGEM PROPOSTA	39
5.1	Considerações Iniciais	39
5.2	Descrição do problema	39
5.3	<i>Framework</i> de otimização	42
5.4	Módulos de gerenciamento do SLA	46
5.4.1	<i>Integração do framework em outras arquiteturas</i>	48
5.5	Algoritmos considerados	51
5.5.1	<i>Codificação do problema</i>	52
5.5.2	<i>Função objetivo</i>	52
5.5.3	<i>Método Determinístico</i>	53
5.5.4	<i>MPGA</i>	53
5.5.5	<i>Micro-GA</i>	56
5.5.6	<i>Simulated Annealing</i>	57
5.5.7	<i>Tabu Search</i>	58
5.5.8	<i>NSGA-II</i>	59
6	AVALIAÇÃO EXPERIMENTAL	65
6.1	Considerações Iniciais	65
6.2	Avaliação do <i>framework</i> de otimização	65
6.2.1	<i>Configuração do Ambiente</i>	65
6.2.2	<i>Planejamento de experimentos</i>	66
6.2.2.1	<i>Cenário 1</i>	66
6.2.2.2	<i>Cenário 2</i>	66
6.2.2.3	<i>Cenário 3</i>	67
6.2.2.4	<i>Cenário 4</i>	69
6.2.2.5	<i>Cenário 5</i>	71
6.2.3	<i>Análise dos resultados</i>	72
6.2.3.1	<i>Resultados obtidos no cenário 1</i>	72
6.2.3.2	<i>Resultados obtidos no cenário 2</i>	73
6.2.3.3	<i>Resultados obtidos cenário 3</i>	76
6.2.3.4	<i>Resultados obtidos no cenário 4</i>	84
6.2.3.5	<i>Resultados obtidos no cenário 5</i>	88
6.3	Considerações Finais	91
7	CONCLUSÃO	95
8	PUBLICAÇÕES E TRABALHOS FUTUROS	97
	REFERÊNCIAS	99

GLOSSÁRIO 107

INTRODUÇÃO

Em razão da ascensão tecnológica de *hardware* e ao sucesso da Web, os recursos computacionais estão se tornando cada vez mais acessíveis (financeiramente) e com suporte a um alto desempenho, em comparação às máquinas de décadas atrás. Esta evolução tecnológica possibilitou o desenvolvimento de um paradigma denominado computação em nuvem (*Cloud Computing*), cuja finalidade é disponibilizar recursos computacionais (processamento, armazenamento, software, etc.) como utilidades, permitindo o atendimento de acordo com a demanda dos usuários.

A utilização da computação em nuvem surgiu como uma proposta do Google para todos os tipos de usuários (indivíduos e empresas) da Internet (CHEN; LI; CHEN, 2011). Como exemplos de tais serviços pode-se citar Google Docs ¹, Amazon Elastic Compute Cloud and Simple Storage Services ², Microsoft Windows Azure Platform ³, IBM Smart Business ⁴, Salesforce.com ⁵. Atualmente, o modelo de computação em nuvem vem sendo alvo de muitos debates e discussões, pois devido a inexistência de um padrão, cada provedor de serviço constrói seus serviços de computação em nuvem de acordo com a sua própria política.

Embora não haja uma definição de padrões para este paradigma, na prática os provedores disponibilizam serviços, os quais podem ser agrupados em três categorias: *Infrastructure as a Service* (IaaS), *Platform as a Service* (Paas) e *Software as a Service* (SaaS). A partir desses serviços básicos, o provedor define o seu modelo de negócio organizando um ambiente computacional, no qual componentes virtualizados são oferecidos para seus clientes em uma interface com a plataforma implementada.

Segundo Zhang, Cheng e Boutaba (2010), a computação em nuvem apresenta diversas

¹ <http://docs.google.com>

² <http://aws.amazon.com/ec2>

³ <http://www.windowsazure.com>

⁴ <http://www.ibm.com/cloud>

⁵ <http://www.salesforce.com>

características atrativas para proprietários de negócios, como baixo investimento em infraestrutura e custo de operação, alta escalabilidade e facilidade de acesso. Mesmo com tais vantagens, a tecnologia de computação em nuvem ainda possui algumas dificuldades, principalmente em termos de desempenho, confidencialidade de dados, escalabilidade, segurança e armazenamento de dados (YAN *et al.*, 2016) (RITTINGHOUSE; RANSOME, 2016).

1.1 Motivação e Objetivos

Para o desenvolvimento de um ambiente propício para a execução de serviços em um cenário em nuvem, inicialmente o provedor de serviço deve investir em muitos equipamentos computacionais de alto desempenho como servidores, discos e componentes de rede para o atendimento das mais variadas requisições oriundas dos usuários em um nível global.

Para melhor utilização e distribuição dos recursos computacionais, técnicas como a de virtualização são aplicadas. Esta técnica consiste em permitir que múltiplos sistemas operacionais coexistam sobre um mesmo *host* físico, tendo um forte isolamento lógico entre os componentes virtualizados (JING *et al.*, 2013), e vantagens como um melhor gerenciamento e tolerância a falhas (JING *et al.*, 2013). Esse gerenciamento deve ser considerado com base em otimização dos recursos computacionais dos provedores de serviço e também levando em consideração as necessidades dos clientes que “consomem” os recursos.

No entanto, é preciso verificar qual o impacto gerado no sistema durante o provimento de mais recursos computacionais para as máquinas virtuais a fim de manter o desempenho contratado por um cliente, o que também influencia no custo a ser pago. Tanto o impacto das mudanças realizadas quanto a resposta ao cliente precisam ser fornecidas em tempo real ou dentro de um prazo bastante curto.

A proposta deste projeto de mestrado visa o desenvolvimento e avaliação de algoritmos de otimização para o problema de provisionamento de recursos (instâncias de máquinas virtuais), com foco no cumprimento do acordo de nível de serviço. Métodos heurísticos e metaheurísticos se apresentam como técnicas de otimização capazes de fornecerem soluções de tempo computacional reduzido. Nesse contexto, algoritmos evolutivos têm sido amplamente utilizados na resolução dos mais variados tipos de problema.

1.2 Estrutura do Documento

Este documento está organizado da seguinte forma: a Seção 2 discorre sobre os conceitos envolvendo computação em nuvem e que compõem a base para o desenvolvimento deste projeto de mestrado como, modelos de prestação de serviços, tipos de nuvens, virtualização, QoS em computação em nuvem, provisionamento de recursos e planejamento de capacidade. A Seção 3 discorre sobre conceitos que envolvem computação evolutiva, que compõem a base para a

elaboração e desenvolvimento dos algoritmos de otimização. Na Seção 4 são discutidos os trabalhos relacionados. Na Seção 5 é abordado o desenvolvimento do trabalho. Na Seção 6 são apresentados os resultados. Por fim, na Seção 7 são discutidas as conclusões e na Seção 8 são apresentadas as publicações obtidas e trabalhos futuros.

COMPUTAÇÃO EM NUVEM

2.1 Considerações Iniciais

O crescimento constante da Internet e o desenvolvimento de plataformas que facilitam a mobilidade e acesso de dados, juntamente ao ganho de velocidade da rede mundial de computadores, viabilizam pesquisas e oportunidades de negócio. Nesse contexto, a computação vem ganhando espaço e crescendo paralelamente à Internet.

Este capítulo tem por objetivo apresentar conceitos de computação em nuvem para a compreensão deste trabalho, além de discutir a contextualização do problema a ser tratado. A Seção 2.2, apresenta uma definição de computação em nuvem, destaca a estrutura de uma arquitetura em nuvem e discorre sobre a definição e importância da virtualização. A Seção 2.4, apresenta os principais atributos relacionados a qualidade de serviço em nuvem. A Seção 2.4.1, discute a importância do provisionamento de recursos. Em seguida, a Seção 2.4.2, discorre sobre a importância do planejamento de capacidade. Por fim, a Seção 5.2, aborda a contextualização do problema a ser tratado neste trabalho.

2.2 Definição e arquitetura

Computação em nuvem pode ser definida de várias formas. De maneira formal, o NIST (*National Institute of Standards and Technology*) traz a seguinte definição: “um modelo que permite o acesso à rede de forma ubíqua, conveniente e sob demanda, a um conjunto compartilhado de recursos computacionais configuráveis que podem ser rapidamente provisionados e liberados com um esforço mínimo de gestão ou interação com o provedor de serviços” (MELL; GRANCE, 2011).

A computação em nuvem apresenta uma proposta que fornece diversas aplicações, por meio do compartilhamento de ferramentas computacionais e pela interligação de sistemas. Con-

tudo, a computação em nuvem não deve ser vista apenas como um conjunto de computadores. Uma arquitetura de nuvem deve fornecer uma infraestrutura composta por diversas características, envolvendo escalonamento, escalabilidade de recursos, monitoramento, segurança e balanceamento de carga. Outro ponto importante, é que uma arquitetura em nuvem proporciona diversas vantagens aos usuários, tais como (VELEV; ZLATEVA, 2011):

- Escalabilidade e elasticidade instantânea;
- Compartilhamento de recursos (*hardware*, *software*, banco de dados, entre outras);
- Gerenciamento por meio de APIs (*Application Programming Interface*);
- Alta abstração dos recursos;
- Redução dos custos, pois os serviços são oferecidos sob demanda e, não necessitam da compra de *hardware*;
- Alta mobilidade;
- Provisionamento de recursos;
- Ubiquidade.

Como mencionado anteriormente, o acesso a nuvem é fornecido sob demanda. Outra característica peculiar, é que ela é baseada em serviços sob demanda (CHIEU *et al.*, 2009). De propósito geral, a nuvem oferece uma abstração do comportamento da infraestrutura (lógica e física) dos provedores de serviços, o que acaba a identificando como um *middleware*.

Muitos estudos encontrados na literatura apontam a existência de diversos modelos de serviços prestados pela computação em nuvem, contudo, são apresentados três principais, sendo eles (VELEV; ZLATEVA, 2011) (CHIEU *et al.*, 2009) (MELL; GRANCE, 2011):

- **Software como Serviço (*Software as a Service* (SaaS))**: esse modelo oferece o *software* como um serviço sob demanda. Devido ao crescimento da Internet, uma largura de banda maior é oferecida, fazendo com que a utilização do SaaS seja cada vez mais comum. Dessa forma, um único *software* pode ser oferecido simultaneamente a múltiplos clientes;
- **Plataforma como Serviço (*Platform as a Service* (PaaS))**: nesse modelo é oferecido a plataforma como um serviço sob demanda, na qual, é permitido que os usuários desenvolvam suas aplicações utilizando APIs. Ainda nesse modelo, o usuário é capaz de desenvolver aplicações com linguagens de programação específicas (de sua preferência);
- **Infraestrutura como Serviço (*Infrastructure as a Service* (IaaS))**: esse último modelo, oferece uma camada mais baixa como um serviço sob demanda, a qual, está relacionado

às máquinas virtuais e outras abstrações de *hardware*. Dessa maneira é fornecida uma infraestrutura como serviço, sem que o usuário necessite realizar a aquisição de um novo *hardware*.

Além desses modelos de serviço, de acordo com sua finalidade e localização, a computação em nuvem é composta por mais quatro modelos de implementação, sendo elas (VELEV; ZLATEVA, 2011) (MELL; GRANCE, 2011):

- **Nuvem Pública:** em nuvens públicas a infraestrutura é hospedada pelo provedor. Dessa forma, o usuário não possui a visão de como é realizado o controle sobre a localização da hospedagem da infraestrutura, a qual, é compartilhada entre quaisquer usuários;
- **Nuvem Privada:** é o oposto das nuvens públicas. Neste modelo, por meio de um *data center* privado, as nuvens são dedicadas a um conjunto específicos de usuários. Dessa forma, o usuário tem total controle sobre as aplicações da nuvem;
- **Híbrida:** compreende-se como um misto de nuvem pública e privada. Permite, por meio de nuvens públicas, que as nuvens privadas tenham seus recursos ampliados;
- **Comunitária:** nesse modelo, diferentemente dos outros, a nuvem pode ser administrada também por um grupo de usuário, ou seja, não necessariamente ela será oferecida por um provedor. Portanto, nesse modelo, a infraestrutura da nuvem é compartilhada por diversos usuários para uma aplicação específica.

A Figura 1 destaca uma representação da estrutura de uma arquitetura em nuvem (com as características descritas citadas anteriormente).

Segundo Carissimi (2008), “a virtualização consiste em estender ou substituir um recurso, ou uma interface, existente por um outro, de modo a imitar um comportamento”, ou seja, tem como finalidade a abstração. No contexto de computação em nuvem, a virtualização permite que um único computador físico possa assumir o papel (ou se comporte) como vários computadores lógicos independentes (tais computadores lógicos são denominados máquinas virtuais).

A virtualização tornou-se fundamental para a computação em nuvem, pois oferece vantagens importantes no compartilhamento, gerenciamento e isolamento dos recursos (RIMAL; CHOI; LUMB, 2009). Segundo Menken e Blokdijsk (2010), como consequência, alguns objetivos e benefícios são alcançados com a utilização da virtualização, sendo eles:

- **Redução dos custos com gerenciamento de recursos:** com a virtualização é possível utilizar melhor os recursos físicos, possibilitando, dessa forma, uma economia relacionada à aquisição desnecessária de *hardware*;

Figura 1 – Modelo visual da definição de computação em Nuvem proposta pelo NIST - Adaptado de Alliance (2011).

- **Uso eficiente dos recursos:** é possível, por meio da virtualização, uma gerência adequada dos recursos físicos, de tal forma a obter-se o uso eficiente dos recursos;
- **Isolamento:** apesar das máquinas virtuais compartilharem recursos (como processador e memória), elas se comportam como computadores lógicos independentes, portanto, os *softwares* ficam isolados uns dos outros;
- **Maior flexibilidade e portabilidade:** com a virtualização é possível um encapsulamento e isolamento de *softwares*. Dessa forma, fica mais simples portar os *softwares* de um *host* físico para outro, ou até mesmo instanciar uma nova máquina virtual de acordo com a necessidade, evitando gastos adicionais com *hardware*;
- **Independência de *hardware*:** a virtualização proporciona propriedades como encapsulamento, compatibilidade e independência do *hardware*, sendo possível mover uma máquina virtual de um *host* físico para outro sem a necessidade de alteração dos *drivers* dos dispositivos;
- **Eliminação de problemas com compatibilidade:** por meio da virtualização, é possível que diferentes aplicações e sistemas operacionais possam ser instalados em um mesmo *host* físico, sem que um altere a configuração do outro.

Apesar das máquinas virtuais proporcionarem uma melhor utilização do *hardware*, elas podem gerar algumas restrições, pois imitam o comportamento de um sistema computacional

dentro de outro. As máquinas virtuais podem ser implementadas de duas maneiras (CARISSIMI, 2008): a primeira é a Máquina Virtual de Processos, que pode ser vista como uma aplicação do sistema operacional executando em modo cliente; outra é o Hypervisor ou Monitor de Máquina virtual (*Virtual Machine Monitor* (VMM)), um *software* que atua em uma camada entre o sistema operacional e o *hardware*.

O VMM é responsável por hospedar as máquinas virtuais, pela virtualização e controle dos recursos compartilhados (processador, memória e dispositivos de entrada e saída) (ROSE, 2004). O VMM também atua como um escalonador, organizando a ordem de execução das máquinas virtuais (MENASCÉ, 2005).

A virtualização é uma parte fundamental da computação em nuvem para atingir seus objetivos. Os provedores de computação em nuvem normalmente utilizam virtualizadores a fim de virtualizar as máquinas fornecidas aos usuários (RIMAL; CHOI; LUMB, 2009).

A virtualização pode ser feita de duas formas (LI; LI; JIANG, 2010): a primeira é a virtualização total, a qual fornece ao sistema operacional virtualizado uma réplica do *hardware* da máquina física em que está hospedado. Dessa forma, o sistema operacional não sofre alterações; a segunda é a para-virtualização, em que o sistema operacional virtualizado sofre algumas modificações, ou seja, os dispositivos de *hardware* são acessados por *drivers*, o que acaba afetando a portabilidade da *Virtual Machine* (VM).

2.3 Conceitos fundamentais

A computação em nuvem acaba sendo uma extensão das tecnologias de virtualização que permitem o gerenciamento de máquinas virtuais por meio de sistemas conectados fisicamente (LI; LI; JIANG, 2010). A computação em nuvem exige mecanismos que possam manter o uso eficiente dos recursos e que permitam que o atendimento sob demanda possa ser efetivado. Portanto, existem alguns fatores fundamentais que devem ser levados em consideração, como, *Quality of Service* (QoS), o provisionamento de recursos, planejamento de capacidade e *Service Level Agreement* (SLA).

2.4 QoS

A qualidade de serviço é dada pela visão do usuário em relação à eficiência dos serviços prestados. Segundo Vegesna (2001) e Ferguson e Huston (1998) o objetivo da QoS é prover ao usuário a garantia do serviço prestado com um mínimo de perda de desempenho, consistência e integridade dos dados.

Segundo Stantchev e Schröpfer (2009), devido à vasta gama de clientes na nuvem com comportamentos imprevisíveis, a garantia de QoS torna-se uma tarefa complexa, pois a nuvem deve ser monitorada constantemente. Para isso, um *Service Level Agreement* (SLA) deve ser

definido. Um SLA é um acordo de nível de serviço firmado para que haja o cumprimento das metas estipuladas (GUO *et al.*, 2007). Outro ponto relevante é que não há um padrão de provimento de QoS. Como há diversos provedores, cada um possui uma forma de prestação de serviços (CHARD, 2011).

Apesar de não haver um padrão para a garantia de QoS, há diversos fatores que devem ser levados em consideração, dentre os quais destacam-se (WEBER, 2007):

- **Disponibilidade:** refere-se à capacidade de um sistema estar acessível, mesmo com a necessidade de pausas para reparo;
- **Segurança:** neste ponto são considerados problemas de proteção, tais como, invasão, fraudes, acessos indevidos, vazamento de informações, entre outros;
- **Segurança de funcionamento:** considera a probabilidade de uma falha inesperada. Talvez seja necessário parar a operação para evitar danos maiores;
- **Confiabilidade:** é a capacidade de manter o funcionamento e solicitações de serviço dentro das condições definidas.

Devido a grande diversidade de serviços prestados pelos provedores, descobrir quais provedores realmente satisfazem as exigências dos clientes, acaba se tornando um desafio. Outro ponto, é que um mesmo cliente pode ter diferentes exigências a diversos provedores, ou um mesmo provedor pode prestar um mesmo (ou vários) serviços a vários clientes. Dessa forma, há dificuldade de avaliar os vários serviços prestados em diferentes níveis e ainda em diversos provedores. Portanto, além de descobrir quais serviços são oferecidos pelos diversos provedores, é importante também saber avaliar qual o serviço de qual nuvem é mais adequado.

Nesse contexto, com o objetivo de realizar essa avaliação, o CSMIC (*Cloud Service Measurement Index Consortium*) tem trabalhado em uma definição de medidas globalmente aceitas para serviços em nuvem. Para tal, é necessário identificar um conjunto de atributos que juntos compõem um índice denominado SMI (*Service Measurement Index*), que fornece uma análise comparativa dos diferentes tipos de serviços em nuvem (GARG; VERSTEEG; BUYYA, 2011). Esse índice proporciona uma ampla visão de QoS que os clientes necessitam para selecionar serviços de um provedor. Para isso, alguns atributos devem ser considerados, tais como (GARG; VERSTEEG; BUYYA, 2011):

- **Cumprimento do acordo do serviço:** referente a capacidade de um serviço em nuvem cumprir com o contrato;
- **Desempenho:** os diferentes serviços oferecidos pelos provedores podem apresentar diferentes desempenhos em termos de tempo de resposta, funcionalidade e precisão. É

necessário que o usuário saiba quais serviços de quais provedores atendem melhor as suas expectativas;

- **Agilidade:** a agilidade de um serviço pode ser medida em termos de portabilidade, elasticidade, flexibilidade e adaptabilidade. No SMI, a agilidade é dada em razão de quão rapidamente novas capacidades (ou serviços, dependendo do contexto) são integrados de acordo com a exigência do cliente;
- **Responsabilidade:** esse atributo é importante para conquistar a confiança do cliente, pois é utilizado para medir diversas características dos provedores, tais como, cumprimento do contrato de nível de serviço, direito de propriedade de dados, ética, auditabilidade e sustentabilidade;
- **Custo:** o custo é um atributo essencial para computação em nuvem. Contudo, como clientes distintos possuem características distintas, consequentemente possuirão custos distintos para determinados serviços, sendo importante expressar em qual característica difere o custo, em termos de serviço prestado, poder de processamento, memória, disco, entre outras características;
- **Usabilidade:** diversos fatores estão relacionados à usabilidade, como capacidade de aprendizagem da tecnologia, instabilidade, o quanto é acessível e o quão é operável;
- **Segurança e privacidade:** outro atributo extremamente importante para o cliente, pois refere-se à proteção e privacidade dos seus dados. Segurança e privacidade geralmente são associados a três características: a integridade, confidencialidade e a disponibilidade dos dados.

Outro ponto que dificulta a provisão de QoS é o fato dos serviços serem prestados sob demanda. Esse fato faz com que haja uma dinamicidade na execução dos serviços, podendo em um certo momento um ou vários serviços serem consumidos massivamente e em outro momento o serviço ficar ocioso. Independentemente do caso, a configuração da arquitetura deve atender às necessidades do cliente, sem que isso viole a QoS estabelecida no SLA. Nesse contexto, é necessário incluir uma outra característica, que é a Computação Autônoma na nuvem, na qual o sistema tem a capacidade de se auto-gerenciar ([HASSAN; AL-JUMEILY; HUSSAIN, 2009](#)), e para tal, deve-se fazer um levantamento minucioso sobre o provisionamento de recursos.

2.4.1 *Provisionamento de Recursos*

O modelo de computação em nuvem reduz a sobrecarga de tecnologias da informação ao cliente final, pois torna propício a redução do custo total da aquisição, além de grande escalabilidade, flexibilidade e melhor gerenciamento de toda a infraestrutura relacionada à uma

empresa (RITTINGHOUSE; RANSOME, 2009). Contudo, para que se possa usufruir dessas vantagens é necessário ter um mecanismo eficiente de provisionamento de recursos.

Ao aplicar-se um modelo eficiente de provisionamento de recursos em computação em nuvem, pode-se ter uma melhor eficiência dos serviços prestados. Tal eficiência está relacionada à redução dos custos, melhor utilização dos recursos computacionais, economia de energia, entre outros. Por outro lado, ter um modelo eficiente de provisionamento de recursos não é uma tarefa trivial. Segundo Calheiros, Ranjan e Buyya (2011), alguns fatores importantes estão relacionados, tais como, as melhores configurações de *hardware* e *software* para o cumprimento dos SLAs.

Para haver um eficiente provisionamento de recursos em nuvem, diversos desafios são encontrados, os quais integram modelagem de carga e desempenho, virtualização, monitoramento e depuração dos recursos. Além desses desafios, há ainda algumas situações que podem prejudicar o eficiente provisionamento de recursos, tais como (CALHEIROS; RANJAN; BUYYA, 2011):

- **Comportamento inesperado:** o não determinismo do comportamento do sistema pode impossibilitar o provisionamento de recursos de forma eficiente, assim como o cumprimento do acordo a nível de serviço;
- **Erro de estimativa:** está relacionado a um arranjo equivocado de recursos, os quais podem causar uma super ou sub estimativa do lado do cliente, consequentemente, gerando grande impacto na QoS relacionada ao contrato e no custo do serviço;
- **Carga dinâmica:** uma carga dinâmica pode ocorrer por meio de diversas formas, desde o grau de clientes de uma aplicação até fatores climáticos. Nesse contexto, sérios problemas podem surgir durante o provisionamento de recursos (pois afeta tanto a estimativa do comportamento da carga como a definição dos recursos).

Um ponto importante é que nem sempre aumentar a quantidade de recursos significa aumentar o desempenho. Contudo, quanto mais recursos são requisitados, consequentemente incorre em um custo maior. Ou ainda, nem sempre é necessário aumentar todos os recursos computacionais para aumentar o desempenho, e talvez, em pontos específicos já seja o suficiente para atender às necessidades dos clientes.

Provavelmente um mecanismo eficiente de provisionamento de recursos esteja relacionado tanto à aplicação da escalabilidade vertical quanto da horizontal. A escalabilidade vertical aplica uma metodologia onde os recursos são aumentados de forma incremental, ou seja, modificando parâmetros como CPU e memória de forma unitária, e não em conjunto, enquanto a escalabilidade horizontal trabalha com classes de VM, na qual, a necessidade de mais recursos implica na contratação de mais VMs.

Os provedores de serviços comerciais como Amazon EC2 ¹ e Microsoft Azure ² utilizam o escalonamento horizontal. Esta é a maneira que as empresas realizam o escalonamento, pois não precisam atentar-se às necessidades específicas de cada cliente, tais como o tempo de resposta, disponibilidade e custos desejados por parte do mesmo. Além disso, aplicar a escalabilidade horizontal é mais lucrativo para a empresa.

Portanto, encontrar um provisionamento eficiente de recursos em sistemas como *clusters*, grades e computação em nuvem é uma tarefa complexa. Uma vez que a demanda e atendimento às requisições pode ser dinâmica e incerta, diversas outras estratégias devem ser estudadas, com destaque para o planejamento de capacidade. Essa estratégia é utilizada para estudar tendências de utilização dos recursos ao longo de um período, de modo a monitorar o consumo e gerenciamento do contrato estabelecido, a fim de garantir a QoS.

2.4.2 Planejamento de Capacidade

O planejamento de capacidade é fundamental para garantir a QoS. Visto que os recursos são finitos e custosos, é necessário que haja um planejamento de capacidade a fim de garantir que as aplicações funcionem de forma eficiente, e consequentemente atinja níveis aceitáveis da QoS estipulada no SLA.

Sistemas que prestam serviços sob demanda possuem cargas de trabalho distintas, as quais, aplicadas sob uma determinada rede resultam em diferentes quantidades possíveis de atendimento aos serviços (MONKS, 2006) (JAIN, 2008) (ALMEIDA; MENASCÉ, 2002). Dessa forma, considerar a carga de serviço e procurar sempre identificar com maior detalhamento os fatores envolvidos, são características fundamentais para o cumprimento do acordo de nível de serviços.

Segundo MONKS (2006), o processo de planejamento de capacidade pode ser decomposto em quatro fases:

- Caracterização da carga de trabalho;
- Definição dos níveis de serviço (QoS);
- Previsão de desempenho sobre os recursos disponíveis;
- Adequação dos recursos.

A adequação dos recursos está diretamente relacionada com a provisão de recursos, pois necessita de técnicas que permitam prever o desempenho do sistema diante de novas cargas de trabalho. Uma vez que se pode fazer essa previsão, é necessário saber o limite de recursos

¹ <http://aws.amazon.com/ec2/>

² <http://www.windowsazure.com/en-us/pricing/details/virtual-machines/>

disponíveis. Uma questão fundamental do planejamento de capacidade está relacionada com a estimação do ponto (momento) em que a demanda excederá o limite de recursos disponíveis.

Os métodos para mensurar esses pontos requerem que as redes (reais) sejam avaliadas por experimentação (MONKS, 2006). Monitorar diretamente a rede traz algumas vantagens, pois a real utilização da rede está sendo medida, e isso faz com que nenhum detalhe seja excluído.

Em situações onde deseja-se avaliar o desempenho da futura implementação de uma rede, as técnicas de medições reais não são possíveis. Nesse caso, são utilizados modelos analíticos ou simulados, os quais diferem entre si em termos de precisão de resultados, tempo de aplicação e custo de utilização. No contexto deste trabalho, como o Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) possui uma boa infraestrutura e, possibilita a aplicação de testes reais, o desenvolvimento e avaliação dos algoritmos de otimização poderão ser executados em um ambiente real (*clusters* do LaSDPC)³, levando em conta diversos fatores que em um ambiente simulado não são considerados. A implementação em ambiente real foi iniciada, contudo, não finalizada neste projeto de mestrado, entrando assim em trabalhos futuros.

2.4.3 Acordos de níveis de serviços

Um contrato entre provedor e cliente pode incluir qualidade de serviço como tempo de resposta e disponibilidade, termos legais (*copyrights*), direitos intelectuais e de negócios como condições financeiras e taxas. Opcionalmente, a licença de um serviço pode incluir termos do SLA, contudo, uma licença de serviço é algo unilateral por parte do provedor, por outro lado, um acordo a nível de serviço é algo bilateral, um acordo entre o provedor e o cliente.

Neste trabalho, é abordado apenas as condições estabelecidas no contrato a nível de serviço. Em um SLA diversos atributos de QoS podem ser levados em consideração. Na Tabela 1 são apresentados alguns exemplos desses atributos e suas unidades de medida, e na Tabela 2 em quais níveis de serviço se aplica cada parâmetro.

Antes de ser estabelecido um SLA entre cliente e provedor, uma negociação é realizada entre as partes interessadas. Na literatura há alguns modelos de negociação, os quais podem ser compreendidos como descrições da forma como é realizado o processo de negociação entre cliente e provedor:

- **Aceita/rejeita (“abordagem de mercado”):** Neste processo, nenhum parâmetro é negociável. O vendedor/provedor apenas aceita ou rejeita qualquer escolha feita pelo cliente. Trata-se do protocolo de negociação mais simples, pois o cliente oferta e o vendedor apenas aceita ou rejeita;

³ No link: <http://infra.lasdpicmc.usp.br> é possível verificar uma descrição detalhada da infraestrutura que será utilizada para o desenvolvimento deste projeto de mestrado

Tabela 1 – Parâmetros do SLA e suas unidades de medida (FANIYI; BAHSOON, 2015)

Parâmetros do SLA	Unidade de Medida
Violações permitidas	Numero máximo de violações permitidas no SLA
Disponibilidade	Percentual do tempo de atividade ou inatividade
Largura de banda	Vazão (KM/s ou MBit/s), tempo de transferência de dados, tempo da rota dos dados
Custo	Dólares/reais, custo das VMs por unidade de tempo
CPU	MIPS, MHz, numero de núcleos, utilização de CPU
Duração do serviço	tempo de resposta, tempo de execução, tempo de duração do SLA
Memória	GB, MB
Taxa de penalidade	média da penalidade (\$)
Desempenho	tempo de resposta, latência, tempo de processamento
Taxa de chegada das requisições	requisições atendidas por segundo, parâmetro de QoS do cliente
Espaço de armazenamento	GB, taxa de leitura e escrita (MB)
Frequência de atualização das requisições	parâmetro de QoS do Cliente
Energia	custo por kWh
Segurança	Não especificado
Outros	Tempo de inicialização das VMs, classificação do cliente (ouro/prata/bronze), numero de VMs, prioridade das tarefas

Tabela 2 – Parâmetros do SLA por modelo de serviço (FANIYI; BAHSOON, 2015)

XaaS	Parâmetros do SLA
SaaS	violações permitidas, largura de banda, CPU, duração do serviço, memória, taxa de penalidade, desempenho, taxa de chegada das requisições, espaço de armazenamento
PaaS	Custo, desempenho
IaaS	Disponibilidade, largura de banda, custo, CPU, energia, memória, taxa de penalidade, desempenho, taxa de chegada das requisições, espaço de armazenamento

- **Oferta discreta (“pegar ou largar”):** assume-se que os provedores já possuem de forma pré-definida um conjunto de serviços oferecidos, os quais podem ser acessados pelo cliente. O cliente tem a oferta de um serviço, contudo, essa negociação não é flexível e o cliente apenas concorda com os termos ou rejeita o serviço;
- **Proposta enviada:** esse modelo sempre parte do protocolo aceita/rejeita, no qual o consumidor procura por um serviço apropriado. O cliente especifica os requisitos e divulga sua oferta, esperando que algum provedor possa atender as especificações;
- **Multi-fase (n-fases):** o consumidor tem a possibilidade de adaptar um oferta e enviá-la para o provedor como uma contra-proposta, e o provedor pode modificá-la e enviar outra

contra-proposta;

- **Leilão inglês:** começa com uma proposta estabelecida pelo auditor, que receberá outras ofertas aumentando o preço, se não recebem mais nenhuma oferta, então quem deu o ultimo lance recebe o serviço;
- **Leilão holandês:** em contraste ao modelo anterior, o leilão holandês começa com uma oferta grande (preço alto) e vai baixando de acordo com as ofertas (o valor mínimo aceitável é definido pelo provedor);
- **Leilão Vickrey:** similar ao leilão inglês, o participante com a maior oferta vence a disputa. Contudo, nesse modelo, o vencedor paga apenas a segunda maior oferta e não a maior.

A Tabela 3 apresenta um contrates desses modelos com o modelo aplicado neste trabalho.

Tabela 3 – Modelos de negociação e suas características.

Modelos de Negociação	Cliente oferta	Provedor oferta	Cliente escolhe	Provedor escolhe	Permite contra-proposta	Automático	Evita violações precoces
Aceita/rejeita	✓			✓			
Oferta discreta		✓	✓				
Proposta enviada	✓			✓			
Multi-fase	✓	✓	✓	✓	✓		
Leilão inglês		✓	✓				
Leilão holandês		✓	✓				
Leilão Vickrey		✓	✓				
Modelo utilizado neste trabalho	✓	✓	✓	✓	✓	✓	✓

Após todo o processo de negociação e o SLA ser estabelecido, é necessário que haja um gerenciamento desse acordo. O gerenciamento de um SLA é uma tarefa composta por várias fases, tais como, negociação, implantação, monitoramento, relatório e finalização. A Figura 2 apresenta a sequencia em que essas fases são realizadas. A seguir é apresentada uma breve descrição sobre cada fase desse ciclo (FANIYI; BAHSOON, 2015):

1. **Negociação do SLA:** nessa fase, são definidos os termos dos serviços e o acordo do nível em que o serviço é oferecido, como custo monetário, disponibilidade, tempo de resposta, entre outros. Geralmente, o SLA é formado por meio de um modelo padronizado de SLAs que envolve as partes interessadas;
2. **Estabelecimento do SLA:** nessa fase, as requisições do cliente são atribuídas aos recursos contratados (estabelecidos no SLA). O ideal, é que os recursos contratados possam satisfazer as exigências estabelecidas pelo cliente. Geralmente, SLAs possuem diferentes classes como, ouro, prata e bronze, onde cada classe possui um tipo específico de SLA;
3. **Monitoramento do SLA:** durante o ciclo de vida do SLA é importante haver periodicamente um monitoramento em relação à taxa de utilização dos recursos, requisições atendidas, entre outros fatores relacionados ao ciclo de vida do SLA. O monitoramento

é uma fase delicada e pode gerar um gargalo, pois a quantidade de dados gerados pode crescer rapidamente. Portanto, essa fase deve ser desenvolvida com cuidado, para coletar apenas informações necessárias.

4. **Gerenciamento de violações:** alertas de violação podem indicar a probabilidade de uma tarefa não ser atendida, ou ainda, indicar a violação de um ou mais dos parâmetros estabelecidos no SLA. Geralmente, esses alertas são reportados na fase do monitoramento. O objetivo dessa fase é reduzir os riscos de violações, pois no pior caso, as tarefas serão executadas novamente caso não haja atenuação do risco.
5. **Relatório e finalização do SLA:** o objetivo principal dessa fase final é realizar um relatório detalhado do ciclo de vida do SLA e do provisionamento de recursos. Essa fase prove mecanismos para ambas as partes do contrato para que o SLA seja finalizado, ou ainda, para a resposta de alguma violação causada em alguma parte específica do SLA.

Figura 2 – Gerenciamento do ciclo de vida do SLA (FANIYI; BAHSOON, 2015).

2.5 Considerações Finais

Este capítulo apresentou conceitos de computação em nuvem relacionados a esta dissertação de mestrado. Além disso, foi discutido a contextualização do problema, na qual ressaltou-se a importância do provisionamento de recursos e planejamento de capacidade. O capítulo seguinte apresenta os conceitos e definições da computação evolutiva.

ALGORITMOS DE OTIMIZAÇÃO MONO-OBJETIVO E MULTI-OBJETO

3.1 Considerações Iniciais

Neste capítulo, conceitos relacionados aos algoritmos de otimização mono-objetivo e multi-objetivo serão introduzidos. Na Seção 3.2, são apresentadas definições de métodos exatos, aproximados, heurísticos e metaheurísticos. Na Seção 3.3, são conceituados alguns algoritmos de busca local estudados neste trabalho. Uma breve introdução aos algoritmos evolutivos (AEs) é apresentada na Seção 3.4, uma vez que tais métodos se mostraram mais efetivos nos estudos conduzidos. Assim, a Seção 3.4.1 discorre sobre a motivação para utilização da computação evolutiva. Na Seção 3.4.2, são apresentados os fundamentos biológicos e terminologias básicas. Na Seção 3.4.3 são descritos alguns operadores utilizados nos AEs. A Seção 3.4.4 discute a representação real em algoritmos evolutivos. Por último, a Seção 3.5 introduz conceitos relacionados ao tratamento de problemas multi-objetivo.

3.2 Conceitos Fundamentais

Há na literatura diversos métodos de otimização que podem ser classificados em linhas gerais como métodos do tipo exato, de aproximação, heurístico e meta-heurístico. Os métodos exatos se destacam por garantir a melhor solução; podendo-se afirmar assim que os métodos exatos encontram a solução ótima. Contudo, os métodos exatos, quando aplicados a problemas caracterizados como NP-completos, podem demandar tempo exponencial para encontrar soluções. Esse é o caso do problema tratado neste projeto de mestrado. Na literatura, destacam-se dois métodos exatos clássicos: a programação dinâmica e o método de enumeração *branch and bound* (TALBI, 2009). A programação dinâmica parte do princípio de “dividir para conquistar”, aplicando um método recursivo. Assim, o problema geral é dividido em problemas menores e

a solução de cada problema menor compõe a solução do problema geral (BELLMAN, 2013). O método *branch and bound* realiza a busca estruturando as soluções de forma enumerada em uma árvore, onde os nós (sub-árvore) que não representam soluções aptas são descartados (NARENDRA; FUKUNAGA, 1977).

Conforme mencionado, métodos exatos podem demandar tempo exponencial para obter a solução ótima. Por isso, os métodos aproximados se tornam uma abordagem válida ao retornarem soluções factíveis para o problema (subótimas) em um tempo razoável. Dentro da classe de métodos aproximados, duas sub-classes de métodos se destacam: os algoritmos heurísticos (heurísticas específicas e meta-heurísticas) e os algoritmos aproximados (TALBI, 2009).

Os algoritmos heurísticos são aplicados na busca de uma solução ótima, podendo retornar uma solução sub-ótima, contudo, eles não garantem a qualidade da solução. Dessa forma, a solução obtida pode não ser factível para o problema. Por outro lado, os algoritmos aproximados garantem um limite de qualidade da solução para o problema. Em teoria, os algoritmos aproximados garantem que uma solução x não seja pior que α vezes a solução ótima, onde α pode ser um valor ou uma função do problema. Assim, torna-se possível definir qual é a qualidade mínima da solução a ser obtida (VAZIRANI, 2013).

O objetivo da aplicação de algoritmos aproximados é encontrar um limite de qualidade para o pior caso, auxiliando assim no desenvolvimento de heurísticas eficientes. Por serem aplicados em casos específicos, os algoritmos aproximados possuem limitações e, na prática, a garantia de qualidade fica muito distante da solução ótima, tornando-se inviável a aplicação (TALBI, 2009). Por essa razão, os algoritmos heurísticos possuem maior aplicabilidade em casos reais.

Os algoritmos heurísticos podem ser classificados em dois tipos: heurísticas específicas e meta-heurísticas. As heurísticas específicas, como o nome sugere, são métodos desenvolvidos para um problema em particular, enquanto as metaheurísticas são algoritmos de propósitos geral, capazes de serem empregadas praticamente a qualquer problema que trate de otimização (EIBEN; SMITH, 2003; TALBI, 2009). Tanto as heurísticas específicas quanto as metaheurísticas podem retornar soluções factíveis em um tempo razoável para problemas complexos.

As metaheurísticas se dividem em métodos baseados em solução única e métodos que utilizam população. Aqueles que consideram solução única partem de uma solução e realizam buscas na sua vizinhança a fim de obter melhorias. *Simulated Annealing*, *Tabu Search* e *Hill Climbing* são exemplos de métodos baseados em solução única. Os métodos que utilizam população buscam pela melhor solução partindo da combinação ou troca de informações em um conjunto de soluções. Podemos citar como exemplo os algoritmos evolutivos e de exame de partículas (TALBI, 2009). A Figura 3 ilustra a taxonomia apresentada para os métodos de otimização.

Figura 3 – Taxonomia dos métodos de otimização (adaptado de (TALBI, 2009)).

3.3 Metaheurísticas baseadas em solução única

Os métodos de busca local são classificados como métodos baseados em solução única e, dentre eles, destacam-se o algoritmo *Simulated Annealing* e *Tabu Search*. O algoritmo *Simulated Annealing* foi proposto por Kirkpatrick *et al.* (1983) e se trata de uma metaheurística inspirada na termodinâmica, considerando o princípio de aquecimento e resfriamento de estruturas. Através de um parâmetro denominado temperatura, é decidido qual o próximo vizinho que será avaliado e se aquela solução é apta ou não para o problema. Assim, quando o valor da temperatura é alto, maior a chance de ser aceitar uma solução vizinha pior que a atual ou não tão boa quanto a atual, emulando o comportamento mais instável da estrutura. Quando a temperatura diminui, soluções vizinhas piores terão chances reduzidas de serem exploradas, emulando um comportamento mais estável da estrutura. Logo, O *Simulated Annealing* é um algoritmo estocástico que explora soluções vizinhas não tão boas quanto a solução atual. Isso permite escapar de possíveis ótimos locais. Todavia, o método pode apresentar uma convergência lenta.

Proposto por Glover (1986), o algoritmo *Tabu Search* apresenta uma diferença na busca por soluções ao utilizar memória para armazenar soluções que já foram avaliadas. O objetivo de salvar as soluções anteriores é evitar explorar novamente uma mesma região do espaço de busca. Assim, evita-se retornar a soluções já visitadas ao não atualizar a solução corrente por uma anteriormente visitada. Para realizar isso, as soluções são armazenadas em uma lista chamada Lista Tabu. Ambos os algoritmos mencionados são utilizados neste trabalho. As suas implementações estão descritas com mais detalhes na Seção 5.5.

3.4 Algoritmos evolutivos

De maneira simplificada, um AE parte de uma dada população inicial, representada por um conjunto de possíveis soluções, e tenta evoluir esta população até convergir para um conjunto de soluções aceitáveis.

Os AEs são amplamente utilizados em problemas de otimização, principalmente aqueles problemas onde o tempo necessário para resolvê-los é intratável. Em outras palavras, pode-se dizer que um problema é intratável se o seu limite de tempo de complexidade é exponencial como $n!$ ou k^n , sendo k uma constante qualquer maior que 1 e n o número de critérios do problema, no contexto deste trabalho é o número instâncias de VMs (TOSCANI; VELOSO, 2008).

Na literatura, podem ser encontradas diversas classes de AEs, contudo, é possível descrever um AE em linhas gerais como apresentado em Algoritmo 1.

Algoritmo 1 – Algoritmos Evolutivos

```

1: procedimento AE(parâmetros do problema)
2:    $g \leftarrow 1$                                 ▷ inicializa um contador para as interações
3:   IniciaPopulação( $P_g$ )
4:   Avalia( $P_g$ )                                ▷ avalia o fitness para cada indivíduo da população
5:   repita
6:      $g \leftarrow g + 1$                             ▷ atualiza o contador de interações
7:      $P_g \leftarrow \text{Seleção}(P_{g-1})$                 ▷ seleciona indivíduos para o cruzamento
8:     Cruzamento( $P_g$ )                                ▷ Realiza o cruzamento
9:     Mutação( $P_g$ )                                ▷ Realiza mutação na nova população
10:    Avalia( $P_g$ )                                ▷ Avalia o fitness de cada indivíduo da nova população
11:  até Critério de parada ser satisfeito          ▷ verifica critério de parada
12:  retorna  $P_g$ 
13: fim procedimento
  
```

3.4.1 Motivação

A computação evolutiva surgiu com base no comportamento e evolução natural dos sistemas biológicos, ou seja, nos princípios fundamentais da Teoria da Evolução das Espécies proposta por Darwin (DARWIN, 1859). A computação evolutiva apresenta um bom desempenho na resolução de problemas reais. Segundo Eiben e Smith (2003), há duas principais motivações para a utilização dessa abordagem: (1) os métodos de solução desenvolvidos são inspirados em problemas e soluções encontrados na natureza; (2) os algoritmos evolutivos retornam uma boa solução (não necessariamente a ótima) com um tempo aceitável para problemas complexos. Logo, a computação evolutiva se inspira na forma como a natureza solucionou diversos problemas durante o seu processo evolutivo. Além disso, do ponto de vista técnico, a computação evolutiva apresenta algoritmos capazes de solucionar diferentes tipos de problemas complexos de forma robusta.

3.4.2 Terminologias Básicas

Tanto a computação bio-inspirada quanto a computação natural fundamentam suas técnicas computacionais em uma série de conceitos biológicos (TEUSCHER *et al.*, 2003) (BALLARD, 1999). Esses conceitos biológicos se concentram nas premissas de adaptação e

evolução genética, de forma que cria-se uma população de indivíduos que vão se reproduzindo e competindo pela sobrevivência.

Uma série de regras deve ser respeitada e alguns componentes devem ser levados em consideração para que essas técnicas computacionais tenham similaridade de fato com os fundamentos da evolução biológica.

Segundo [Rafiq, Mathews e Bullock \(2003\)](#), os principais componentes dos sistemas evolutivos são:

- **População:** é composta por um ou mais conjuntos de indivíduos que concorrem entre si por recursos limitados;
- **Variabilidade:** deve ser respeitada a variabilidade dos indivíduos em uma população, ou seja, não é benéfico para um sistema evolutivo possuir indivíduos homogêneos;
- **Hereditariedade:** apesar da variabilidade dos indivíduos ser respeitada, os indivíduos podem possuir traços ou características de outros indivíduos que o geraram;
- **Fitness:** é resultante de uma função que reflete o seu potencial para sobreviver e proliferar seus genes pela população.

A partir desses componentes, é possível simular o comportamento do processo evolutivo o qual se dará pela reprodução, mutação e seleção natural.

Segundo [Sait e Youssef \(1999\)](#), as principais terminologias utilizadas em AEs são:

- **Cromossomos:** são as características que compõem cada indivíduo, tais características tendem a variar de indivíduo para indivíduo. Os cromossomos possuem uma peculiaridade que diferencia um de outro: o **gene**. A representação de um cromossomo em um AEs é fundamental para obter êxito em sua execução, ou seja, deve ser realizado uma codificação adequada do problema. A codificação se trata de um mapeamento das principais características do problema como genes que compoem o cromossomo nos AE. Por exemplo, vários problemas na literatura são representados através de uma codificação binária, ou seja, os cromossomos são representados por genes com valores 0s e 1s. Logo, a solução de um problema é codificada como um indivíduo ou cromossomo em um AE. Nesse contexto, indivíduo e cromossomo podem apresentar o mesmo significado;
- **Operadores de Reprodução:** reprodução nada mais é que a combinação genética dos cromossomos de dois ou mais indivíduos (pais) a fim de gerar outros indivíduos (filhos), os quais serão novas possíveis soluções para o problema. O processo de reprodução pode ser compreendido também como recombinação genética ou *crossover*. Esse operadores são análogos aos encontrados na evolução natural da biologia;

- **Mutação:** também análogo à mutação da biologia, é uma alteração aleatória realizada em um cromossomo. A mutação garante a diversidade da população e pode levar à formação de indivíduos com melhor valor de *fitness*;
- **Seleção:** é uma das etapas dos AEs similar a seleção natural encontrada na teoria de Darwin. Devido à reprodução, a população tende a se renovar. Nesse contexto, a seleção controla a qualidade do processo de renovação dos indivíduos. Indivíduos mais aptos (melhor valor de *fitness*) apresentam maior chance de serem selecionados para reprodução e de permanecerem na população.

3.4.3 Operadores Genéticos

Como mencionado anteriormente, para que os AEs tenham comportamento similar à teoria da evolução de Darwin, é preciso respeitar uma série de regras. Nesse contexto, devem ser levados em consideração alguns operadores genéticos descritos nesta seção.

Dado o valor da função de *fitness* de um indivíduo, há diversas técnicas de seleção dos indivíduos para a aplicação dos operadores de reprodução como a seleção por *ranking*, torneio e roleta.

O método de seleção por *ranking*, apresentado por [Mitchell \(1998\)](#), evita a convergência prematura e a dominância de um indivíduo que possui o maior *fitness*. Para aplicar tal método, primeiro é necessário ordenar os indivíduos pelo seu *fitness* e, em seguida, um mapeamento com base nessa ordenação é feito. Esse mapeamento pode ser definido por um método de avaliação como o método linear que utiliza a Equação 3.1.

$$E(i,t) = Min + (Max - Min) * \frac{rank(i,t) - 1}{N - 1} \quad (3.1)$$

na qual,

Min é o indivíduo pior colocado no *rank*;

Max é o indivíduo com a melhor colocação no *rank*;

N é o número de indivíduos da população;

Rank(i,t) é a colocação que o indivíduo *i* obteve na iteração *t*.

O método de seleção por roleta faz a seleção de acordo com seu índice de aptidão. De forma análoga a uma roleta, cada fatia da roleta representa um indivíduo e essa fatia tem o tamanho proporcional ao índice de aptidão de cada indivíduo. Os indivíduos com maior índice de aptidão possuem mais chances de serem selecionados. A [Figura 4](#), apresenta uma abstração do esquema de seleção pelo método da roleta, onde se deve executar os seguintes passos para selecionar cada indivíduo:

- Uma variável *X* recebe a soma do *fitness* de todos os indivíduos;

Figura 4 – Método de seleção por roleta.

- Uma variável r recebe um valor aleatório entre 0 e X ;
- São percorridos todos os indivíduos acumulando em uma variável S o valor dos indivíduos já percorridos;
- Se S for maior ou igual a r , então é selecionado o indivíduo corrente, caso contrário, continua a percorrer os demais indivíduos.

O método de seleção por torneio é realizado em duas etapas. Na primeira, um número “ x ” de indivíduos (2 ou mais) da população é selecionado aleatoriamente para compor uma população temporária. Na segunda etapa, são escolhidos dois indivíduos desse grupo temporário para o cruzamento. O [Algoritmo 2](#) apresenta um exemplo básico desse método, no qual o torneio é realizado selecionando apenas 2 indivíduos da população.

Algoritmo 2 – Seleção por torneio

```

1:  $n \leftarrow 0$  ▷ contador para o laço
2: repita ▷ executa até que uma população intermediária seja preenchida
3:   Seleciona 2 indivíduos aleatoriamente
4:   se  $Individuo_1.fitness > Individuo_2.fitness$  então ▷ Competição entre os indivíduos
5:      $Individuo_1$  é escolhido para população intermediária
6:   senão
7:      $Individuo_2$  é escolhido para população intermediária
8:   fim se
9:    $n \leftarrow n + 1$ 
10: até  $n = \text{tamanho da população original}$ 

```

O cruzamento ou *crossover* seleciona dois indivíduos (pai 1 e pai 2) por algum método de seleção. A próxima etapa consiste em copiar partes do cromossomo do pai 1 e do pai 2 para o novo indivíduo gerado (indivíduo filho). Por meio do cruzamento, são gerados novos indivíduos

que herdam características de seus pais. O cruzamento pode ser realizado de várias maneiras, tais como:

- **Um-ponto:** um ponto de corte é definido aleatoriamente para os indivíduos pais. Uma parte do cromossomo do indivíduo filho será preenchida com os genes do início do cromossomo do pai 1 até o ponto escolhido. A outra parte do indivíduo filho será preenchida com os genes copiados do ponto escolhido até o final do cromossomo do pai 2;
- **Multi-ponto:** análogo a de um-ponto, nesse método, por meio de dois pontos ou mais, o cromossomo dos indivíduos pais são divididos em partes. No caso de dois pontos de corte, a primeira parte do pai 1 é copiado para o indivíduo filho, depois a segunda parte do pai 2 é copiada e, por fim, a terceira parte do pai 1 é copiada, formando assim um novo indivíduo;
- **Uniforme:** os genes do indivíduo filho são preenchidos aleatoriamente a partir dos pais, onde cada gene dos indivíduos pai tem 50% de chance de ser herdado pelo indivíduo filho;

A mutação em computação evolutiva é um operador que pode ser utilizado para garantir maior diversidade à população. Os indivíduos tem uma probabilidade de sofrer mutação, definida por uma *taxa de mutação*, de forma que novos indivíduos sejam introduzidos e, consequentemente, garantir a diversidade da população.

Uma taxa de mutação muito alta pode tornar lenta a convergência do algoritmo, e uma taxa de mutação baixa pode fazer o algoritmo cair em ótimos locais. Portanto, são necessários ajustes tanto na taxa de mutação quanto no tamanho da população, assim como na escolha do operador de mutação a ser aplicado.

Por exemplo, a mutação numa codificação binária pode alterar todos ou um único gene de um indivíduo, como ilustrado na Figura 5, onde, é realizado uma mutação de 100% e 10% em um indivíduo. Na mutação, quando é definido um percentual de mutação, significa que cada indivíduo tem essa chance de sofrer mutação, por exemplo, para 10% de taxa de mutação, cada indivíduo tem 10% de probabilidade de sofrer a mutação, no caso da Figura 5 apenas um indivíduo sofreu a mutação.

3.4.4 Representação Real

A representação binária é considerada a mais simples, muito utilizada nos trabalhos de Holland (1975), onde o genótipo é definido por um arranjo de 0s e 1s. Contudo, diversas aplicações do mundo real apresentam problemas com parâmetros contínuos, onde uma representação binária pode se tornar inviável (DEB *et al.*, 2000). Neste contexto, por exemplo, a decodificação de uma representação binária em uma representação real pode demandar o emprego de longas cadeias binárias (indivíduos) que tornam o processo evolutivo mais lento.

Figura 5 – Mutao com uma taxa de 10% e 100% em um indivduo.

Como alternativa, uma representao real pode ser utilizada na qual os genes so definidos por nmeros reais, distribudos em um intervalo contnuo no lugar de valores discretos (EIBEN; SMITH, 2003). Para facilitar a descrio desses operadores, representaremos os cromossomos pais por $P_1 = (P_{11}, P_{12}, P_{13}, \dots, P_{1n})$ e $P_2 = (P_{21}, P_{22}, P_{23}, \dots, P_{2n})$ e o cromossomo filho por $f = (f_1, f_2, f_3, \dots, f_n)$, onde, $P_1 \in \mathbb{R}^n$, $P_2 \in \mathbb{R}^n$ e $f \in \mathbb{R}^n$.

Diversos operadores podem ser utilizados em uma representao real, tais como:

- **Crossover mdia:** dado os cromossomos pais P_1 e P_2 o cromossomo filho produzido por $f = (P_1 + P_2)/2$;
- **Crossover mdia geomtrica:** dado os cromossomos pais P_1 e P_2 , cada gene do cromossomo filho (f_i) produzido por $f_i = \sqrt{P_{1i} * P_{2i}}$, $i = 1, 2, \dots, n$.
- **Crossover BLX-:** dado os cromossomos pais P_1 e P_2 , o cromossomo filho produzido por $f = P_1 + \beta(P_2 - P_1)$.
- **Crossover linear:** dado os cromossomos pais P_1 e P_2 , so obtidos trs filhos sendo $f_1 = 0,5P_1 + 0,5P_2$, $f_2 = 1,5P_1 - 0,5P_2$ e $f_3 = -0,5P_1 + 1,5P_2$. Desses trs filhos, apenas o que possuir melhor *fitness* ser escolhido e o restante ser descartado;

Assim como para o *crossover*, diversos operadores de mutao podem ser utilizados em uma representao real, tais como:

- **Mutao uniforme:** considerado o mais simples, apenas substitui um gene por um nmero aleatrio (respeitando o intervalo de domnio dos valores imposto pelo problema);
- **Mutao gaussiana:** a substituio de um gene por um nmero aleatrio seguindo uma distribuo normal;

- **Mutação *creep*:** esse operador adiciona ao gene mutado um número aleatório obtido de uma distribuição normal ou uniforme. Uma alternativa a isso seria multiplicar o gene por um número aleatório próximo de 1. Esse operador causa apenas uma pequena perturbação, ou seja, ele não é considerado "destrutivo";
- **Mutação por limite:** esse operador substitui o gene mutado por um dos limites do intervalo imposto pelo problema. Esse operador tende a levar os genes para o limite do intervalo de valores permitido, visando evitar a perda de diversidade;
- **Mutação não-uniforme:** é a substituição de um gene por um número extraído de uma distribuição não uniforme;
- **Mutação não-uniforme múltipla:** é a aplicação do operador anterior em todos os genes do cromossomo em que a mutação será realizada.

3.5 Problemas multi-objetivo

Há muitos problemas de otimização onde os objetivos podem ser conflitantes. Isso significa que pode não ser possível melhorar a solução para um dado objetivo sem que haja ônus em outro. Tais problemas são classificados como Problemas de Otimização Multi-objetivo (*Multiobjective Optimization Problem* (MOP)) (DEB *et al.*, 2000).

No contexto deste trabalho, é possível identificar dois objetivos conflitantes: Tempo e Custo (descritos na Seção 5.2). Na medida em que o objetivo se torna minimizar o custo, consequentemente, há um aumento no tempo. O mesmo ocorre no sentido inverso, pois quanto menor o valor investido na aquisição de uma infraestrutura, menor será o seu poder computacional. No gráfico da Figura 6 ilustramos esse conflito de objetivos.

Figura 6 – Relação dos objetivos conflitantes Tempo de Custo.

Em problemas caracterizados como monoobjetivo, é possível obter uma solução a partir de um determinado conjunto de soluções escolhendo a melhor entre elas, ou seja, a solução ótima. Por outro lado, em problemas multi-objetivos, não se pode garantir que há uma única solução melhor que as demais. Um problema de otimização multi-objetivo pode ser descrito como se segue:

$$\begin{aligned} \text{Minimizar} \quad & f(x) = [f_1(x), f_2(x), \dots, f_k(x)]^T \\ \text{Sujeito a} \quad & x \in \mathbb{R}^n \end{aligned} \quad (3.2)$$

onde k corresponde ao número de funções objetivos do problema considerando soluções factíveis $x \in \mathbb{R}^n$.

Em problemas mono-objetivos, para definir qual é a melhor solução, basta comparar os valores usando uma relação de menor ou igual em problemas de minimização. Usando essa comparação é possível obter várias soluções, mas apenas uma será selecionada $f_{\min} = \min\{f_i(x) | x \in \mathbb{R}^n\}$.

Em contraste, não é possível realizar uma comparação em um MOP de forma que as soluções fiquem ordenadas da melhor para a pior. Dessa forma, para que seja escolhida uma solução do conjunto de soluções, é aplicada a relação de dominância de Pareto proposta por [Edgeworth \(1881\)](#) e generalizada por [Pareto \(1964\)](#).

Definição¹. A relação de dominância de Pareto define que uma solução x^1 é dominante sobre x^2 (notado como $x^1 \prec_{\text{pareto}} x^2$), se e somente se ([BUI; ALAM, 2008](#)):

$$\begin{aligned} & \forall i \in \{1, \dots, k\} : f(x)^1 \leq f(x)^2 \\ & e \\ & \exists i \in \{1, \dots, k\} : f(x)^1 < f(x)^2 \end{aligned} \quad (3.3)$$

A dominância de Pareto é ilustrada na Figura 7 com quatro pontos para dois fatores. A solução x^3 possui valor menor que x^2 em ambos fatores, portanto diz-se que $x^3 \prec_{\text{pareto}} x^2$, também é possível afirmar que x^3 é dominante sobre x^1 e x^4 , apesar de possuírem um dos fatores com o mesmo valor, o outro fator de x^3 é menor. Contudo, x^1 e x^4 não são comparáveis entre si, sendo assim, $x^1 \not\prec_{\text{pareto}} x^4$ e $x^4 \not\prec_{\text{pareto}} x^1$. Na Figura 7 pode-se afirmar que x^3 é não dominada, caso x^3 não existisse, seria possível afirmar que x^1 e x^4 seriam soluções não dominadas.

Definição. Uma solução $x^* \in \mathbb{X}$ é definida como Pareto ótimo se não existir nenhuma outra solução $x \in \mathbb{X}$ que seja $f(x) \prec_{\text{pareto}} f(x^*)$ ([BUI; ALAM, 2008](#)).

Na Figura 8, é ilustrado o conceito de Fronteira de Pareto, onde, os pontos vermelhos representam os conjuntos de soluções e os pontos azuis, são todas as soluções não dominadas, ou seja, as soluções que compoem a fronteira de Pareto.

¹ Assumindo apenas problemas de minimização

Figura 7 – Representação do conceito da relação de Dominância de Pareto.

Figura 8 – Ilustração do conjunto de soluções e da Fronteira de Pareto (BUI; ALAM, 2008).

Definição. Para um conjunto Pareto ótimo $\mathbb{P}$, a Fronteira de Pareto (PF) é definida como (BUI; ALAM, 2008):

$$\mathbb{PF}^* = \{f(x) = (f_1(x), \dots, f_k(x)) | x \in \mathbb{P}^*\} \quad (3.4)$$

Na literatura é possível encontrar diversas abordagens que tratam de MOP. As chamadas

Multi-Criteria Decision Making (MCDM) são abordagens caracterizadas pelo uso de técnicas de programação matemáticas, enquanto *Evolutionary Multiobjective Optimization* (EMO) utilizam técnicas baseadas na computação evolutiva para retornar o Pareto Ótimo aproximado (BUI; ALAM, 2008). Os EMOs são amplamente utilizados no caso de problemas em que é impraticável obter o Pareto Ótimo. Neste caso eles retornam um Pareto Ótimo aproximado, o que se aplica no contexto deste trabalho.

3.6 Considerações Finais

Este capítulo apresentou as terminologias básicas relacionadas a computação evolutiva, os fundamentos biológicos, os principais operadores genéticos e as codificações para os dados trabalhados nos algoritmos evolutivos, além do conceito de problemas multi-objetivo. Neste projeto de mestrado foram investigados os algoritmos de otimização disponíveis na literatura nesse contexto. Por fim, estes foram aplicados no escopo do problema abordado neste projeto de mestrado.

REVISÃO BIBLIOGRÁFICA

4.1 Considerações iniciais

Neste capítulo será abordada uma revisão da literatura, na qual foram identificados diversos trabalhos dentro do contexto do problema abordado. Alguns dos trabalhos descritos nesta Seção foram extraídos de uma revisão sistemática descrita em [Faniyi e Bahsoon \(2015\)](#), onde é realizado um estudo da literatura sobre gerenciamento a nível de serviço em nuvem utilizando o termo de busca (“*Cloud computing*” AND “*SLA*” AND “*Resource allocation*”). Os demais trabalhos foram identificados por meio de bibliotecas digitais como: IEEE Xplore, ACM Digital Library, SCOPUS e Google Scholar, utilizando o mesmo termo com o adicional de (“*Optimization*”). Tanto na revisão realizada por [Faniyi e Bahsoon \(2015\)](#), quanto no levantamento realizado neste trabalho, foram considerados trabalhos publicados em periódicos, conferências e *workshops*.

4.2 Trabalhos relacionados no contexto geral de computação em nuvem

A nuvem é um ambiente altamente escalável, no qual a demanda de serviços pode mudar instantaneamente. Dessa forma, a alocação automática de recursos para atender essa demanda se torna um tópico interessante tanto no âmbito acadêmico quanto no industrial. O correto provisionamento permite um melhor uso dos recursos computacionais disponíveis e, consequentemente, de toda a infraestrutura que compõe a nuvem, pois o mapeamento entre a carga e os recursos é mais eficiente. Além disso, isso ajuda no cumprimento das exigências feitas pelos clientes e provê um grande dinamismo ao sistema ([BATISTA et al., 2015](#)).

A tarefa de prover mais recursos computacionais a um cliente, por exemplo, é altamente viável e de fácil disponibilização, uma vez que os recursos computacionais são virtualizados e

podem ser considerados ilimitados na visão dos clientes. No entanto, é importante lembrar que a alocação de mais recursos computacionais influencia no custo final que é repassado ao cliente e requer mecanismos eficientes por parte do provedor (COUTINHO *et al.*, 2015).

Há na literatura diversos trabalhos que analisam e propõem mecanismos para o gerenciamento de recursos em um ambiente em nuvem. A proposta da Amazon para a reconfiguração automática da infraestrutura de seus clientes é baseada em monitoramento, por meio de alertas (*CloudWatch Alarms*) e políticas (*Scaling Policies*). O escalonamento da AWS (*Amazon Web Services*) pode ser feito baseado em métricas que são geralmente baseadas no consumo de CPU (AMAZON, 2015b) ou em políticas que são basicamente divididas em políticas manuais (*Manual Scaling*), políticas dinâmicas (*Dynamic Scaling*) e políticas agendadas (*Scheduled Scaling*) (AMAZON, 2015c).

Em (SONI; KALRA, 2014), é proposto um balanceador de carga central para um ambiente de computação em nuvem de grande escala. Todas as requisições chegam a um centro de controle, no qual o balanceador de carga central é conectado a todos os usuários e a todas as máquinas virtuais presentes nessa nuvem. A prioridade para cada máquina virtual é calculada com base na velocidade da CPU (MIPS) e memória. Com esse escalonador centralizado, foi possível obter um escalonamento aceitável. Porém, diagnosticou-se a necessidade da implementação de outros algoritmos para distribuir a carga de acordo com a utilização dos recursos, de modo a obter uma distribuição de carga mais dinâmica e robusta.

Os autores em (KOUSIOURIS; CUCINOTTA; VARVARIGOU, 2011) investigam diferentes tipos de carga de trabalho para as VMs, a fim de encontrar melhores estratégias de alocação de recursos. Para tal, é levado em consideração um número de parâmetros (críticos) sobre o desempenho das VMs, os quais incluem percentuais de alocação, decisões de escalonamento em tempo real e realocação de VMs. Além disso, é inserido um método caixa preta baseado em redes neurais, o qual é comparado com um método de regressão linear. A aplicação de redes neurais para a alocação de recursos demonstrou-se eficiente, apresentando uma margem de erro inferior a 5%.

Dessa forma, o provedor pode ter conhecimento prévio de possíveis interferências na carga de trabalho e, consequentemente, otimizar a alocação de recursos. Por outro lado, são necessárias algumas alterações para que haja a detecção automática do tipo de carga de trabalho gerada por cada aplicação. Além disso, os testes deveriam ser adequados para serem executados em aplicações reais e para vários clientes.

Zhao *et al.* (2011) propõem um modelo de escalonamento de recursos virtuais baseado na seleção por meio de algoritmos genéticos multi-objetivo (NSGA II - *Non-dominated Sorting Genetic Algorithm*). Esse modelo foi avaliado pelo equilíbrio da carga distribuída, recursos físicos e virtuais com a análise de escalonamento dos recursos virtuais. Os autores compararam esse modelo com algoritmos de geração aleatório, algoritmos estáticos e algoritmos de ranqueamento. Com base nos testes executados, esse trabalho mostrou que o NSGA II pode ser utilizado

para resolver o problema de escalonamento de recursos virtuais. Nos experimentos nos quais os algoritmos são comparados, o NSGA II demonstrou-se mais apropriado para esse tipo de aplicação. Contudo, esse modelo não leva em consideração as falhas encontradas em um ambiente real, as necessidades do cliente em contraste com o SLA e os recursos disponíveis, bem como o custo cobrado do cliente.

Wu, Garg e Buyya (2012) propõem um algoritmo de controle de admissão e escalonamento para que os provedores de SaaS possam utilizar de maneira eficaz os recursos e maximizar seus lucros, além de reduzir os custos e melhorar a satisfação dos clientes. Nesse trabalho, foram apresentadas quatro estratégias para avaliar se uma requisição pode ou não ser atendida de maneira a garantir a QoS estipulada. Na fase de avaliação, os desempenhos dos algoritmos de cada estratégia foram comparados com outros dois de referência na literatura: o *MinResTime* e o *StaticGreedy*. Nas simulações executadas, o algoritmo proposto por esse trabalho conseguiu obter uma redução do custo de até 40% levando em consideração todas as faixas de variação de parâmetros de QoS estipuladas. Apesar dos resultados promissores, diagnosticou-se a necessidade de melhorar a robustez dos algoritmos a fim de tratar erros dinamicamente, bem como a necessidade de considerar negociações do SLA para melhorar o desempenho.

4.3 Trabalhos relacionados no contexto de computação em nuvem que abordam SLA

Em um trabalho prévio, Batista *et al.* (2015) elaboraram uma proposta de análise de desempenho no gerenciamento de recursos em computação em nuvem, onde foi implementado um módulo de reconfiguração que tinha como objetivo reduzir o número de violações no SLA. Contudo, constatou-se a necessidade de técnicas de otimização, pois em alguns casos o módulo de reconfiguração não conseguia manter a QoS contratado entre o provedor e cliente. Esse trabalho motivou a proposta desta dissertação.

A maioria dos trabalhos abordam a fase de monitoramento do SLA, neste trabalho também é abordado a fase de estabelecimento do SLA e até o momento não encontramos trabalhos na literatura que abordem o problema de tal forma. Há ainda na literatura, outros trabalhos abordando o provisionamento de recursos que podem ser classificados da seguinte forma: políticas dinâmicas, baseados em heurísticas, multi-critério e otimização.

As políticas dinâmicas são as mais interessantes para este trabalho. A Amazon, por exemplo, permite ao usuário configurar o tipo de escalonamento de recursos baseado no ajuste do número de instâncias que devem ser alocadas ou desalocadas que seguem uma quantidade ou uma porcentagem definida pelo usuário (AMAZON, 2015a). Dessa forma, fica a cargo do usuário tentar prever qual seria a melhor configuração para um escalonamento eficiente para a sua infraestrutura hospedada na nuvem do provedor. Contudo, nenhuma configuração eficiente é feita para que seja garantida a QoS da infraestrutura.

A abordagem baseada em políticas é a mais simples para gerenciamento de recursos. Ela consiste na tomada de decisão com base em uma condição, sendo utilizada quando há um número limitado de cenários para modificar. Dessa forma, esta é uma abordagem bem limitada, sendo inviável para aplicação no contexto de aplicações em nuvem, devido à alta complexidade e aos diversos parâmetros envolvidos (HUEBSCHER; MCCANN, 2008) (YUCHAO; BO; FUYANG, 2012).

Abordagens baseadas em heurísticas partem do princípio de que um conjunto de heurísticas são previamente definidas para serem aplicadas em alguns cenários. Essas estratégias são relativamente simples, nas quais, diversas heurísticas podem ser desenvolvidas e adicionadas em tempo de execução. Contudo, isso fica limitado à previsão de cenários específicos em um determinado tempo. Dessa forma, pode ocorrer violação do SLA em cenários que não estão inclusos nessa previsão. Na literatura foram encontrados diversos trabalhos que aplicam heurísticas para o problema de provisionamento de recursos. Alguns aplicam heurísticas para a inicialização automática de VMs caso as VMs utilizadas atinjam um percentual alto de utilização (EMEAKAROHA *et al.*, 2011). Outro trabalho, conduzido por Kessaci, Melab e Talbi (2014), utiliza uma abordagem com meta-heurísticas para redução da utilização dos recursos para economia de energia. Contudo, esses trabalhos não tomam decisão de forma otimizada e consideram um número bem limitado de SLAs e de clientes.

No contexto de heurísticas, o trabalho da literatura que mais se aproxima do escopo deste projeto de mestrado é o Batista *et al.* (2015). Os autores propuseram um módulo denominado *Resource Management Module* (ReMM). O ReMM procura satisfazer a QoS para o lado do cliente e ao mesmo tempo prover o melhor uso possível dos recursos. Para isso, é aplicado uma heurística baseada nos modelos de negócio atuais. Contudo, constatou-se a necessidade de técnicas de otimização para esse módulo, pois em alguns casos o módulo de reconfiguração não conseguia manter a QoS contratada.

As técnicas baseadas em otimização utilizam abordagens similares aos métodos heurísticos, sendo aplicada na detecção e tratamento da violação do SLA. Neste contexto, a detecção pode ocorrer através da análise do modelo de desempenho do sistema ou ocorrência de falhas, com a finalidade de reajustar a capacidade das VMs contratadas. Para que ocorra a detecção, abordagens de otimização geralmente utilizam métodos de *machine learning*, análise de séries temporais, técnicas de tolerância a falhas, entre outros (EYRAUD-DUBOIS; LARCHEVÊQUE, 2013) (JIANG *et al.*, 2013) (WANG; DU; CHEN, 2012). Contudo, esses métodos sofrem com o problema da complexidade gerada para dar uma resposta e, como consequência, demoram para resolver problemas específicos. Portanto, a aplicação dessa abordagem em tempo de execução torna-se um desafio.

Finalmente, em multi-critério, como o nome sugere, é uma abordagem que permite a análise de múltiplos critérios ou situações em um problema (HO; XU; DEY, 2010). Soluções baseadas em multi-critério para o problema de provisionamento de recursos tendem a ser

descentralizadas, ou seja, avaliando cada critério ou situação de forma independente (YAZIR *et al.*, 2010) (YAZIR *et al.*, 2012). Essa abordagem sofre do mesmo problema da otimização devido a sua complexidade.

Há diversos problemas complexos dentro do contexto de computação em nuvem que são abordados por soluções que utilizam otimização. Por exemplo, os problemas de alocação de máquinas virtuais em máquinas reais (MASDARI; NABAVI; AHMADI, 2016), economia de energia (BELOGLAZOV; ABAWAJY; BUYYA, 2012), escalonamento e balanceamento de carga das aplicações em máquinas virtuais (L.D.; KRISHNA, 2013) e utilização de recursos visando minimizar o custo e ainda garantir desempenho satisfatório (BYUN *et al.*, 2011) (BATISTA *et al.*, 2015), de forma a satisfazer o SLA e o uso eficiente dos recursos. Porém, os trabalhos não tomam decisão de forma otimizada, aplica-se somente a uma quantidade limitada de SLAs (entre 3 a 15) e em sua maioria apenas um cliente é executado por experimento.

Todos esses problemas exigem soluções ótimas ou sub-ótimas que podem ser alcançadas por técnicas conceituadas dentro da área de otimização. Este projeto de mestrado procura resolver o problema de provisionamento de recursos aplicando meta-heurísticas. Os métodos propostos visam solucionar o problema de alocação de recursos de forma adequada, utilizando abordagens baseadas em meta-heurística para alcançar resultados que garantam o SLA, o auto gerenciamento e a QoS para os clientes de um provedor em nuvem.

4.4 Principais características dos trabalhos relacionados

A Tabela 4 apresenta as principais características dos trabalhos analisados, levando em consideração o SLA, o auto-gerenciamento do mecanismo proposto, a heurística utilizada e a garantia de QoS.

Tabela 4 – Contraste com os trabalhos relacionados

Trabalhos Relacionados	Ambiente de execução	SLA	Auto Gerenciamento	Técnica de otimização	QoS	Fase
[Provedores, 2018]	Real	–	✓	–	–	Monitoramento
Soni e Kalra (2014)	Simulator	–	✓	–	–	Monitoramento
Kousiouris, Cucinotta e Varvarigou (2011)	Simulator	–	✓	✓	✓	Monitoramento
Zhao <i>et al.</i> (2011)	Simulator	–	✓	✓	–	Monitoramento
Wu, Garg e Buyya (2012)	Simulator	✓	–	–	✓	Monitoramento
Emekaroha <i>et al.</i> (2011)	Simulador	✓	✓	–	–	Monitoramento
Kessaci, Melab e Talbi (2014)	Simulador	✓	–	✓	–	Monitoramento
Eyraud-Dubois e Larchevêque (2013)	Simulador	✓	–	✓	–	Monitoramento
Yazir <i>et al.</i> (2012)	Simulador	✓	–	✓	–	Monitoramento
Batista <i>et al.</i> (2015)	Simulador	✓	✓	–	✓	Monitoramento
Mecanismo Proposto	Simulador	✓	✓	✓	✓	Todas

4.5 Considerações finais

Neste Capítulo foram apresentados trabalhos da literatura e que estão relacionados a proposta deste projeto. Além disso foi possível relacionar as propostas existentes e enfatizar as contribuições que este trabalho busca contemplar. O próximo Capítulo apresenta a proposta com base na modelagem do problema e no desenvolvimento deste projeto de mestrado.

DESENVOLVIMENTO DA ABORDAGEM PROPOSTA

5.1 Considerações Iniciais

Neste capítulo será discutido o desenvolvimento da solução proposta nesta dissertação de mestrado, bem como os desafios encontrados no processo de desenvolvimento.

Para o contexto deste trabalho, os algoritmos consideram informações relacionadas à demanda dos serviços, onde a carga aplicada sobre os recursos do provedor é dinâmica. Dessa forma, a análise proposta visa proporcionar um planejamento de capacidade e um provisionamento de recursos eficiente, em que pode ser verificado quais são os melhores algoritmos para satisfazer clientes e provedores.

5.2 Descrição do problema

Provedores como Amazon EC2 e Microsoft Azure utilizam uma metodologia de provisionamento de recursos na qual os clientes são responsáveis por estimar com precisão a quantidade de recursos necessários e selecionar a instância a ser contratada (HUU; MONTAGNAT, 2010). Desse modo, os próprios usuários são responsáveis por elaborar um planejamento de capacidade de acordo com suas necessidades.

Importante lembrar que nem sempre o usuário possui conhecimentos técnicos para fazer um gerenciamento manual do provisionamento de recursos. Dessa forma, a aplicação de alguma técnica de otimização para o provisionamento de recurso tornaria tal tarefa automatizada, melhoraria a utilização dos recursos computacionais e conseqüentemente, o usuário não pagaria além do que realmente necessita, ou seja, pagaria um preço justo pelo serviço contratado.

Como citado anteriormente, diversos atributos de QoS podem ser tratados. Neste trabalho

em específico, trabalharemos com quatro delas, as quais identificamos com base na literatura, serem as mais relevantes em um contexto geral (FANIYI; BAHSOON, 2015): capacidade, tempo de resposta, disponibilidade e custo por hora, os quais são apresentado a seguir:

- **Capacidade (C):** número de máquinas virtuais contratadas pelo cliente. A princípio tem-se três tipos de VMs, sendo elas, Small, Medium e Large. Tais instâncias foram baseadas nas configurações das instancias m3.medium, m3.large, m3.xlarge e m3.2xlarge da Amazon EC2¹. Uma infraestrutura de um *data center* pode possuir inúmeras VMs com diversas configurações diferentes. Para obter a capacidade foi elaborada a equação (5.1) baseada no projeto de Doutorado de Luis H. V. Nakamura (NAKAMURA, 2017), na qual n corresponde ao número de máquinas virtuais:

$$C = \sum_{i=1}^{i=n} Capacidade(VM_i) \quad (5.1)$$

- **Tempo de Resposta (T):** tempo de resposta da aplicação esperado pelo cliente, que é obtido pela execução de sua aplicação na infraestrutura contratada. O tempo de resposta pode ser obtido pela equação (5.2) (KO; KIM; KWON, 2008), na qual n corresponde ao número de máquinas virtuais:

$$T = \frac{\sum_{i=1}^{i=n} TempoDeResposta(VM_i)}{n} \quad (5.2)$$

- **Disponibilidade (D):** consiste na infraestrutura estar presente e pronta para a utilização. Nesse contexto, a disponibilidade pode ser obtida de duas formas, (1) inicialmente, pode ser calculada pelo número de máquinas em razão da probabilidade de ter erro, ou seja, $1/(\text{número de VMs})$, nesse caso se fossem instanciadas quatro VMs, a probabilidade de ocorrer erro seria de $1/4$ (25%) e a disponibilidade seria de 75%, ou ainda, (2) cria-se um contador para saber qual o percentual de requisições atendidas com sucesso e qual o percentual que falhou. O percentual de requisições atendidas com sucesso será o percentual de disponibilidade. Neste trabalho a disponibilidade será calculada das duas maneiras, inicialmente pelo método (1) e durante o monitoramento por meio do método (2). Para o segundo método, a disponibilidade pode ser obtida pela equação (5.3) (KO; KIM; KWON, 2008), na qual n corresponde ao número de máquinas virtuais:

$$D = \prod_{i=1}^{i=n} Disponibilidade(VM_i) \quad (5.3)$$

- **Custo por hora (C/h):** valor monetário estipulado no SLA, referente a quanto o cliente irá pagar por hora pelo serviço (utilização da VM). O custo monetário por hora pode ser

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

obtido pela fórmula (5.4) (KO; KIM; KWON, 2008), na qual n corresponde ao número de máquinas virtuais:

$$C/h = \sum_{i=1}^{i=n} \text{Custo}(VM_i) \quad (5.4)$$

Com essas variáveis pode-se estabelecer um modelo ou um sistema de identificação do problema (EIBEN; SMITH, 2003), o qual tem-se duas entradas e três saídas, como apresentado na Figura 9.

Figura 9 – Modelo ou sistema de identificação do problema.

Problemas dessa natureza, como escalonamento de tarefas e provisionamento de recursos, são considerados problemas da classe NP-difícil (KUMBHARE; SIMMHAN; PRASANNA, 2013). Muitos problemas da classe NP-difícil são resolvidos através da programação inteira e *branch-and-bound* (WOEGINGER, 2003). Porém, não são adequados pra resolver problemas de decisão que possuem adaptação contínua (KUMBHARE *et al.*, 2015). Neste contexto, optamos por investigar algoritmos de otimização encontrados na computação evolutiva, pois apresentam um bom desempenho na resolução de problemas reais.

Outro ponto, partindo da análise dos trabalhos relacionados e do contexto prático em ambientes empresariais, foi identificado que o gerenciamento do ciclo de vida do SLA pode possuir um fluxo adicional ao que foi descrito por Faniyi e Bahsoon (2015) na Seção 2.4.3. Este fluxo é apresentado na Figura 10.

Neste trabalho é tratado de forma automatizada e otimizada todo o ciclo do SLA, com foco principal nas fases de tomada de decisão, que são as fases de 1 a 4 abordadas no final da Seção 2.4.3.

Figura 10 – Ciclo de vida identificado do SLA.

5.3 Framework de otimização

Os algoritmos desenvolvidos nesta dissertação de mestrado consideram a manipulação dos recursos em tempo de execução buscando otimizar o uso eficiente da infraestrutura do provedor da nuvem a fim de melhor atender diversos tipos de requisições dos clientes e também garantir as métricas de QoS definidas no contrato SLA. Na Figura 11 é apresentado como é o gerenciamento de carga de uma arquitetura em nuvem padrão (disponível em um ambiente empresarial e na literatura) e, na Figura 12 é representado como será o gerenciamento de carga e provisionamento de recursos após a implementação dos algoritmos de otimização, todo o sistema foi modelado dentro do simulador Cloud Sim, possuindo como componentes:

- **Framework de algoritmos de otimização:** o *framework* é a proposta desenvolvida neste projeto, nele estão contidos os algoritmos de otimização e as políticas de tomada de decisões. O *framework* é composto por:
 - *Algoritmos de balanceamento:* os algoritmos de balanceamento são responsáveis pela atribuição das requisições para serem executadas nos recursos disponíveis. Neste projeto, foi utilizado o algoritmo *Round Robin*, uma vez que o foco deste trabalho é o provisionamento de recursos. Novas abordagens de balanceamento de carga serão exploradas em trabalhos futuros;
 - *Algoritmos de alocação de recursos:* ou ainda, algoritmos para o provisionamento de recursos. Neste componente foram implementados os algoritmos descritos na Seção 5.5, os quais tem a finalidade de encontrar um novo conjunto de VMs que melhor se adequam às necessidades do cliente;
 - *Modelo de negócios (SLA):* define o modelo de SLA e o modelo de negociação aplicado no contexto da aplicação, tais modelos foram definidos na Seção 2.3 e 5.2;
 - *Planejamento de capacidade:* é importante salientar que os recursos em nuvem, apesar da grande oferta, são finitos. Portanto, é necessário um planejamento de capacidade e de tomada de decisão para tratar os pontos nos quais a demanda de serviços

excederá o limite de recursos disponíveis. Neste componente são determinados as políticas de tomada de decisão para quando a demanda estiver próxima de exceder os limites dos recursos da infraestrutura.

- **Escopo de outros projetos:** a proposta desenvolvida nesta dissertação vai de encontro com outras propostas do grupo de pesquisa em que se encontra, portanto, o *framework* aqui desenvolvido compreende-se como um módulo para outras arquiteturas (descritas na Seção 5.4.1). Essas arquiteturas possuem alguns componentes em comum, utilizados pelo *framework*, tais como:
 - *Módulo de balanceamento de carga:* responsável pela distribuição da carga de trabalho executada nas VMs, pode haver necessidade de alteração na forma como a carga é distribuída, neste contexto é utilizado as decisões tomadas pelos *Algoritmos de balanceamento*, este módulo foi desenvolvido pelo trabalho do Batista (2016);
 - *Módulo de reconfiguração:* ao renegociar um SLA, possivelmente um novo conjunto de VMs será atribuído para o cliente. A fim de cumprir o contrato, os módulos de reconfiguração são responsáveis por atribuir o novo conjunto de VMs determinado pelos algoritmos de otimização, este módulo foi desenvolvido nos trabalhos de Batista (2016) (ReMM) e Nakamura (2017) (ReMan);
 - *Módulo de monitoração:* é responsável por coletar as informações em tempo de execução. É nesse componente que são emitidos os alertas de violação, este módulo foi desenvolvido nos trabalhos de Batista (2016) e Nakamura (2017);
 - *Dados da ontologia:* um dos projetos ao qual este trabalho está relacionado, propõe algumas ontologias para auxiliar no modelo de negócios. Contudo, essas ontologias não foram aplicadas neste contexto. Foi identificado a necessidade de melhor elaboração das ontologias para tal aplicação, este módulo foi desenvolvido nos trabalhos de Nakamura (2017).
- **Observações:** os componentes desenvolvidos neste trabalho são flexíveis e fracamente acoplados, de modo que as metodologias e políticas aplicadas em um determinado componente, podem ser substituídas por outras sem a necessidade de alterações nos demais.

De forma mais abstrata, através da Figura 13 é possível ter uma melhor compreensão do comportamento da arquitetura e a ativação dos módulos de otimização, implementados nas Fases 1 e 2. Na Figura 13 são apresentados quatro estágios de execução, sendo elas:

- **Fase 1:** esta fase contempla o processo de negociação e fechamento do SLA, a qual é composta pelo módulo de estabelecimento do SLA desenvolvido neste projeto e descrita na Seção 5.4. Tal módulo é composto pelos algoritmos de otimização;

- **Fase 2:** após o SLA ser estabelecido e o contrato fechado, a infraestrutura contratada já estará disponível para o cliente. Nesta fase é executado o monitoramento, no qual é verificado se o SLA está sendo executado sem violações. Caso ocorra um determinado

Figura 11 – Representação do gerenciamento de carga de uma arquitetura padrão em nuvem.

Figura 12 – Representação do gerenciamento de carga de uma arquitetura em nuvem com os algoritmos de otimização.

número de violações por um determinado tempo, uma renegociação é realizada para que não ocorram mais violações. Os critérios para a tomada de decisão, se haverá ou não renegociação, são definidos de acordo com a política estabelecida;

- **Fase 3:** esta fase é acionada de acordo com as políticas estabelecidas na Fase 2. Caso seja necessária uma renegociação, os algoritmos de otimização serão executados para encontrar um novo SLA que adere as necessidades do cliente;
- **Fase 4:** essa fase final contempla o fechamento do SLA, na qual os recursos são desalocados e disponibilizados novamente para o *datacenter*. Após finalizar, é emitido um relatório para o cliente com os dados coletados na Fase 2;
- **Capacidade total da infraestrutura:** é importante que esteja bem claro o fato de que os recursos não são ilimitados. Dessa forma, é feito um controle sobre a quantidade de recursos disponíveis, e sobre quando eles são alocados e desalocados para os clientes. Pode acontecer uma situação em que o cliente deseja contratar os recursos da infraestrutura e eles não estarem disponíveis. Neste caso ele aguarda em uma fila.

Figura 13 – Comportamento da arquitetura e ativação dos módulos de otimização.

Exemplo 1. O atendimento aos clientes é realizado sob demanda, basicamente em 6 passos. 1) O cliente notifica o provedor que deseja contratar recursos da infraestrutura; 2) o provedor recebe a proposta do cliente e verifica se pode atender, e retorna um possível SLA de forma que possa atender de melhor forma (o mais próximo do que o cliente quer) possível as necessidades do cliente; 3) após essa negociação um SLA é estabelecido e a infraestrutura descrita no SLA é disponibilizada para o cliente pelo período determinado; 4) então é executado um monitoramento durante a utilização dos recursos a fim de verificar se há violações no contrato; 5) dependendo da política estabelecida, se houverem mais de “X” violações em um determinado espaço de tempo, é realizada uma renegociação do SLA, a fim de atender melhor as novas necessidades do cliente; 6) ao término do tempo contratado pelo cliente, é gerado um relatório das informações coletadas

durante o monitoramento e disponibilizado para o cliente, a fim de identificar os motivos das violações e penalidades se for o caso. Todos esses passos são executados em paralelo para todos os clientes: enquanto um cliente está no passo 6, outro pode estar no 2 e outro pode estar no 4. Para tratar essa condição de corrida é implementado um *mutex* na “Capacidade total da infraestrutura” e, o atendimento só é interrompido caso não haja mais recursos disponíveis na infraestrutura. Dessa forma, é notificado aos clientes o motivo pelo qual não são atendidos, em seguida são enfileirados e aguardam até que os recursos sejam liberados.

As chamadas dos algoritmos de otimização para definir o SLA, são realizadas nas fases 1 e 3 apresentadas na Figura 13. Tais algoritmos foram implementados em dois módulos distintos descritos na Seção 5.4.

5.4 Módulos de gerenciamento do SLA

Neste projeto foram desenvolvidos dois módulos, o módulo de estabelecimento de SLA e o módulo de monitoramento do SLA. O primeiro módulo é acionado no estágio inicial do processo de negociação do SLA, antes da execução da aplicação do cliente. Sendo assim, os algoritmos de otimização são aplicados para encontrar os recursos que devem ser alocados para determinado cliente, de forma que não haja violações no SLA. Contudo, até o presente momento, não foram encontrados exemplos ou instâncias de SLAs para execução de testes. Portanto, outra contribuição deste projeto de mestrado foi o desenvolvimento de um gerador de SLAs que será apresentado nesta Seção.

Este primeiro módulo contempla as fases 1 e 2 descritas na Seção 2.4.3. Neste contexto, o módulo de otimização para estabelecimento do SLA deve ser executado nas fases iniciais do ciclo de vida do acordo, a negociação e a implantação. Para essa primeira fase da otimização, o objetivo é que o cliente especifique a aplicação que deseja executar, o tempo de resposta desejável dessa aplicação, qual disponibilidade mínima aceita, e quanto está disposto a pagar. Dessa forma, é possível identificar o perfil do cliente e oferecer um SLA que atenda suas exigências.

Considerando os trabalhos disponíveis na literatura que abordam o cumprimento do SLA, identificou-se que todos utilizam instâncias privadas, ou seja, geram os SLAs de acordo com os atributos de QoS considerados no contexto do estudo. Além disso, esses estudos utilizam um número pequeno de SLAs para as execuções de experimentos, em média de 3 a 15 SLAs por estudo. Partindo desse levantamento, identificou-se a necessidade do desenvolvimento de um gerador de SLAs para a avaliação dos módulos desenvolvidos neste trabalho.

Neste contexto, o SLA é tratado a princípio com quatro atributos de QoS, os mesmos atributos descritos na Seção 5.2. Optou-se por utilizar esses atributos por serem os mais comuns utilizados na literatura, contudo, o gerador é flexível para que sejam adicionados outros fatores.

Com esses atributos de QoS, foi estimado um valor mínimo e máximo para cada um

deles. A partir desses valores foi realizada uma distribuição discreta entre eles, para que fossem geradas as combinações dos atributos e gerados os SLAs.

Para gerar os SLAs, o simulador CloudSim foi configurado para executar com três tipos de instâncias de VMs, sendo elas baseadas nas instâncias **m3.medium**, **m3.large** e **m3.xlarge** da Amazon EC2². Para a aplicação do cliente foram simulados dois serviços, o primeiro simulando um repositório de arquivos (*Apache Benchmark* (APACHE, 2017)), e o segundo uma aplicação de renderização de imagens (*Smallpt Benchmark* (SUITE, 2017)), nos quais foi gerada uma carga de trabalho mínima em que a instância com menor poder computacional conseguisse processar, e o valor máximo ficou dinâmico, variando de acordo com a quantidade de SLAs gerados. Além disso, foi estipulado um intervalo para o tempo de resposta da aplicação, custo por hora, disponibilidade e capacidade.

Os intervalos seguiram uma ordem discreta para todos os atributos, esses valores foram combinados entre si para dar origem aos SLAs. No algoritmo 3 é descrito como foi realizado a implementação e na Tabela 5 estão alguns exemplos de SLAs gerados.

Algoritmo 3 – Gerador de SLA

```

1: procedimento GS(Valores mínimos e máximos dos atributos de Qos)
2:   para w = custo mínimo até w < custo máximo faça
3:     para x = carga de trabalho mínima até x < carga de trabalho máxima faça
4:       para y = tempo de resposta mínimo até y < tempo de resposta máximo faça
5:         para z = disponibilidade mínima até z < disponibilidade máxima faça
6:           válido ← executar(CloudSim[w,x,y,z])
7:           se válido então
8:             SLAs.add(SLA[w, x, y, z])
9:           fim se
10:        fim para
11:      fim para
12:    fim para
13:  retorna SLAs
14: retorna SLAs                                     ▷ retorna uma lista de SLA
15: fim procedimento

```

Com objetivo de avaliar os SLAs gerados, na linha 5 do Algoritmo 3, o CloudSim é executado com os valores de QoS gerados. Caso nem um dos valores exceda 20% dos valores retornados pelo CloudSim, então o SLA é adicionado como um SLA válido. Esse teste é executado com o objetivo de evitar SLAs ineficazes, que não são possíveis de aplicar em determinado contexto.

Este segundo módulo é acionado após a implantação do SLA determinado nas fases iniciais. O módulo de monitoramento é executado na Fase 3 apresentado na Figura 13, na qual, é realizado uma renegociação do SLA caso ocorram mais de “X” violação em um determinado espaço de tempo de monitoramento. O número de violações e o tempo de monitoramento é

² <https://aws.amazon.com/pt/ec2/instance-types/>

Tabela 5 – Exemplos de SLAs gerados

	Capacidade	Tempo de resposta (seg.)	Disponibilidade (%)	Custo/h (US\$)	Carga de Trabalho**
SLA 1	X*	100	70	0,2	80500
SLA 2	X*	150	70	0,53	100000
SLA 3	X*	180	80	0,6	125000
SLA 4	X*	200	90	0,7	170000

*A capacidade é a saída dos algoritmos de otimização de modo que atendam os atributos de QoS do SLA, uma vez que o cliente não possui conhecimento suficiente para estimar com precisão os recursos necessários.

**Cloudlet length: número de instruções que o processador irá executar.

definido de acordo com as políticas de tomadas de decisões estabelecidas pelo provedor ou negociadas no SLA.

5.4.1 Integração do framework em outras arquiteturas

Este trabalho vai de encontro com outras propostas do grupo de pesquisa em que se encontra, como o projeto de Doutorado de Bruno Guazzelli Batista ([BATISTA, 2016](#)) o qual propõe modelos de negócio que consideram atributos de QoS relacionados ao desempenho e a segurança. Nesse projeto de doutorado foi proposto um módulo de gerenciamento de recursos denominado *Resource Management Module* (ReMM) para gerenciar os recursos alocados aos clientes de forma a fazer frente à sobrecarga imposta por diferentes mecanismos de segurança.

ReMM considera a manipulação dos recursos computacionais durante o tempo de execução, respeitando as métricas de QoS definidas no contrato SLA, a fim de atender qualquer demanda de requisições. Esse módulo aplica tanto a escalabilidade horizontal quanto a vertical de forma dinâmica com impacto sobre o preço.

Uma requisição do cliente será respondida por um provedor, o qual utiliza três camadas básicas para oferecer diferentes serviços: camada de aplicação (SaaS), camada de plataforma (PaaS) e camada de infraestrutura (IaaS). A camada de aplicação gerencia todas as aplicações que são oferecidas para os clientes. A camada de plataforma inclui as políticas de mapeamento e de escalonamento responsáveis por transferir à camada de infraestrutura as exigências de QoS feitas pelos clientes e, dessa forma, alocar as máquinas virtuais para atendê-las. Por fim, a camada de infraestrutura controla a inicialização e remoção de máquinas virtuais de forma transparente ao cliente. A Figura 14 apresenta os detalhes do funcionamento do ReMM.

Inicialmente, cliente e provedor devem negociar um SLA, onde as métricas de QoS bem como os detalhes do contrato devem ser definidos (1). Após esse procedimento, diferentes clientes requisitam diferentes tipos de serviços a um provedor. O módulo Controle de Admissão é responsável por analisar a requisição e decidir se ela pode ser atendida ou não (2). Durante a sobrecarga do sistema, por exemplo, o provedor de serviços pode decidir entre rejeitar novas

Figura 14 – Funcionamento do ReMM (BATISTA, 2016).

requisições ou violar o SLA. A violação do SLA deve implicar em penalidades ao provedor. Se a requisição for aceita, ela será armazenada no módulo Fila de Requisições (3), onde ela receberá uma prioridade que definirá a sequência de execução (pode-se aplicar uma diferenciação de clientes, por exemplo).

Em seguida o ReMM fornecerá os recursos virtuais baseado nas informações definidas no SLA (4) alocando-os nos recursos físicos (5). Após a alocação dos recursos, o módulo Escalonador encaminha a requisição (6) para ser executada nos recursos providos (7.a) (7.b) utilizando políticas de escalonamento. Em determinados intervalos de tempo, o módulo Monitor de Desempenho coleta informações sobre o desempenho do sistema (8) e as envia ao ReMM (9.a). Essas informações (que representam o desempenho da requisição naquele momento) são comparadas com as informações de QoS definidas no contrato (9.b) e, caso alguma cláusula não esteja de acordo com o SLA, o ReMM ajusta dinamicamente a quantidade de recursos (de forma horizontal ou vertical) para garantir a QoS contratada (10). Todas as alterações no sistema influenciam no Modelo de Negócio (11).

Embora o ReMM tente cumprir o contrato com o cliente, garantindo a QoS contratada a um preço justo, e o provedor, com o cumprimento do contrato e o uso eficiente dos recursos disponíveis no sistema, ele não utiliza algoritmos de otimização para a realização das tarefas descritas como por exemplo, no processo de alocação das VMs nos recursos físicos. Além disso, ReMM na versão atual (utilizando CloudSim 3.0.3³) considera que todos os recursos físicos são ilimitados, e portanto, todas as requisições são aceitas. Dessa forma, verificou-se que ele carece de mecanismos responsáveis pela decisão de alocação dos recursos virtuais nos recursos físicos de forma otimizada, além de mecanismos que permitam um eficiente planejamento de capacidade de forma que todas as requisições aceitas pelo módulo **Controle de Admissão** possam ser atendidas conforme estipulado no SLA.

A proposta desenvolvida neste projeto de mestrado avaliou diferentes algoritmos de otimização com o intuito de fornecer informações sobre a demanda de serviços e, então, alocar as máquinas virtuais nos recursos físicos de forma eficiente, tornando mais precisas as alterações (quando necessárias) dos recursos realizadas pelo ReMM.

Outro projeto de doutorado que segue o mesmo viés do gerenciamento eficiente e automático de recursos virtualizados e que também busca garantir a qualidade de serviço é o projeto de Luis H. V. Nakamura (NAKAMURA, 2017). Nesse projeto de doutorado, busca-se criar mecanismos de autoconfiguração e auto-otimização que são baseados na teoria da computação autônoma. Esse projeto de doutorado envolve o uso da Web Semântica para criação e manipulação de informações armazenadas em ontologias. Tais dados são processados juntamente com regras estabelecidas pelos clientes e provedores a fim de se obter um gerenciamento personalizado da infraestrutura virtualizada. O doutorando responsável por esse projeto optou por utilizar a prototipagem e desenvolver todos os mecanismos em ambiente real, para isso ele contou com a infraestrutura disponível no LaSDPC que provem recursos de *hardware* como *clusters* que foram financiados pela FAPESP e o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) ao longo dos diversos projetos desenvolvidos pelo grupo.

A Figura 15 exibe a arquitetura proposta no projeto de doutorado descrito anteriormente. Nessa arquitetura observa-se a existência de um módulo de autoconfiguração que utiliza um adaptador elástico para o gerenciamento dos recursos virtualizados (VMs). Esses recursos são monitorados por agentes computacionais dentro do módulo *Resource Manager* (ReMan) que também tem a função de alterar as configurações dos mesmos. De acordo com os dados da carga de trabalho coletada pelos os agentes, o sistema de autoconfiguração toma uma decisão se deve ou não alterar as configurações das VMs ou inicializar ou finalizar novas instâncias de VMs. Para isso, o módulo de autoconfiguração utiliza dados do sistema de decisão que considera os dados e as regras estabelecidas entre usuário e provedor (SLA). Para a tomada de decisão, deve-se considerar o tempo de resposta para a carga atual e, se ela estiver violando o contrato SLA, então tomasse a decisão de aumentar ou diminuir os recursos dependendo do contexto. Tomada essa

³ <http://www.cloudbus.org/cloudsim/>

Figura 15 – Arquitetura proposta em Nakamura (NAKAMURA, 2017).

decisão, aciona-se o módulo de auto-otimização que está em constante funcionamento e utiliza de algoritmos de otimização para determinar dois pontos principais: 1). Determinar se a alteração de configuração será vertical ou horizontal e quantificar quanto de recurso será incrementado ou decrementado. 2) Determinar de qual forma e onde esse recurso será alocado ou liberado. Para cumprir esses dois pontos os algoritmos de otimização devem considerar a atual situação da infraestrutura, as informações contidas na ontologia e respeitar as regras definidas (como se fossem restrições).

Os algoritmos desenvolvidos neste projeto de Mestrado atuam diretamente nesse módulo de auto-otimização, uma vez que o *framework* desenvolvido é modular. Sendo preciso apenas alguns ajustes para definir e padronizar os parâmetros de entrada e as restrições para os algoritmos de otimização.

5.5 Algoritmos considerados

Os algoritmos de otimização foram desenvolvidos com o auxílio da ferramenta *Professional Optimization Framework* (ProOF) (ARANTES, 2014). O ProOF tem por objetivo orientar a implementação computacional de métodos variados, tais como, heurísticas, meta-heurísticas,

métodos exatos e híbridos aplicados a problemas de otimização. Essa ferramenta fornece uma base tanto para codificação dos algoritmos quanto para a realização dos experimentos. O primeiro passo foi a integração das funcionalidades do ProOF e do CloudSim. Em seguida, o problema foi codificado na ferramenta com cinco métodos de otimização: Algoritmo determinístico, *micro-Genetic Algorithm* (μ GA) *Simulated Annealing* (SA), *Tabu Search* (TS) e *Multi-Population Genetic Algorithm* (MPGA).

5.5.1 Codificação do problema

Esses métodos tem por objetivo otimizar o número de máquinas virtuais contratadas pelo cliente de forma que ele contrate apenas os recursos necessários, descritos na Seção 5.2. Assim, a representação de uma solução (codificação) é definida pela tripla (s, m, l) que corresponde respectivamente às VMs m3.large (*small*), m3.xlarge (*mean*) e m3.2xlarge (*large*). Trata-se do conjunto possíveis VMs a serem contratadas, dispostas em um vetor, onde cada posição do vetor corresponde à quantidade de instâncias de cada tipo, como apresentado na Figura 16.

Figura 16 – Representação da codificação da tripla (s, m, l) .

Os clientes podem definir a capacidade desejada (C^c), bem como o tempo de resposta (T^c), disponibilidade (A^c) e custo por hora (C/h^c). Contudo, é difícil satisfazer todos os requisitos sem conflitos. Por exemplo, a requisição da capacidade pode não satisfazer o custo desejado, ou ainda, pode não dar o tempo de resposta desejado. Por esse motivo, os métodos descritos nesta Seção serão aplicados com o objetivo de encontrar o melhor conjunto de VMs (s^*, m^*, l^*) que cheguem o mais próximo do que o cliente requisitou em termos de C^c , T^c , A^c e C/h^c .

5.5.2 Função objetivo

Para saber a qualidade de uma solução (codificação) (s', m', l') , é utilizado o simulador CloudSim para retornar os valores de $C^{c'}$, $T^{c'}$, $A^{c'}$, $C/h^{c'}$. Se esses parâmetros não forem compatíveis com os definidos no SLA (C^c , T^c , A^c , C/h^c), um outro conjunto de máquinas (s', m', l') deverá ser avaliada. Essa compatibilidade é avaliada através da Equação 5.5.

$$f(P[i]) = \left| \frac{C^c - C^*}{C^*} \right| + \left| \frac{T^c - T^*}{T^*} \right| + \left| \frac{A^c - A^*}{A^*} \right| + \left| \frac{C/h^c - C/h^*}{C/h^*} \right| \quad (5.5)$$

Essa equação é uma adaptação da Distância de Manhattan (BLACK, 2006), utilizada para avaliar o quão próxima a solução gerada está dos valores inicialmente definidos no SLA.

5.5.3 Método Determinístico

O primeiro método desenvolvido foi o chamado algoritmo determinístico. Esse método avalia exaustivamente todas as possíveis combinações para (s, m, l) como descrito no Algoritmo 4. O método gera e avalia todos os possíveis conjuntos de VMs nas linhas 3 a 5, mantendo a melhor solução encontrada. O método $Gerar(s, m, l)$ cria diferentes combinações de valores para (s, m, l) em cada iteração, essas combinações são avaliadas pelo método $Avaliar(s, m, l)$ utilizando a Equação 5.5. Finalmente, o melhor conjunto (s, m, l) (solução ótima) é retornada na linha 6.

Algoritmo 4 – Algoritmo Determinístico

```

1: procedimento DETERMINÍSTICO(SLA:  $C^c, T^c, A^c, C/h^c$ )
2:   repita ▷ varre todo o espaço de busca
3:      $Gerar(s, m, l)$  ▷ gera uma capacidade para ser avaliada
4:      $Avaliar(s, m, l)$ 
5:   até todas as combinações  $(s, m, l)$  dentro do domínio serem testadas
6:   retorna  $SLA^* : C^*, T^*, D^*, C/h^*$  ▷ retorna melhor configuração encontrada para
   satisfazer o SLA do cliente
7: fim procedimento

```

5.5.4 MPGA

O Algoritmo 5 descreve o MPGA. Trata-se de um algoritmo evolutivo que opera com uma população estruturada de forma hierárquica em árvore como proposto por Franca, Mendes e Moscato (2001). O MPGA foi aplicado para resolver diferentes problemas na literatura (BURIOL; MOSCATO, 2004; MOSCATO; MENDES; BERRETTA, 2007; TOLEDO *et al.*, 2013; ARANTES *et al.*, 2017) com relevantes resultados apresentados.

A Figura 17 ilustra a estrutura da população. As posições dos indivíduos (nós) nos *clusters* indicam sua posição dentro da hierarquia. Em cada *cluster*, os nós filhos (*supporters*) são os que possuem pior valor de *fitness* (função objetivo) em relação ao seus pais (*Leader*). Assim, o melhor indivíduo será o nó raiz enquanto os piores serão os nós folhas. O método evolui várias populações e, após a avaliação de uma população, o melhor indivíduo encontrado migra para a próxima população. O Algoritmo 5 é baseado no algoritmo genético híbrido descrito em Toledo *et al.* (2013)

Cada indivíduo representa um possível conjunto de VMs (s, m, l) para um cliente. O método InicializaPopulação(P) gera aleatoriamente os indivíduos (s, m, l) com $\min \leq (s + m + l) \leq \max$. Um total de $P[i].Tamanho\ indivíduos$ é gerado para cada população $P[i]$ e avaliado nas linhas 2 e 3 executando simulações no CloudSim. Em seguida, é iniciado o processo de evolução até que

Algoritmo 5 – Algoritmo MPGA

```

1: procedimento MPGA(SLA:  $C^c, T^c, A^c, C/h^c$ )
2:   para  $i \leftarrow 1$  Até  $nPopulacao$  faça
3:     InicializaPopulacao( $P[i]$ )
4:     AvaliacaoPop( $P[i]$ )
5:   fim para
6:   repita
7:     para  $i \leftarrow 1$  Até  $nPopulacao$  faça
8:       repita
9:         para  $count \leftarrow 1$  Até  $P[i].Tamanho * cruzamentoTaxa$  faça
10:          ( $paiA, paiB$ )  $\leftarrow$  Seleção( $P[i]$ )
11:           $novoInd \leftarrow$  Cruzamento( $paiA, paiB$ )
12:          Mutacao( $novoInd$ )
13:          AvaliacaoInd( $novoInd$ )
14:          Estruturar( $P[i], novoInd$ )
15:        fim para
16:      até  $P[i]$  convergir
17:      executaMigracao( $P[i]$ )
18:      reinicializarPop( $P[(i \bmod nPopulacao) + 1]$ )
19:    fim para
20:  até o tempo limite determinado
21:  retorna  $s$ 
22: fim procedimento

```

Figura 17 – Estrutura da população e migração.

haja convergência (linhas 4 a 16). Nesse processo, um total de $P[i].Tamanho * cruzamentoTaxa$ novos indivíduos é criado a cada ciclo evolutivo. Em um ciclo evolutivo, o método Seleção($P[i]$) seleciona aleatoriamente um indivíduo *Leader* e um dos seus nós seguidores no *cluster*. O operador de cruzamento é aplicado sobre esses dois indivíduos gerando um novo indivíduo.

Dois operadores de cruzamento foram implementados: *blend alpha crossover* (blx- α) e o cruzamento uniforme (EIBEN; SMITH, 2007). O cruzamento uniforme realiza trocas de genes entre os pais, por exemplo, dois indivíduos (A e B) são selecionados e cada gene do indivíduo filho tem 50% de probabilidade de ser proveniente do pai A ou do pai B (EIBEN; SMITH, 2007). Na Figura 18, o indivíduo filho (s', m', l') apresenta $s' = s^1, m' = m^1$ do pai A e $l' = l^2$ do pai B. O cruzamento blx- α define (s', m', l') em um novo indivíduo como $s' = \alpha \cdot s^1 + (1 - \alpha) \cdot s^2$, $m' = \alpha \cdot m^1 + (1 - \alpha) \cdot m^2$, $l' = \alpha \cdot l^1 + (1 - \alpha) \cdot l^2$, onde $\alpha \in [0, 1]$ (Figura 19). Um desses dois operadores de cruzamento é selecionado aleatoriamente cada vez que o cruzamento é realizado. O novo indivíduo pode apresentar $(s + m + l) \leq \min$ ou $\max \leq (s + m + l)$, logo, um ajuste é realizado incrementando ou decrementando o último valor l .

Figura 18 – Cruzamento uniforme.

Figura 19 – Cruzamento Blx- α .

O operador de mutação pode ser aplicado caso a taxa de mutação seja satisfeita, o que significa gerar aleatoriamente valores para $\lambda \in [0, 1]$ com $\lambda \leq mutRate$. Neste caso, um dos seis operadores de mutação desenvolvidos são selecionados para serem executados:

- **Reset Position:** altera um dos valores da codificação (s, m, l);
- **Reset Individual:** similar ao *reset position*, contudo, neste caso altera todas as posições em (s, m, l);

- **Swap:** troca os valores de duas posições em (s, m, l) ;
- **Proximity:** subtrai o valor de uma posição do conjunto (s, m, l) e incrementa em outra;
- **Incremental Position:** adiciona ou subtrai o valor de uma posição do conjunto (s, m, l) , respeitando os limites máximos e mínimos definidos;
- **Incremental Individual:** similar ao *incremental position*, contudo, incrementa ou subtrai os valores de todas as posições do conjunto (s, m, l) .

Após a aplicação dos operadores de cruzamento e mutação, o indivíduo gerado é avaliado em *AvaliaçãoInd(IndNovo)*. Esse novo indivíduo é incluído quando seu valor de *fitness* é melhor que o pai que possui pior *fitness*. Após sua inserção, *Estruturar(P[i], IndNovo)* atualizará a posição do novo indivíduo na hierarquia. Por exemplo, se esse indivíduo for melhor que o melhor encontrado, a hierarquia será ajustada para que ele seja o nó raiz da árvore.

Os ciclos evolutivos aplicados na população $P[i]$ convergem quando nenhum indivíduo novo é inserido após $P[i].Size * crossRate$ execuções. Neste ponto, uma cópia do melhor indivíduo de $P[i]$ é enviada pelo procedimento *executaMigracao(P[i])* para a próxima população. Finalmente, *reinicializarPop(P[(imodnPopulacao) + 1])* reinicializa a próxima população, exceto pelo seu melhor indivíduo e pelo indivíduo migrado. A execução do MPGA termina quando o critério de parada é satisfeito. Nesse trabalho, considerou-se um limite de tempo de execução como critério de parada.

5.5.5 Micro-GA

O Algoritmo 6 descreve o que é uma versão de AG cuja evolução considera uma população de menor porte e emprega constantes reinicializações para evitar convergência prematura. O termo μ GA foi introduzido por Krishnakumar (KRISHNAKUMAR, 1990) quem primeiro implementou o método com uma população de 5 indivíduos. O método também tem sido aplicado em diferentes tipos de problemas (JAEN-CUELLAR *et al.*, 2017; TRUONG; NGUYEN; KIM, 2017; TSAI *et al.*, 2015). Não é usual a presença do operador de mutação em um μ GA, mas os testes realizados no problema aqui estudado nos levaram a incluí-lo. O critério de convergência adotado foi baseado no pressuposto de que o melhor indivíduo não foi atualizado por uma certa quantidade de iterações. Quando o critério de convergência é satisfeito, a população é reinicializada e apenas o melhor indivíduo é mantido. O limite de tempo é o critério de parada.

Algoritmo 6 – Algoritmo μ GA

```

1: procedimento  $\mu$ GA(SLA:  $C^c, T^c, D^c, C/h^c$ )
2:   IniciaPopulação( $P$ )                                ▷ gera uma população com 5 indivíduos
3:   Avalia( $P$ )                                           ▷ avalia o fitness para cada indivíduo da população
4:   repita
5:     para  $i=1$  até  $P.size$  faça
6:        $Data\ Center \leftarrow P[i]$                       ▷ Data ceter recebe um indivíduo
7:       Avalia( $P[i]$ )                                    ▷ Avalia o fitness
8:     fim para
9:     se Melhor indivíduo não é atualizado por 10 iterações então
10:      Reinicializa população
11:    fim se
12:    Seleção( $P$ )
13:    Cruzamento( $P$ )
14:    Mutacao( $P$ )
15:  até Critério de parada ser satisfeito
16:  retorna  $s$       ▷ retorna melhor indivíduo encontrado para satisfazer o SLA do cliente
17: fim procedimento

```

O μ GA possui a mesma codificação do problema descrito para o MPGA, contudo, neste método é utilizado apenas uma população e os operadores genéticos aplicados são o BLX- α para cruzamento, *Reset Individual* para mutação e seleção para reprodução por torneio. Esse algoritmo possui uma complexidade menor e menos operadores, pois foi a primeira metaheurística desenvolvida visando observar o comportamento das soluções obtidas para o problema.

5.5.6 Simulated Annealing

Outro algoritmo desenvolvido é uma adaptação do método *Simulated Annealing* (SA) proposto por Kirkpatrick *et al.* (1983). Trata-se de uma metaheurística que executa uma busca local inspirada nos princípios da termodinâmica para aquecimento e resfriamento controlado de materiais. O Algoritmo 7 descreve o método SA implementado.

Para cada temperatura T , um total de SA_{max} soluções são geradas na vizinhança da solução corrente(s). O método *gerarVizinho*(s) seleciona aleatoriamente um dos seis operadores de mutação (descritos anteriormente para o MPGA), para gerar um vizinho s' da solução s (linhas 8-10). Caso a solução s' seja melhor, então o vizinho passa a ser a solução atual s (linha 12). Essa nova solução também pode se tornar a melhor solução (linhas 13-14). Por outro lado, existe a probabilidade de um uma solução s' pior ser aceita como a solução corrente, com base nos valores de Δ e T na expressão $e^{-\Delta/T}$ (linhas 15-19). Essa probabilidade é maior durante altas temperaturas T , a qual diminui utilizando a razão de resfriamento (linha 23). Se o tempo limite de execução não for atingido, a temperatura é reaquecida e é continuada a busca pela vizinhança da solução atual (linhas 26-27).

As soluções na vizinhança são avaliadas através da razão de resfriamento que atualiza

Algoritmo 7 – SA algorithm

```

1: procedimento SA(SLA:  $C^c, T^c, D^c, C/h^c$ )
2:    $s^* \leftarrow s$  ▷ melhor solução
3:    $T \leftarrow T_0$  ▷ temperatura atual
4:    $IterT \leftarrow 0$  ▷ número de iterações em T
5:   repita
6:     enquanto  $T > 0.001$  faça
7:       enquanto  $IterT < SMax$  faça
8:          $IterT \leftarrow IterT + 1$ 
9:          $s' \leftarrow \text{gerarVizinho}(s)$ 
10:         $\Delta \leftarrow f(s') - f(s)$ 
11:        se  $\Delta < 0$  então
12:           $s \leftarrow s'$ 
13:          se  $f(s') < f(s^*)$  então
14:             $s^* \leftarrow s'$ 
15:          fim se
16:        senão
17:           $x \in [0, 1]$ 
18:          se  $x < e^{-\Delta/T}$  então
19:             $s \leftarrow s'$ 
20:          fim se
21:        fim se
22:      fim enquanto
23:       $T \leftarrow \frac{T}{1 + \alpha \sqrt{T}}$ 
24:       $IterT \leftarrow 0$ 
25:    fim enquanto
26:     $s \leftarrow s^*$ 
27:     $T \leftarrow T_0$ 
28:  até tempo limite determinado
29:  retorna  $s$ 
30: fim procedimento

```

a temperatura atual $T = \frac{T}{1 + \alpha \sqrt{T}}$, onde α é um parâmetro de controle do método. Há outros parâmetros de controle como o número de iterações ($SMax$) para procurar por soluções na vizinhança atual e a temperatura inicial (T_0).

5.5.7 Tabu Search

Tabu Search (TS) ou Busca Tabu, é outro método de busca local proposto por Glover (1986) que utiliza uma estrutura de memória que permite escapar de ótimos locais. Isso pode ser feito pela proibição de movimentos de busca em vizinhanças que tenham sido recentemente executados. Os operadores de mutação do MPGA também são utilizados nesse algoritmo como movimentos para gerar soluções vizinhas. O Algoritmo 8 descreve o método TS proposto. Neste caso, s representa uma configuração (s, m, l) e a função $f(P[i])$ (Equação 5.5) é reescrita como $f(s)$. O parâmetro *melhorIter* é a última iteração em que a melhor solução foi atualizada, TL

é a lista que contém os movimentos tabu e $BTmax$ é o número máximo de iterações em que a melhor solução não foi modificada.

Algoritmo 8 – Algoritmo TS

```

1: procedimento TS(SLA:  $C^c, T^c, D^c, C/h^c$ )
2:    $s^* \leftarrow s$ 
3:   repita
4:      $melhorIter \leftarrow 0$ 
5:      $Iter \leftarrow 0$ 
6:      $TL \leftarrow \emptyset$ 
7:     enquanto  $Iter - melhorIter \leq TSmax$  faça
8:        $Iter \leftarrow Iter + 1$ 
9:        $s' \leftarrow \text{selecionaMelhorVizinho}(s)$ 
10:       $\Delta \leftarrow f(s') - f(s)$ 
11:      se  $\Delta < 0$  então
12:         $s \leftarrow s'$ 
13:         $\text{Atualiza}(TL)$ 
14:        se  $f(s) < f(s^*)$  então
15:           $s^* \leftarrow s$ 
16:           $melhorIter \leftarrow Iter$ 
17:        fim se
18:      fim se
19:    fim enquanto
20:     $s \leftarrow s^*$ 
21:  até tempo limite determinado
22:  retorna  $s$ 
23: fim procedimento

```

A busca local é realizada enquanto o tempo limite não é atingindo e se a melhor solução foi atualizada recentemente (linhas 3-21). Inicialmente, os seis operadores de mutação são aplicados na solução corrente s e o melhor vizinho s' encontrado é selecionado (linha 9). Caso s' seja melhor que s , a solução atual é substituída por s' e o movimento (operador de mutação) que gerou s' se torna tabu (linha 13). Assim, esse movimento não será aplicado novamente para gerar um outro vizinho. Se o tamanho de TL for excedido, a política *First In, First out* (FIFO) é utilizada para inserir e remover movimentos da lista. O parâmetro $melhorIter$ é atualizado com o valor da atual iteração $Iter$ quando uma nova melhor solução é encontrada, então é possível continuar a busca local na vizinhança de s^* (linha 20).

5.5.8 NSGA-II

Como descrito na Seção 3.5, o problema tratado neste trabalho é caracterizado como um problema multi-objetivo. Nos algoritmos anteriores, a função objetivo empregava uma normalização dos diferentes objetivos. Apesar de obter uma solução capaz de estabelecer o contrato a partir das demandas do cliente, o conflito entre os objetivos existentes pode não ter sido devidamente tratado ao se assumir pesos iguais. Portanto, para avaliar e oferecer soluções

que sejam ainda mais aderentes ao contexto do problema, foi implementado uma meta-heurística multi-objetiva chamada NSGA-II.

O NSGA-II foi proposto por [Deb et al. \(2002\)](#) como uma versão melhorada do NSGA. O método consiste em criar uma população (P) de forma aleatória para que, em seguida, sejam aplicados os operadores de mutação e cruzamento. Assim como no MPGA e no AG, isso resulta em uma população de descendentes ou filhos (F). Após a geração dos descendentes, tanto os indivíduos filhos como os pais são agrupados em um conjunto (Q). Nesse grupo, é aplicada então a comparação de não dominância como descrita na Seção 3.5. Isso resulta em um agrupamento f_1 e, de forma análoga, é aplicada a comparação de não dominância para os demais indivíduos, resultando em outros grupos (f_2 e f_3). Como última etapa, para determinar a próxima população, os indivíduos são organizados em cada grupo de forma decrescente a partir das distâncias entre as suas funções objetivos. A próxima população será formada pelos indivíduos do melhor grupo e os mais distantes dele, mantendo assim a diversidade. Esse processo é repetido até que atinja o tempo limite determinado. A Figura 20 ilustra esse procedimento.

Figura 20 – Ilustração do NSGA-II (adaptado de [Deb et al. \(2002\)](#)).

O NSGA-II aplica um método para ordenar as soluções pela não-dominância. Esse método é denominado *fast-non-dominated-sort*. O método calcula a dominância em duas etapas. Primeiro, para todas as soluções, é calculado um grau de dominância (n_p), ou seja, o número de soluções que dominam a solução p (s, m, l). Se o valor de n_p for 0, significa que é uma solução p que não é dominada e fará parte do primeiro conjunto. O segundo passo é separar as soluções em

grupos S_p por ordem de dominância. Logo, cada indivíduo que é adicionado a um conjunto S_p é removido da população e os indivíduos dominados por ele tem seu valor de n_p decrementado. O passo dois é repetido até que não existam mais indivíduos na população. No Algoritmo 9, essas duas etapas são apresentadas passo-a-passo, onde q é apenas outra solução da população a ser comparada com p .

Algoritmo 9 – *fast-non-dominated-sort*

```

1: para cada  $p \in P$  faça
2:    $S_p = \emptyset$ 
3:    $n_p = 0$ 
4:   para  $q \in P$  faça
5:     se  $p \prec q$  então
6:        $S_p = S_p \cup \{q\}$ 
7:     senão
8:       se  $q \prec p$  então
9:          $n_p = n_p + 1$ 
10:    fim se
11:  fim se
12:  fim para
13:  se  $n_p = 0$  então
14:     $f_1 = f_1 \cup \{p\}$ 
15:  fim se
16: fim para
17:  $i = 1$ 
18: enquanto  $f_i \neq \emptyset$  faça
19:    $Q = \emptyset$ 
20:   para cada  $p \in f_i$  faça
21:     para cada  $q \in S_p$  faça
22:        $n_p = n_p - 1$ 
23:     se  $n_p = 0$  então
24:        $Q = Q \cup \{q\}$ 
25:     fim se
26:   fim para
27:   fim para
28:    $i = i + 1$ 
29:    $f_i = Q$ 
30: fim enquanto

```

Após a indexação das soluções nos conjuntos pela não dominância, as soluções são ordenadas pela distância entre as funções. Essa distância é obtida através do cálculo da distância média entre um ponto central i . Isso nada mais é do que um indivíduo da população e os pontos em suas extremidade, que acabam por formar um cuboide. Esta ordenação é feita com o objetivo de manter na população as soluções mais distantes, garantindo certa diversidade na população. Na Figura 21 é ilustrado a disposição dos pontos em relação a essa distância.

Uma vez realizada a indexação das soluções pela ordem de não dominância e o orde-

Figura 21 – Distancia entre as funções objetivo (adaptado de [Deb et al. \(2002\)](#)).

namento das mesmas pela distância entre as funções, todo o processo evolutivo é repetido até que o tempo definido seja atendido. No Algoritmo 10, é apresentado o fluxo dessa execução passo-a-passo. Na linha 20 do algoritmo é feita a ordenação pela distância da função no último conjunto de de f , e na linha 21 é selecionado o primeiro indivíduo desse ultimo conjunto para permanecer na próxima população

Algoritmo 10 – Algoritmo NSGA-II

```

1: procedimento NSGA-II(SLA:  $C^c, T^c, A^c, C/h^c$ )
2:    $t = 1$ 
3:   InicializaPopulacao( $P_t$ )
4:   Avaliao( $P_t$ )
5:   repita
6:      $F_t = copia(P_t)$ 
7:     Selecao( $F_t$ )
8:     Cruzamento( $F_t$ )
9:     Mutacao( $F_t$ )
10:    Avaliacao( $F_t$ )
11:    Estruturar( $F_t$ )
12:     $Q = P \cup F$ 
13:     $f[] = fast-non-dominated-sort(Q)$ 
14:     $P_{t+1} = \emptyset$ 
15:     $i = 1$ 
16:    repita
17:      Ordenar-Pela-Distância( $f_i$ )
18:       $P_{t+1} = P_{t+1} \cup f_i$ 
19:       $i = i + 1$ 
20:    até  $|P_{t+1}| + |f_i| \leq P_t.tamanho$ 
21:    Odernar( $f_i, \prec_n p$ )
22:     $P_{t+1} = P_{t+1} \cup f_i[1 : (N - P_t + 1)]$ 
23:     $t = t + 1$ 
24:  até o tempo limite determinado
25:  retorna  $P_{t-1}$ 

```

AVALIAÇÃO EXPERIMENTAL

6.1 Considerações Iniciais

O presente Capítulo apresenta os resultados obtidos neste projeto de mestrado. Assim, os experimentos visando a avaliação dos módulos de estabelecimento do SLA e monitoramento dos contratos serão descritos. Em seguida, os resultados alcançados pelas técnicas propostas são apresentados e analisados. A Seção 6.2.1 apresenta a configuração de ambiente utilizada para os experimentos. Na Seção 6.2.2, é discutido o planejamento dos experimentos. Por fim, na Seção 6.2.3, a análise dos resultados é reportada. Por fim, a Seção 6.3 apresenta as considerações finais deste capítulo.

6.2 Avaliação do *framework* de otimização

Nesta seção é avaliado o *framework* de otimização com os dois módulos desenvolvidos, os quais contém os algoritmos de otimização implementados.

6.2.1 Configuração do Ambiente

Todos os experimentos foram executados no simulador CloudSim versão 3.0.3 ¹, utilizando um computador com um processador AMD Phenom(tm) II X6 1090T, com 16 GB de memória RAM, com 1,5 TB de armazenamento em disco e sistema operacional Ubuntu 14.04.3 LTS com núcleo versão 3.13.0. O CloudSim foi configurado para executar com três tipos de instâncias de VMs como descritas na Seção 5.4.

¹ <http://www.cloudbus.org/cloudsim/>

6.2.2 Planejamento de experimentos

Para a execução dos experimentos foram elaborados 5 cenários. Nos 4 primeiros cenários, foi criado um SLA estipulando a capacidade (C^c), o tempo (T^c), a disponibilidade (D^c) e o custo/h (C/h^c) para um determinado cliente, no ultimo cenário foram avaliados apenas os objetivos conflitantes, o tempo e o custo. A simulação consiste em executar no CloudSim a aplicação do cliente a partir da capacidade descrita no SLA. A simulação retorna o tempo, a disponibilidade e o custo/h demandados pelo sistema modelado por [Batista \(2016\)](#) no CloudSim. Caso o tempo, disponibilidade e custo/h não sejam compatíveis com as descritas no SLA, é aplicado então um método de otimização para determinar qual a capacidade que melhor atende ao SLA.

Um número máximo de VMs é considerado no *data center* e tais VMs se dividem em três tipos: **m3.medium**, **m3.large** e **m3.xlarge** (apresentadas na Seção 5.2). Um terço do total das VMs é destinado a cada tipo.

6.2.2.1 Cenário 1

No cenário 1, o SLA é atendido de acordo com a quantidade de VMs disponíveis, onde é suposto que o usuário contrata o ponto médio do número de VMs em cada configuração. A Tabela 6 apresenta o número de VMS disponíveis, o intervalo de busca e o número de VMS contratadas pelo cliente em cada situação avaliada. Por exemplo, se a configuração executada possui 10 VMs, então o número de VMs contratadas é 5 e o intervalo de busca dos métodos de otimização será de 1 até 10 VMs. Neste cenário, foi utilizado um modelo de serviço de repositório de arquivos, o Apache Benchmark ([APACHE, 2017](#)), este modelo de serviço foi o mesmo utilizado no trabalho de [Batista \(2016\)](#), trabalho este, precursor do presente projeto de mestrado.

Outra variação no SLA é em relação ao tempo, custo e carga de trabalho executada. Essa variação pode ocorrer de duas formas. Na primeira, quanto maior a configuração, maior será o custo contratado no SLA e maior será a carga de trabalho aplicada. Na segunda, a carga de trabalho permanece a mesma para todas as configurações, no entanto, o custo contratado no SLA aumenta e o tempo de resposta esperado pelo cliente diminui. A Tabela 7 apresenta a variação em relação ao número de VMs.

Neste cenário é executado o μ AG, com uma população de 5 indivíduos e aplicando 3 minutos como critério de parada. Foram utilizados os operadores genéticos, BLX- α para cruzamento, *Reset Individual* para mutação e seleção por torneio para reprodução.

6.2.2.2 Cenário 2

A diferença do cenário 2 para o cenário 1 reside na definição de um subintervalo para o número de VMs. Logo, os métodos de otimização não executam a busca em todo o intervalo de busca, mas na vizinhança do número médio de VMs contratadas pelo cliente como apresentado na

Tabela 8. Nesse cenário, supõe-se que o cliente indique uma quantidade de VMs a ser contratada. Assim, a busca para atender ao cliente será feita na vizinhança do número de VMs que ele está disposto a contratar. Os métodos utilizam a mesma constante α para estipular o intervalo de busca.

Neste cenário, foi replicado o experimento do cenário 1 com adição de um outro algoritmo, o MPGA. O MPGA foi executado com os mesmos operadores e critério de parada do μ AG, mas com 3 populações compostas por 5 indivíduos.

6.2.2.3 Cenário 3

Dois modelos de serviço foram utilizados para a execução dos experimentos, avaliando assim aplicações CPU-Bound e I/O-Bound. O primeiro foi um serviço de repositório de arquivos (Apache Benchmark), e o segundo foi um serviço de renderização de imagens (Smallpt Benchmark (SUITE, 2017)). Além disso, foi utilizado o gerador de SLAs descrito na Seção 5.4 que criou 100 SLAs para execução dos experimentos. A configuração de ambiente utilizada nesta bateria de testes é a mesma descrita na Seção 6.2.1.

O CloudSim foi configurado para executar com uma quantidade limitada de recursos (100 VMs), dividido de maneira uniforme entre VMs *small*, *medium* e *large*, assim como descrito na Seção 6.2.2. Para os experimentos neste ambiente, 100 clientes foram gerados que alocariam

Tabela 6 – Variação do número de VMs de acordo com a configuração.

Número de VMs disponíveis	Intervalo de busca	Número de VMs contratadas
10	1-10	5
20	1-20	10
30	1-30	15
40	1-40	20
50	1-50	25
60	1-60	30
70	1-70	35
80	1-80	40
90	1-90	45
100	1-100	50

Tabela 7 – Variação do SLA.

Parâmetros	Variação 1	Variação 2
Configurações (nº VMs)	↑	↑
Tempo	—	↓
Custo	↑	↑
Carga de trabalho	↑	—

↑: Aumenta;
 ↓: Diminui;
 —: Permanece inalterado.

Tabela 8 – Variação do número de VMs.

Número de VMs disponíveis	Intervalo de busca	Número de VMs contratadas
10	$ \alpha - 5 - 5 + \alpha $	5
20	$ \alpha - 10 - 10 + \alpha $	10
30	$ \alpha - 15 - 15 + \alpha $	15
40	$ \alpha - 20 - 20 + \alpha $	20
50	$ \alpha - 25 - 25 + \alpha $	25
60	$ \alpha - 30 - 30 + \alpha $	30
70	$ \alpha - 35 - 35 + \alpha $	35
80	$ \alpha - 40 - 40 + \alpha $	40
90	$ \alpha - 45 - 45 + \alpha $	45
100	$ \alpha - 50 - 50 + \alpha $	50

α : é uma constante para estipular um intervalo de busca.

recursos na nuvem em tempos distintos (5 clientes por vez), como ilustrado na Figura 22. A medida que os clientes enviavam a requisição para estabelecer o SLA, executava-se os algoritmos de otimização para negociá-lo.

Figura 22 – Linha do tempo da negociação do SLA entre cliente e provedor.

Para cada conjunto de clientes em que eram negociados os SLAs, os algoritmos de otimização SA, TS e MPGA foram executados por 10 vezes e a melhor solução obtida era selecionada. Essas soluções eram comparadas com as soluções retornadas pelo método determinístico, considerando o intervalo total dentro das 100 VMs (1-100). As 10 execuções de cada algoritmo foram realizadas de maneira distribuída (simultaneamente), assim se podia aumentar a chance de obtenção de uma solução de qualidade dentro de um tempo computacional reduzido. Na Tabela 9 são apresentadas as configurações aplicadas para cada algoritmo. Esses valores foram definidos empiricamente com base em configurações anteriores definidas para os mesmos métodos em (TOLEDO *et al.*, 2013). Em uma primeira instância, foram avaliados todos os algoritmos para o fechamento do contrato, em seguida, foi selecionado o algoritmo com melhor desempenho para executar em todas as fases do SLA, desde o fechamento e monitoramento até a finalização (Cenário 4).

Ainda neste cenário, foi realizada uma avaliação da utilização dos recursos e da redução de custo em relação ao atendimento do SLA. Nesta avaliação, o objetivo foi comparar duas

Tabela 9 – Configuração dos algoritmos

Características	SA	Tabu	MPGA
Tempo de execução (minutos)	5	5	5
TEmpérature (max-min)	1000-0.001	-	-
alpha	0.95	-	0.2
Operadores	All	All	All
SAmax	3	-	-
TSmax	-	10	-
TL	-	1	-
Vizinhos	-	5	-
Tabu Max	-	1	-
Indivíduos	-	-	5
Populações	-	-	3
Taxa de mutação	-	-	0.7
Taxa de cruzamento	-	-	5.0

arquiteturas, uma com e outra sem os algoritmos de otimização, de modo a manter os atributos de QoS estipulados e reduzir o custo do cliente. Neste mesmo experimento, foi reduzido o poder computacional do provedor, com objetivo de avaliar o comportamento do *framework* nos casos de disputa por recursos. No último experimento deste cenário e nos cenários seguintes, foi aplicado apenas uma vez o algoritmo que obteve o melhor desempenho (MPGA). Nesse caso, o tempo de execução foi reduzido para o teto do tempo para encontrar a melhor resposta nos experimentos anteriores, porém, foram obtidas as melhores respostas em menos de 2 minutos em todas as execuções. Assim, o tempo limite de 2 minutos passou a ser definido como critério de parada.

6.2.2.4 Cenário 4

Neste cenário ocorre a avaliação do módulo de monitoramento. O objetivo é avaliar a execução completa da arquitetura, ou seja, tanto a execução do módulo de estabelecimento do SLA quanto o módulo de monitoramento. Para avaliar o comportamento da arquitetura em caso de violações de contrato, é necessário causar uma perturbação no ambiente, realizada por meio da carga de trabalho. Os valores da perturbação da carga foram gerados seguindo uma distribuição normal gaussiana fornecida pela Equação 6.1:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x^2-\mu)^2}{2\sigma^2}} \quad (6.1)$$

onde, x é uma variável aleatória contínua definida por $-\infty < x < +\infty$, μ é a mediana utilizada no desvio que neste caso é a carga de trabalho e σ é o desvio padrão aplicado na distribuição, onde foi atribuído o valor de 20% da carga de trabalho.

Aplicando essa distribuição, os valores gerados apresentam maior proximidade do valor médio (a carga de trabalho). Dessa forma, a variação de carga do cliente ficou próxima ao valor de

carga informado inicialmente, conforme apresentado na Figura 23. No histograma apresentado, onde o eixo “x” indica a distribuição dos valores gerados enquanto o eixo “y” representa a frequência média com que esses valores são gerados.

Figura 23 – Distribuição gaussiana.

De modo a garantir que houvessem violações nos contratos para a avaliação conduzida, foram gerados outras duas distribuições, nas quais os *outliers* foram utilizados como mediana. Assim, os valores atribuídos para a carga de trabalho de alguns clientes foram as distribuições contidas nas normais dos *outliers* como apresentado na Figura 24. Os clientes nos quais foram aplicados essas distribuições dos *outliers* são denominados **clientes mutantes**,

Figura 24 – Distribuição gaussiana com a gaussiana dos valores aberrantes.

Para os experimentos, foram utilizados os mesmos clientes definidos na Seção 6.2.3.3 e a execução segue o seguinte fluxo para cada cliente:

1. Um contrato inicial é definido para o cliente aplicando o MPGA apenas uma vez com as configurações descritas na Tabela 9, exceto pelo critério de parada que foi reduzido para 2 minutos;

2. O cliente executa a aplicação na infraestrutura contratada de acordo com o SLA;
3. Uma perturbação é causada nos clientes mutantes aplicando a distribuição normal descrita anteriormente.
4. O passo 3 é repetido enquanto o cliente estiver alocado no *data center*;
5. Um novo SLA é gerado baseado no histórico de violações, caso o número de violações exceda uma quantidade previamente definida. Esse será o caso dos clientes rotulados como mutantes. Assim, é realizada uma média das cargas que ocasionaram as violações que será utilizada como parâmetro de entrada para o algoritmo de otimização. Nesta fase, o MPGA é executado novamente por 2 minutos (apenas uma vez);
6. Por fim, após ser proposto e aceito um novo SLA, o cliente mutante volta para a fase de monitoramento.

Como descrito na Seção 5.3, todos esses passos são executados em paralelo e o número de *threads* é limitado pelo número de núcleos disponíveis na máquina *host* que gerencia todos esses processos. Portanto, é estabelecido uma sincronização entre os clientes (*threads*) para tratar a condição de corrida (apresentado na Figura 13), visto que os clientes estão a todo momento requisitando recursos.

6.2.2.5 Cenário 5

No cenário 5, executa-se a meta-heurística multi-objetiva NSGA-II por 10 vezes, aplicando seis minutos como critério de parada para alguns dos clientes gerados anteriormente. Uma curva de Pareto foi obtida para cada cliente a partir da qual são coletadas três tipos de soluções não dominantes: uma priorizando a redução do tempo de resposta, outra priorizando a redução do custo e a última fornece a maior compatibilidade possível para ambos os valores conflitantes, denominada solução compromisso.

Assim como no Cenário 3, neste experimento, dois modelos de serviço foram utilizados para a execução dos experimentos: Apache e Smallpt Benchmark. Neste caso, a função objetivo focou na minimização dos dois objetivos conflitantes, o tempo e o custo, resultando nas seguintes funções objetivos:

$$\begin{aligned} f_1(x) &= \text{Tempo} & e, \\ f_2(x) &= \text{Custo}/h \end{aligned} \quad (6.2)$$

Neste cenário, foi avaliada a negociação do contrato de forma a oferecer para o cliente 3 soluções, em que ele passa priorizar a não dominante com menor custo, menor tempo ou a solução compromisso.

6.2.3 Análise dos resultados

Nesta Seção são apresentados os resultados para as execuções dos experimentos de cada cenário descrito na Seção 6.2.2. Para todos os cenários, compreende-se por **taxa de sucesso** o percentual em que os algoritmos chegaram à solução ótima (aquela encontrada pelo método determinístico).

6.2.3.1 Resultados obtidos no cenário 1

A Tabela 10 apresenta os resultados do método determinístico, listando o número de VMs, o tempo de resposta do método e o número de combinações geradas até a melhor configuração (solução) ser obtida. O objetivo é avaliar o tempo que o método exato leva para gerar a quantidade de combinações necessária para encontrar a solução ótima.

Tabela 10 – Tempo de resposta e número de combinações obtidas pelo método determinístico (cenário 1).

Número de VMs	Tempo de resposta (minutos)	Número de combinações
10(9)	0,83	63
20(18)	5,00	342
30(30)	15,51	1330
40(39)	35,29	2743
50(48)	67,18	4912
60(20)	113,47	9260
70(69)	176,66	13823
80(78)	259,71	19682
90(30)	386,97	29790
100(99)	533,37	39303

Observa-se que para um *data center* com 20 VMs já há um tempo de resposta consideravelmente alto, na escala de minutos (5 minutos). A medida que a quantidade de VMs no *data center* aumenta, o tempo de resposta cresce exponencialmente. Por exemplo, para 100 VMs, a configuração é obtida depois de 8,88 horas. Isso demonstra que o método determinístico se torna inviável neste cenário mesmo para uma quantidade pequena de VMs.

O μ AG foi executado 100 vezes por durante 3 minutos para cada configuração a ser determinada. Ao final, foi obtido o tempo de resposta e uma taxa de sucesso do μ AG nas 100 execuções. Essa taxa indica o número de execuções em que a meta-heurística conseguiu chegar na mesma solução que o método determinístico. O tempo de resposta representa o tempo gasto em média para se alcançar a solução do método determinístico. Se a solução ótima não foi encontrada em alguma execução, o tempo total é considerado no cálculo da média. A Tabela 11 apresenta os resultados das 100 execuções em cada configuração. Os resultados obtidos neste cenário apresentaram uma taxa de sucesso consideravelmente baixa e tempo de resposta alto.

A fim de aprofundar esta investigação, outros testes foram realizados, aumentando o tempo de execução. A Tabela 12 apresenta os resultados de 100 execuções em três instantes

Tabela 11 – Tempo de resposta e taxa de sucesso obtido pelo μ AG em 3 minutos (cenário 1).

VMs	Média em minutos	Taxa de sucesso (%)
10	1,95	53
20	2,75	20
30	2,73	55
40	2,79	36
50	2,98	4
60	2,64	26
70	2,97	23
80	2,99	6
90	2,99	6
100	2,99	4

de tempo distintos. Analisando a Tabela 12, observa-se que quanto maior o tempo de execução do algoritmo, maior é a taxa de sucesso. O tempo de execução do μ AG é bem inferior ao do determinístico, embora as execuções que leva de 5 e 10 minutos podem não ser realistas no contexto do problema considerado.

Tabela 12 – Taxa de sucesso obtido pelo μ AG em 3, 5 e 10 minutos.

VMs	Taxa de sucesso (%)		
	3 minutos	5 minutos	10 minutos
10	53	84	88
20	20	52	85
30	55	80	90
40	36	81	87
50	4	32	81
60	26	45	91
70	23	36	86
80	6	24	81
90	6	38	79
100	4	35	68

6.2.3.2 Resultados obtidos no cenário 2

A Tabela 13 apresenta cada uma das configurações, o tempo de resposta e o número de combinações necessárias para o método determinístico encontrar a solução ótima dentro do intervalo considerado. Nesse experimento, o intervalo coberto segue a taxa $\alpha = 0.5$ acima e abaixo do número de VMs que o cliente requisitou. Por exemplo, se o cliente contratou 50 VMs, o intervalo fica entre 25 e 75 VMs.

Observa-se que, como no Cenário 1, em um *data center* com 20 VMs há um tempo de resposta consideravelmente alto, na escala de minutos (2,19 minutos). A medida que a configuração é modificada em um *data center* para uma quantidade maior de VMs, o tempo de

Tabela 13 – Tempo de resposta e número de combinações obtidas pelo método determinístico.

Número de VMs	Tempo de resposta (minutos)	Número de combinações
10	0,35	116
20	2,19	781
30	6,28	2216
40	14,83	5236
50	27,08	9516
60	47,53	16616
70	72,38	25266
80	108,56	38171
90	150,25	4096
100	213,86	4912

resposta continua aumentando exponencialmente, sinalizando uma resposta depois de 3,5 horas. Isso também indica que o método determinístico ainda é inviável mesmo para o intervalo menor de VMs deste cenário.

Assim como no cenário 1, o μ AG foi executado durante 3 minutos, repetindo-se 100 execuções para cada configuração. A Tabela 14 apresenta os resultados obtidos pelas 100 execuções usando as métricas anteriormente definidas. Neste cenário, o μ AG chegou à solução subótima do método determinístico nas 100 execuções para todas as configurações.

Tabela 14 – Tempo de resposta obtido pelo μ AG em todas as configurações (cenário 2).

VMs	Média em minutos	σ Tempo em minutos
10	0,13	0,15
20	0,54	0,32
30	0,19	0,19
40	0,32	0,21
50	0,56	0,24
60	0,17	0,19
70	0,14	0,17
80	0,17	0,20
90	0,19	0,21
100	0,16	0,16

Na Figura 25, observa-se a curva do tempo em relação a cada configuração. A linha contínua com marcador triangular representa o tempo de resposta obtido pelo algoritmo determinístico, a linha pontilhada com marcador retangular representa um limite de tempo de três minutos. Esse limite no contexto do problema é determinado como o tempo de resposta aceitável para o usuário e, por fim, a linha pontilhada com marcador em forma de losango representa a média do tempo de resposta das 100 execuções para cada configuração em que foi aplicado o μ AG. Analisando o tempo de resposta do algoritmo determinístico, pode-se inferir que ele possui complexidade de tempo exponencial, tempo bem acima do tolerado pelo cliente. Por

outro lado, o μ AG obteve um tempo de resposta consideravelmente menor que o esperado pelo usuário.

Figura 25 – Gráfico do tempo de resposta do modelo determinístico e o μ AG em razão de cada configuração aplicada.

Analisando a Tabela 14, observa-se que o μ AG obteve um tempo de resposta reduzido em todas as configurações com uma taxa de sucesso de 100%. Essa taxa de sucesso foi obtida em menos de 1 minuto em média (para todas as configurações), em contraste com o modelo determinístico que levou 213,86 minutos (superior a 3,5 horas) para retornar uma solução. A meta-heurística poderia ter sido executada por 1,5 minuto e ainda assim teria uma alta taxa de sucesso. Logo, observa-se que houve uma melhora considerável no tempo de resposta e um aumento na taxa de sucesso para 100%, ao se comparar as soluções do μ AG com as soluções subótimas do método determinístico.

Nesta etapa do trabalho, observou-se que o μ AG apresentava limitações para retornar as melhores soluções, dentro de um tempo reduzido, caso um intervalo menor de busca não fosse previamente definido. Por isso, uma outra meta-heurística foi avaliada na tentativa de encontrar rapidamente boas soluções dentro do contexto mais realista abordado no cenário 1. O desempenho do μ AG foi comparado a outro tipo de algoritmo evolutivo, o MPGA. Os resultados obtidos seguem na Tabela 15.

O MPGA apresentou um desempenho superior ao μ AG, alcançando 100% de sucesso na maioria dos casos tratados no cenário 1. Além disso, o tempo de execução para obter a solução ótima foi inferior ao tempo obtido pelo μ AG. Assim, nos experimentos seguintes, o MPGA e outras meta-heurísticas passaram a ser avaliados.

Tabela 15 – Tempo de resposta e taxa de sucesso obtido pelo MPGA em relação ao μ AG em 3 minutos (cenário 1).

VMs	μ AG		MPGA	
	Média em minutos	Taxa de sucesso (%)	Média em minutos	Taxa de sucesso (%)
10	1,95	53	1,5	100
20	2,75	20	1,8	100
30	2,73	55	2,25	100
40	2,79	36	1,75	100
50	2,98	4	1,25	100
60	2,64	26	2,5	98
70	2,97	23	1,15	100
80	2,99	6	2,66	100
90	2,99	6	2,25	100
100	2,99	4	1,33	100

6.2.3.3 Resultados obtidos cenário 3

A Figura 26 apresenta os resultados para cada SLA gerado, utilizando como serviço o Apache Benchmark. A função de avaliação representa a média do melhor valor de $f(x)$ ao final das 10 execuções de cada método para cada cliente. O tempo de execução de cada método foi limitado a 5 minutos. Nota-se pelo gráfico da Figura 26 que, com exceção do SA, as linhas estão sobrepostas. Isso indica que a busca tabu e o MPGA encontraram em média soluções próximas às soluções ótimas retornadas pelo método determinístico. Contudo, em um determinado conjunto de clientes, ampliando o gráfico (Figura 27), observa-se que a Busca Tabu não obteve a solução ótima em alguns casos, mas encontrou uma solução quase tão boa quanto a ótima.

Figura 26 – Qualidade das soluções dos algoritmos SA, TB e MPGA em relação ao método determinístico com o Apache Benchmark.

A Figura 28 apresenta os resultados obtidos utilizando como serviço o Smallpt benchmark. Nestes experimentos é possível observar que as soluções geradas pelo determinístico e MPGA não apresentam discrepâncias, enquanto o SA apresentou o pior desempenho entre os

Figura 27 – Ampliação de um intervalo do gráfico da Figura 26.

métodos.

Figura 28 – Qualidade das soluções dos algoritmos SA, TB e MPGA em relação ao método determinístico com o Smallpt benchmark.

Outra análise conduzida foi a aferição da confiabilidade das soluções, uma vez que SA, TABU e MPGA são métodos estocásticos. Uma amostra de cinco clientes, dentre os 100 avaliados, foi extraída aleatoriamente. Uma vez que os métodos foram executados 10 vezes em cada cliente, obteve-se o conjunto total de execuções na avaliação conduzida: 5 métodos x 10 execuções x 5 clientes = 250 soluções.

A Figura 29 apresenta um gráfico *box plot* com a mediana, indicada pela linha no centro do retângulo, a dispersão, dada pela amplitude das linhas verticais do gráfico, e os *outliers* que são os pontos distantes. O MPGA apresentou os resultados mais confiáveis, ou seja, chegou ao melhor resultado em todas as execuções. A busca tabu obteve desempenho quase tão bom quanto o MPGA, mas apresentou algumas soluções discrepantes. SA não apresentou desempenho

estável. Na Tabela 16, é possível observar a taxa de sucesso das soluções ótimas e subótimas referentes às soluções encontradas por cada método, neste contexto, uma solução subótima é uma solução que não é a ótima, mas possui um valor de *fitness* equivalente a ótima.

Figura 29 – Análise de confiabilidade dos métodos.

Uma vez que o MPGA e TS foram os algoritmos com os melhores resultados, uma análise do tempo de convergência desses métodos foi conduzida a partir de cinco clientes aleatoriamente selecionados (os mesmos da análise anterior). Nas Figuras 30 e 31 são apresentadas as análises de convergência do MPGA e TS para esses clientes. O tempo de convergência variou entre 45 e 135 segundos, um tempo bem abaixo dos 5 minutos definidos como critério de parada. Além disso, esse tempo é significativamente menor que as 3,5 horas obtidas pelo método determinístico. Por outro lado, TS obteve um pior resultado de convergência ao não encontrar a solução ótima para

Tabela 16 – Taxa de sucesso dos métodos.

Clientes	Método	Taxa de sucesso ótima (%)	Taxa de sucesso subótima (%)
A	MPGA	100	0
	SA	10	30
	TABU	80	20
B	MPGA	90	10
	SA	0	20
	TABU	70	20
C	MPGA	100	0
	SA	20	20
	TABU	80	20
D	MPGA	100	0
	SA	20	20
	TABU	100	0
E	MPGA	100	0
	SA	30	0
	TABU	80	0

os clientes 1 e 3.

Figura 30 – Análise de convergência do MPGA.

Outra avaliação realizada foi em relação a redução dos custos pagos pelos clientes

Figura 31 – Análise de convergência do TS.

para respeitar os atributos do SLA. Primeiramente, foram selecionados 8 clientes dos 100 gerados anteriormente. Esses clientes contrataram os recursos seguindo a metodologia utilizada por provedores de serviços. Para isso, a arquitetura desenvolvida por [Batista et al. \(2015\)](#) foi utilizada, mas sem o gerenciamento otimizado de recursos.

Na Tabela 17 são apresentados os resultados obtidos dessa simulação, onde é considerado que os recursos são infinitos e todas as requisições são atendidas como proposto em [Batista et al. \(2015\)](#). A primeira coluna corresponde à identificação do cliente e a segunda à configuração (número de VMs) contratada, sendo do tipo *[small, medium, large]*. A terceira coluna representa a carga de trabalho que é dada em MIPS, a unidade utilizada pelo CloudSim, enquanto a coluna Tempo corresponde ao tempo de resposta da execução da aplicação do cliente na infraestrutura. A coluna Custo corresponde ao valor pago por hora pelo cliente. A última linha da tabela lista a quantidade total de recursos consumidos na infraestrutura e o somatório do custo de todos os clientes.

Em uma segunda execução, os algoritmos determinístico e MPGA foram aplicados. Nesse caso, o tempo obtido por cada cliente na Tabela 17 seria mantido, mas o custo deveria ser reduzido. Logo, apenas os recursos necessários de acordo com suas necessidades de tempo seriam fornecidos. Nesse experimento, também foi considerada uma quantidade infinita de recursos. Os algoritmos determinístico e MPGA obtiveram as mesmas soluções e para isso, foi explorado o espaço de busca para o conjunto de VMs *[100,100,100]*. O MPGA foi executado

Tabela 17 – Simulação em uma arquitetura sem gerenciamento otimizado de recursos, considerando recursos infinitos.

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (U\$)
A	[0,50,0]	750000	30	4,97
B	[0,40,0]	900000	45	3,9764
C	[50,0,0]	600000	48	3,067
D	[0,0,50]	1250000	20,8	0,94
E	[0,0,30]	1300000	43,33	4,723
F	[0,20,0]	800000	80	1,9882
G	[0,40,0]	1300000	65	3,9764
H	[50,0,0]	900000	72	3,0674
TOTAL	[100,150,90]			26,7084

apenas uma vez por 3 minutos, contudo, obteve-se a solução ótima para todas as execuções em menos de 2 minutos. O método determinístico ultrapassou o tempo de 3h. Os resultados são apresentados na Tabela 18.

Tabela 18 – Simulação em uma arquitetura aplicando os algoritmos de otimização, considerando recursos infinitos.

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (U\$)
A	[0,0,25]	750000	30	3,93
B	[0,0,20]	900000	45	3,14
C	[0,25,0]	600000	48	2,485
D	[0,0,50]	1250000	20,8	0,94
E	[0,0,30]	1300000	43,33	4,723
F	[0,0,10]	800000	80	1,574
G	[0,0,20]	1300000	65	3,1489
H	[0,25,0]	900000	72	2,4852
TOTAL	[0,50,130]			21,6361

Ao comparar as Tabelas 17 e 18, é possível observar que houve uma redução considerável na quantidade de recursos consumidos pelos clientes, assim como o custo pago por eles. Uma vez que as configurações retornadas pelo MPGA foram as mesmas do método determinísticos, pode-se afirmar que a meta-heurística retornou a solução ótima. Os dois métodos conseguiram atender ao critério de tempo estabelecido, reduzindo os custos relacionados.

De modo a avaliar o comportamento dos algoritmos e da arquitetura, foram realizadas as mesmas execuções considerando uma quantidade limitada de recursos. Assim, torna-se possível avaliar o comportamento da arquitetura em um ambiente real de *cloud*, onde os recursos não são ilimitados. Considerando os tipos de máquinas [*small, medium, large*], observa-se pelos resultados da Tabela 18 que máquinas do tipo *small* não foram utilizadas. Logo, o limite adotado para os recursos foi [0,50,100], mantendo-se a quantidade de máquinas do tipo *medium* e reduzindo as máquinas *large* para gerar alguma competição por recursos. A Tabela 19 apresenta os resultados obtidos, onde a coluna **Atendido** indica se o cliente foi atendido ou não (*true* ou *false*) e, a

coluna **QUEUE** indica *false*, se o cliente foi atendido na sua requisição e *true*, caso contrário. Isso ocorre quando o algoritmo de otimização retorna um conjunto de VMs para o cliente cujo atendimento não é mais possível, uma vez que as máquinas já foram contratadas por outros clientes. Nesse caso, o cliente retorna para o início da fila e o algoritmo de otimização é executado novamente. Agora, o algoritmo de otimização (MPGA) é executado com o número atualizado de VMs disponíveis.

Observe que as negociações de SLA são realizadas simultaneamente para um conjunto de clientes. Logo, a otimização do SLA a ser fechada com os clientes se inicia no mesmo instante. Todavia, um cliente pode ter a otimização do seu SLA concluída antes dos demais e sua demanda por máquinas será atendida imediatamente. Logo, se uma outra otimização de SLA demandar mais máquinas para um cliente do que as atualmente disponíveis, esse cliente entrará no próximo turno de negociação.

Tabela 19 – Simulação em uma arquitetura aplicando o MPGA, considerando recursos limitados.

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h U\$	QUEUE	Atendido
A	[0,0,25]	750000	30	3,93	false	true
B	[0,0,20]	900000	45	3,149	false	true
C	[0,25,0]	600000	48	2,4853	false	true
D	[0,0,25]	1250000	50	3,93	true	true
E	[0,15,0]	1300000	173	1,49	true	true
F	[0,0,10]	800000	80	1,5745	false	true
G	[0,0,20]	1300000	65	3,149	false	true
H	[0,10,0]	900000	180	0,99	true	true
TOTAL	[0,50,100]			20,6978		

O fato dos clientes não serem atendidos é consequência da disputa por recursos, uma vez que eles não são ilimitados e há grande demanda. O passo a passo da execução e atendimento dos clientes é apresentado na Figura 32.

No passo 3 da Figura 32, os clientes A, B, C, F e G, obtiveram o resultado da execução dos algoritmos antes dos Clientes D, E e H, portanto, foi possível alocar o número de VMs requisitado. Quando os clientes D, E e H obtiveram o resultado da execução, o número requisitado já não estava mais disponível, pois foram alocados para outros clientes. Logo, os clientes não atendidos voltam para o processo de negociação. O mesmo acontece no passo 5 para os clientes E e H.

Apesar da disputa e quantidade limitada de recursos, é possível observar que o MPGA fornece soluções para os clientes. O método não atende exatamente as necessidades demandadas pelos clientes, mas aproximará o máximo possível dentro das limitações da infraestrutura. O mesmo pode ser observado nas Tabelas 20 e 21, em que são apresentados resultados da mesma execução com uma quantidade inferior de VMs. A Tabela 20 considera um total de VMs com [50,50,50], enquanto na Tabela 21 esse valor é reduzido para [25,25,25].

Passos	Clientes não Atendidos	Clientes em Processo de negociação	Clientes Atendidos [VM ₁ , VM ₂ , VM ₃]	VMs disponíveis [VM ₁ , VM ₂ , VM ₃]
1	A, B, C, D, E, F, G, H			[0,50,100]
2		A, B, C, D, E, F, G, H		[0,50,100]
3	D, E, H		A[0,0,25], B[0,0,20], C[0,25,0], F[0,0,10], G[0,0,20]	[0,25,25]
4		D, E, H		[0,25,25]
5	E, H		D[0,0,25]	[0,25,0]
6		E, H		[0,25,0]
7			E[0,15,0] H[0,10,0]	[0,0,0]

Figura 32 – Comportamento da arquitetura com recursos limitados.

Tabela 20 – Simulação em uma arquitetura aplicando o MPGA, considerando o conjunto de VMs [50,50,50].

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h U\$	QUEUE	Atendido
A	[0,50,1]	750000	30,00	4,971	true	true
B	[0,0,0]	900000	0,00	0,000	true	false
C	[0,0,0]	600000	0,00	0,000	true	false
D	[0,0,49]	1250000	25,51	7,715	false	true
E	[0,0,0]	1300000	0,00	0,000	true	false
F	[0,0,0]	800000	0,00	0,000	true	false
G	[33,0,0]	1300000	157,58	2,025	true	true
H	[17,0,0]	900000	240,00	0,920	true	true
TOTAL	[48,50,50]			15,630		

Na Figura 33, é possível seguir passo a passo o gerenciamento de recursos em relação ao atendimento dos clientes listados na Tabela 20. Neste caso, como se manteve o número de cliente e foi reduzido o poder computacional do provedor, houve maior disputa por recursos. Os clientes retornaram mais vezes para a fila, como é o caso do cliente “H” que voltou três vezes para a fila antes de ser atendido. Também é possível observar que metade dos clientes não foram atendidos devido ao esgotamento dos recursos. Por outro lado, enquanto haviam recursos, os clientes foram atendidos.

De forma análoga à Figura 33, é possível observar na Figura 34 o passo a passo das negociações entre clientes da Tabela 21 e o provedor. Neste caso, devido a quantidade menor de recursos, em comparação com os experimentos anteriores, um número menor de clientes pode ser atendido. Isso levou ao atendimento de apenas um cliente por vez. Além disso, é possível observar que o cliente “F” retornou duas vezes para a fila antes de ser atendido.

Passos	Clientes não Atendidos	Clientes em Processo de negociação	Clientes Atendidos [VM ₁ , VM ₂ , VM ₃]	VMs disponíveis [VM ₁ , VM ₂ , VM ₃]
1	A, B, C, D, E, F, G, H			[50,50,50]
2		A, B, C, D, E, F, G, H		[50,50,50]
3	A, B, C, E, F, G, H		D[0,0,49]	[50,50,1]
4		A, B, C, E, F, G, H		[50,50,1]
5	B, C, E, F, G, H		A[0,50,1]	[50,0,0]
6		B, C, E, F, G, H		[50,0,0]
7	B, C, E, F, H		G[33,0,0]	[17,0,0]
8		B, C, E, F, H		[17,0,0]
9	B, C, E, F		H[15,0,0]	[2,0,0]
10	B, C, E, F			[2,0,0]

Figura 33 – Comportamento da arquitetura com recursos limitados para o conjunto de VMs [50,50,50].

Tabela 21 – Simulação em uma arquitetura aplicando o MPGA, considerando o conjunto de VMs [25,25,25].

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h U\$	QUEUE	Atendido
A	[1,1,25]	750000	30,00	3,936	false	true
B	[0,0,0]	900000	0,00	0,000	true	false
C	[0,0,0]	600000	0,00	0,000	true	false
D	[0,20,0]	1250000	125,834	1,988	true	true
E	[0,0,0]	1300000	0,00	0,000	true	false
F	[24,4,0]	800000	133,33	1,472	true	true
G	[0,0,0]	1300000	0,00	0,000	true	false
H	[0,0,0]	900000	0,00	0,000	true	false
TOTAL	[25,25,25]			6,999		

6.2.3.4 Resultados obtidos no cenário 4

As Tabelas 22 e 23 apresentam os resultados obtidos considerando toda a arquitetura proposta: estabelecimento e monitoramento do SLA. Na primeira avaliação, os recursos foram considerados ilimitados e foi aplicado o MPGA para obter as soluções, sendo executado apenas uma vez por um minuto na fase de estabelecimento do SLA, e reexecutado na ocorrência de dez ou mais violações para o mesmo cliente durante o monitoramento.

A Tabela 22 possui uma coluna a mais que as tabelas descritas na Seção 6.2.3.3 e a Tabela 23 possui duas colunas a mais. Essas duas tabelas possuem em comum a coluna adicional

Passos	Clientes não Atendidos	Clientes em Processo de negociação	Clientes Atendidos [VM ₁ , VM ₂ , VM ₃]	VMs disponíveis [VM ₁ , VM ₂ , VM ₃]
1	A, B, C, D, E, F, G, H			[25,25,25]
2		A, B, C, D, E, F, G, H		[25,25,25]
3	B, C, D, E, F, G, H		A[1,1,25]	[24,24,0]
4		B, C, D, E, F, G, H		[24,24,0]
5	B, C, E, F, G, H		D[0,20,0]	[24,4,0]
6		B, C, E, F, G, H		[24,4,0]
7	B, C, E, G, H		F[24,4,0]	[0,0,0]

Figura 34 – Comportamento da arquitetura com recursos limitados para o conjunto de VMs [25,25,25].

“Passo” que representa o instante de tempo² em que o cliente foi atendido. Isso significa que o SLA foi fechado e o cliente passa a executar sua aplicação, tendo início a fase de monitoramento. A coluna adicional na Tabela 23 indica as violações no contrato de cada cliente que foram monitoradas. Essas violações são consequência da carga de trabalho aplicada, cuja geração está descrita na Seção 6.2.2.4. Os clientes com tais violações estão assinalados pelas linhas cinzas em cada Tabela.

Tabela 22 – Simulação em uma arquitetura aplicando os algoritmos de otimização na fase de estabelecimento de contrato, considerando recursos ilimitados.

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (U\$)	QUEUE	Passo
A	[1,50,0]	750000	30,00	4,971	false	1
B	[1,40,1]	900000	45,00	3,976	false	1
C	[50,5,5]	600000	48,00	3,067	false	1
D	[0,60,0]	1250000	41,67	5,965	false	1
E	[1,60,0]	1300000	43,33	5,965	false	1
F	[40,0,5]	800000	80,00	2,454	false	1
G	[0,1,20]	1300000	65,00	3,149	false	1
H	[50,4,2]	900000	72,00	3,067	false	1
TOTAL	[143,220,33]			32,614		

A Tabela 22 mostra que no instante de tempo “1” todos os clientes fecharam o contrato e não houve retorno de clientes para a fila, pois todos os recursos requisitados estavam disponíveis. Na tabela 23, observa-se que todos os clientes executaram no instante de tempo 2 sem violações, exceto pelos clientes C e F. Assim, a fase de monitoramento detecta tais violações e inicia a

² O valor do tempo refere-se apenas ao que está acontecendo naquele instante, ele não representa uma unidade de medida de tempo como segundos ou minutos.

Tabela 23 – Simulação aplicando os algoritmos de otimização na fase de monitoramento dos SLAs, considerando recursos ilimitados.

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (US\$)	QUEUE	Passo	Violações
A	[0,50,2]	750000	30,00	4,971	false	2	0
B	[60,7,19]	900000	60,00	3,681	false	2	0
C	[50,7,4]	1080524	83,53	3,067	false	3	100
D	[0,0,60]	1250000	20,83	9,447	false	2	0
E	[0,60,0]	1300000	43,33	5,965	false	2	0
F	[0,20,0]	1058384	105,65	1,989	false	3	79
G	[0,40,7]	1300000	65,00	3,976	false	2	0
H	[50,24,6]	900000	72,00	3,067	false	2	0

renegociação do contrato. Uma vez alterado o SLA, a aplicação do cliente passa a ser novamente executada como indicado no instante de tempo 3.

O próximo experimento repete o anterior, mas considerando um *data center* com recursos limitados de [200,200,200] VMs para [*small,medim,large*]. Os resultados dessa simulação são apresentados nas Tabelas 24 e 25.

Tabela 24 – Simulação em uma arquitetura aplicando os algoritmos de otimização na fase de **estabelecimento** de contrato, com o conjunto de VMs [200,200,200].

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (US\$)	QUEUE	Passo
A	[0,50,2]	750000	30,00	4,971	false	1
B	[60,7,19]	900000	60,00	3,681	false	1
C	[50,7,4]	600000	48,00	3,067	true	1
D	[0,0,60]	1250000	20,83	9,447	false	1
E	[0,60,0]	1300000	43,33	5,965	false	1
F	[40,13,3]	800000	80,00	2,454	true	4
G	[0,40,7]	1300000	65,00	3,976	true	3
H	[50,24,6]	900000	72,00	3,067	true	3
TOTAL	[]			36,6288176		

Ao simular um cenário com recursos limitados, é possível observar que alguns clientes necessitaram voltar para a fila e esperar que os recursos fossem liberados. O mesmo comportamento ocorreu durante a fase de monitoramento, onde houve a renegociação do contrato. Na Tabela 26, é possível compreender melhor o fluxo dessa execução. Os estágios descritos em cada coluna da Tabela 26 são os mesmos descritos anteriormente nas Figura 13. A Tabela 27 apresenta o estado de cada cliente para cada momento da execução, onde os seguintes estados são considerados: (0) aguardando fechamento do contrato, (1) contrato negado, (2) contrato fechado, (3) executando, (4) renegociando contrato e (5) finalizado.

O experimento com recursos limitados foi realizado novamente, reduzindo a quantidade de recursos para um conjunto de [100,100,100] VMs para [*small,medim,large*]. Dessa forma

Tabela 25 – Simulação aplicando os algoritmos de otimização na fase de **monitoramento** dos SLAs, com o conjunto de VMs [200,200,200].

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (US\$)	QUEUE	Passo	Violações
A	[0,50,2]	750000	30,00	4,971	false	2	0
B	[60,7,19]	900000	60,00	3,681	false	2	0
C	[50,7,4]	1080524	83,53	3,067	true	3	100
D	[0,0,60]	1250000	20,83	9,447	true	2	0
E	[0,60,0]	1300000	43,33	5,965	false	2	0
F	[0,20,0]	1058384	105,65	1,989	true	5	79
G	[0,40,7]	1300000	65,00	3,976	true	4	0
H	[50,24,6]	900000	72,00	3,067	true	4	0

Tabela 26 – Fluxo de execução e negociação de cada cliente.

Passo	Estabelecendo SLA (Fase1)	Monitorando (Fase 2)	Renegociando SLA (Fase3)	Finalizando SLA (Fase 4)	Capacidade Total da Infraestrutura
0	A,B,C,D,E,F,G,H				[200,200,200]
1	F,G,H	A,B,C,D,E			[149,60,94]
2	F,G,H	A,B,D,E	C		[149,17,40]
3		C,F,G,H		A,B,D,E	[60,99,117]
4		G,H	F	C	[110,106,121]
5		F		G,H	[200,180,200]
6				F	[200,200,200]

Tabela 27 – Estado de cada cliente durante o fluxo de execução.

Passo	A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0	0
1	2	2	2	2	2	1	1	1
2	3	3	4	3	3	1	1	1
3	5	5	3	5	5	2	2	2
4	5	5	5	5	5	4	3	3
5	5	5	5	5	5	3	5	5
6	5	5	5	5	5	5	5	5

haverá uma disputa ainda maior pelos recursos. Os resultados dessa simulação são apresentados nas Tabelas 28 e 29.

Assim como no experimento anterior, alguns clientes retornaram para a fila e esperaram que os recursos fossem liberados. O mesmo ocorre durante a fase de monitoramento (contraste da coluna “Passo” das Tabelas 28 e 29), em que há a renegociação do contrato. Além disso, como os recursos foram reduzidos, alguns clientes permaneceram mais tempo na fila, esperando que o recursos fossem liberados como foi o caso dos clientes D, F, G e H. Outro ponto a ser observado é que para alguns clientes (F e G) foram alocados um número menor de recursos. Isso se deve à quantidade de recursos disponíveis naquele momento e ao custo geral total dos clientes que foi reduzido. Na Tabela 30, é possível compreender melhor o fluxo dessa execução. A Tabela 31 apresenta o estado de cada cliente para cada momento da execução, onde os seguintes estados são considerados: (0) aguardando fechamento do contrato, (1) contrato negado, (2) contrato

Tabela 28 – Simulação em uma arquitetura aplicando os algoritmos de otimização na fase de **estabelecimento** de contrato, com o conjunto de VMs [100,100,100].

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (U\$)	QUEUE	Passo
A	[0,50,1]	750000	30,00	4,971	false	1
B	[60,1,2]	900000	60,00	3,681	false	1
C	[40,1,13]	600000	55,77	2,454	true	2
D	[0,48,3]	1250000	45,00	5,872	true	4
E	[0,47,1]	1300000	55,32	4,672	false	1
F	[0,0,10]	800000	124,77	1,574	true	4
G	[0,0,20]	1300000	65,00	3,149	true	4
H	[50,8,0]	900000	75,00	3,067	true	5
TOTAL				29,44146384		

Tabela 29 – Simulação aplicando os algoritmos de otimização na fase de **monitoramento** dos SLAs, com o conjunto de VMs [100,100,100].

Cliente	Configuração [s,m,l]	Carga (MIPS)	Tempo (s)	Custo/h (U\$)	QUEUE	Passo	Violações
A	[0,50,1]	750000	30,00	4,971	false	2	0
B	[60,1,2]	900000	60,00	3,681	false	2	0
C	[40,1,13]	600000	124,711	2,454	true	4	100
D	[0,50,3]	1250000	45,00	5,872	true	5	0
E	[0,47,1]	1300000	55,32	4,672	false	2	0
F	[0,0,10]	800000	124,77	1,574	true	6	76
G	[0,0,20]	1300000	65,00	3,149	true	5	0
H	[50,8,0]	900000	75,00	3,067	true	6	0

fechado, (3) executando, (4) renegociando contrato e (5) finalizado.

Tabela 30 – Fluxo de execução e negociação de cada cliente para um conjunto de [100,100,100] VMs.

Passo	Estabelecendo SLA (Fase1)	Monitorando (Fase 2)	Renegociando SLA (Fase3)	Finalizando SLA (Fase 4)	Capacidade Total da Infraestrutura
0	A,B,C,D,E,F,G,H				[100,100,100]
1	C,D,F,G,H	A,B,E			[40,1,78]
2	D,F,G,H	A,B,C,E			[0,1,65]
3	D,F,G,H		C	A,B,E	[60,99,87]
4	H	C,D,F,G			[60,51,54]
5		D,F,G,H	F	C	[50,44,67]
6		F,H		D,G	[50,92,90]
7				F,H	[100,100,100]

6.2.3.5 Resultados obtidos no cenário 5

Como descrito na Seção 3.5, o problema de provisionamento de recursos é caracterizado como um problema multiobjetivo. Nos experimentos anteriores, os valores de tempo e custo possuíam os mesmos pesos. Nesta seção, os resultados serão avaliados considerando os objetivos conflitantes relacionados ao baixo custo e reduzido tempo computacional.

Tabela 31 – Estado de cada cliente durante o fluxo de execução para um conjunto de [100,100,100] VMs.

Passo	A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0	0
1	2	2	1	1	2	1	1	1
2	3	3	2	1	3	1	1	1
3	5	5	4	1	5	1	1	1
4	5	5	3	2	5	2	2	1
5	5	5	5	3	5	3	3	2
6	5	5	5	5	5	3	5	3
7	5	5	5	5	5	5	5	5

Na Figura 35, são apresentados 4 curvas de Pareto (em vermelho) para quatro clientes (W,X,Y,Z) respectivamente com a aplicação de repositório de arquivos (Apache), e na Tabela 32 são apresentadas as três soluções obtidas de cada curva, uma priorizando o tempo, outra o custo e outra que prioriza a minimização de ambos os termos, denominada solução compromisso. Esse experimento consistiu na execução do algoritmo NSGA-II por 10 vezes durante 6 minutos para cada cliente. O tempo e custo foram utilizados na função objetivo conforme descrito na Seção 6.2.2.5. Nos gráficos são apresentadas as curvas obtidas.

Figura 35 – Curva de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Apache.

De forma análoga, o experimento foi repetido para clientes que optaram por usar aplicações de processamento de imagens (Smallpt). Os resultados são apresentado na Figura 36 e na

Tabela 32 – Soluções geradas alternando a prioridade de cada parâmetro para os clientes W, X, Y, Z com benchmark Apache.

Cliente	Prioridade	Capacidade	tempo	custo
W	Tempo	0-14-0	10,7	19,48
	Custo	1-0-0	300	0,0613
	Compromisso	0-0-4	18,8	2,5192
X	Tempo	1-0-10	7,50	17,32
	Custo	1-0-0	300	0,0613
	Compromisso	0-0-2	37,5	0,6200
Y	Tempo	0-0-12	10,41675	22,672
	Custo	1-0-0	500	0,0613
	Compromisso	0-0-3	41,667	1,417
Z	Tempo	0-0-11	11,363	19,051
	Custo	1-1-2	500	0,8088
	Compromisso	11-0-1	45,45	8,098

Tabela 33. Os dois experimentos revelam que, independente do tipo de aplicação executada pelo cliente, o NSGA-II oferece soluções de acordo com as necessidades do cliente. Além disso, com a fronteira de Pareto é possível obter um *trade-off* (relação de compromisso) entre os objetivos conflitantes.

Nos experimentos executados neste cenário, foram selecionadas três tipos de soluções a partir da fronteira de pareto. Todavia, é possível aplicar pesos aos objetivos de acordo com a preferência do cliente. Por exemplo, ele poderia priorizar o custo e abrir mão de 20% (ou qualquer percentual) do tempo de resposta para isso, ou vice-versa. Dessa forma é possível escolher uma solução da fronteira de Pareto de acordo com a preferência e necessidade do cliente.

Tabela 33 – Soluções geradas alternando a prioridade de cada parâmetro para os cliente W, X, Y, Z com benchmark Smallpt.

Cliente	Prioridade	Capacidade	tempo	custo
W	Tempo	0-14-0	17,8572	19,4848
	Custo	3-0-0	500	0,5521
	Compromisso	0-0-3	45,71	1,417
X	Tempo	0-0-11	11,3637	19,0514
	Custo	0-1-0	500	0,0994
	Compromisso	0-0-2	62,5	0,6297
Y	Tempo	0-1-11	15	20,7833
	Custo	0-1-0	330	0,0994
	Compromisso	0-0-3	55	1,417
Z	Tempo	0-0-11	15,05	19,0514
	Custo	0-0-1	165	0,1574
	Compromisso	0-0-3	55	1,417

Outra análise realizada foi em relação a quantidade de fronteiras geradas durante a

Figura 36 – Curva de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Smallpt.

execução. Neste experimento o NSGA-II foi executado 10 vezes por 6 minutos. Nos gráficos da Figura 37 são apresentados os resultados para os clientes W, X, Y e Z (da esquerda para a direita) que utilizaram o modelo de serviço Apache, e nos gráficos da Figura 38, são apresentados os resultados para os clientes que utilizaram o modelo de serviço Smallpt. É possível observar que, a quantidade de fronteiras geradas durante a execução possui crescimento linear e a medida que o tempo aumenta, consequentemente a qualidade das soluções também aumenta.

6.3 Considerações Finais

Neste capítulo foram apresentados os resultados obtidos durante o desenvolvimento deste projeto de mestrado. Os resultados apresentados demonstram que algoritmos evolutivos são promissores na solução do problema abordado, capazes de encontrarem a solução ótima ou sub-ótima em um tempo satisfatório. Foram avaliados também a qualidade das soluções geradas, constatando-se que houve redução no custo pago pelo cliente e mantendo o tempo de resposta exigido e, consequentemente evitando violações. Além disso, foi avaliado a utilização de uma meta-heurística multi-objetiva, o NSGA-II.

No Cenário 1, foram executados experimentos iniciais, com objetivo de avaliar a utilização de meta-heurísticas para o problemas. Neste cenário foi implementado o μ AG, onde

Figura 37 – Quantidade de fronteiras de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Apache.

obteve-se um taxa de sucesso baixa, contudo, apresentando boas solução.

No Cenário 2, foi reduzido o espaço de busca para o μAG , chegando a uma taxa de sucesso de 100%. Contudo, constatou-se que com uso de outra meta-heurística, o MPGA, o qual obteve melhor resultado. Portanto, para os outros cenário optou-se por investigar outras meta-heurísticas.

Para o Cenário 3, foram implementados outras duas meta-heurísticas além do MPGA, o *simulated annealing* e o *tabu search*, ná análise comparativa o MPGA obteve desempenho superior as demais, obtendo a solução ótima em todos os caso e um tempo menor. Em uma análise de cumprimento de acordo e redução de custo, ao compara com outros trabalhos, obteve-se uma redução de 18.9% do custo cumprindo o acordo fechado.

No Cenário 4, vou avaliado todo o sistema, onde foi criada uma fila de clientes. Foi possível avaliar o sistema em tempo de execução, com quantidade limitada e ilimitada de recursos. Neste Cenário, o *framework* gerou soluções para todos os cliente, atendendo a todos de maneira a cumprir como SLA.

Por fim, no Cenário 5, foi aplicado o NSGA-II, uma meta-heurística multi-objetiva. Neste Cenário, com a aplicação do NSGA-II foi possível dar várias opções de SLA parao cliente, possibilitando-o priorizar qualquer um dos fatores conflitantes ou ambos.

Figura 38 – Quantidade de fronteiras de Pareto gerada para os cliente W, X, Y, Z (da esquerda para a direita) com o benchmark Smallpt.

CONCLUSÃO

O paradigma da computação em nuvem acarretou uma grande mudança no contexto de como os recursos computacionais são oferecidos atualmente. Devido ao fato desses recursos serem oferecidos sob demanda, é possível obter maior flexibilidade e escalabilidade em sua aquisição. Por esse motivo, empresas, universidades e até mesmo o governo tem optado pela aquisição de recursos em nuvem a fim de reduzir o custo gasto e espaço ocupado. Um problema em aberto neste contexto é o cumprimento de acordos de níveis de serviços (SLA).

Fornecer ao cliente da nuvem uma infraestrutura eficiente, respeitando o SLA e seus atributos de QoS, e ainda tentando minimizar o custo, não é uma tarefa trivial. Dentro do contexto da computação em nuvem, os mais diversos problemas podem ser mapeados em soluções que geralmente envolvem otimização devido a complexidade e a grande quantidade de recursos que podem ser escaláveis sob demanda.

Este trabalho de mestrado abordou um destes desafios que foi fornecer ao cliente uma infraestrutura respeitando o SLA acordado entre o cliente e provedor. A nossa proposta mapeou alguns atributos de QoS que determinam os critérios do SLA. Também foram desenvolvidos e analisados algoritmos que fornecem uma configuração otimizada da infraestrutura baseada em tais critérios. Os resultados demonstraram que os algoritmos baseados nas meta-heurísticas como *Tabu search* e MPGA são aplicáveis à solução do problema.

Inicialmente foi elaborado um método determinístico capaz de encontrar a solução ótima para as instâncias do problema. Em comparação com as metaheurísticas desenvolvidas, observou-se que o MPGA foi capaz de chegar a solução ótima em frações do tempo obtido pelo determinístico, levando 0,01% do tempo obtido pelo determinístico.

Nos resultados obtidos no Cenário 3, no qual é analisado um contraste com a arquitetura sem os algoritmos de otimização (BATISTA *et al.*, 2015), observou-se que utilizando o *framework* de otimização, manteve-se a QoS contratada pelo cliente e ainda obteve-se uma redução de 18,99% do custo gasto por ele, e que, mesmo utilizando um ambiente com recursos limitados,

foi possível atender aos clientes o mais próximo do requisitado.

No Cenário 4, foi realizada uma avaliação completa da arquitetura, desde o estabelecimento ao fechamento do SLA. Com os resultados, constatou-se que as metaheurísticas são capazes de fornecer ao cliente um SLA factível em um tempo hábil. Além disso, quando há violação do contrato, na qual há necessidade de renegociação, o *framework* de otimização foi capaz de fornecer um novo SLA dentro das limitações da infraestrutura.

Pelos motivos elencados na Seção 3.5, o problema abordado neste trabalho também é classificado como um problema multiobjetivo. Portanto, outra análise foi realizada considerando o NSGA-II, uma meta-heurística multi-objetiva. Nesta análise foi possível observar e oferecer ao clientes variações das soluções geradas, podendo assim, utilizar pesos aos critério de SLA e, escolher a melhor solução de acordo com o contexto, o problema e a preferência do cliente.

PUBLICAÇÕES E TRABALHOS FUTUROS

Durante a vigência do projeto, algumas publicações foram geradas. Os trabalhos descritos são relacionados diretamente com este projeto de mestrado e com outros projetos de doutorado do grupo de pesquisa. Além das publicações geradas, estão previstos a escrita de mais dois artigos científico: o primeiro deve considerar uma experimentação mais detalhadas da abordagem multi-objetiva e o segundo abordará a fase de monitoramento do SLA. A seguir são descritos os artigos oriundos deste projeto:

- AZEVEDO, L. J. M.; ESTRELLA, J. C. ; NAKAMURA, L. H. V. ; SANTANA, M. J. ; SANTANA, R. H. C. ; TOLEDO, C. F. M. ; Batista, B. G ; REI-MARGANIEC, S. . Optimized Service Level Agreement Establishment in Cloud Computing. *COMPUTER JOURNAL (ONLINE)*, 2017. (Artigo em periódico/Qualis A2/) - <https://doi.org/10.1093/comjnl/bxx087>
- NAKAMURA, L. H. V. ; AZEVEDO, L. J. M. ; ESTRELLA, J. C. ; TOLEDO, C. F. M. ; Batista, B. G ; REI-MARGANIEC, S. ; MENEGUETTE, R. I. ; SANTANA, R. H. C. ; SANTANA, M. J. . An Analysis of Optimization Algorithms designed to fully comply with SLA in Cloud Computing. *IEEE Latin America Transactions*, v. 15, p. 1497-1505, 2017. (Artigo em periódico/Qualis B4) - <https://doi.org/10.1109/TLA.2017.7994798>
- AZEVEDO, L. J. M.; ESTRELLA, J. C. ; TOLEDO, C. F. M. ; REI-MARGANIEC, S. . An Analysis of Metaheuristic to SLA Establishment in Cloud Computing. In: *ZEUS*, 2017, Lugano. 9th Central European Workshop on Services and their Composition, 2017. v. 1. p. 79-84. (Artigo em workshop) - <http://ceur-ws.org/Vol-1826/paper14.pdf>
- AZEVEDO, L. J. M.; ESTRELLA, J. C. ; TOLEDO, C. F. M. . Desenvolvimento e avaliação de algoritmos de otimização para o cumprimento de acordos de níveis de serviços em nuvem. In: *1o Encontro Paulista de Pós-graduandos em Computação*, 2017, São Carlos - SP. EPPC, 2017. (Apresentação de Trabalho/Conferência ou palestra).

- AZEVEDO, L. J. M.; Batista, B. G ; ESTRELLA, J. C. . Avaliação da aplicação de meta-heurísticas para o problema de provisionamento de recursos. In: 7a Escola Regional de Alto Desempenho de São Paulo, 2016, São Paulo. ERAD-SP, 2016. (Apresentação de Trabalho/Conferência ou palestra).

Durante o desenvolvimento do projeto de mestrado, outros trabalhos foram publicados em parceria com docentes e alunos do grupo de pesquisa, os quais são descritos a seguir:

- SHISHIDO, H. Y. ; AZEVEDO, L. J. M. ; ESTRELLA, J. C. . Estudo Comparativo de Desempenho e Consumo de Energia em Arquiteturas de Alto Desempenho. In: 7a Escola Regional de Alto Desempenho de São Paulo, 2016, São Paulo. ERAD-SP, 2016. (Apresentação de Trabalho/Conferência ou palestra/trabalho premiado como melhor poster de pós-graduação).
- SHISHIDO, H. Y. ; AZEVEDO, L. J. M. ; SOUZA, P. S. L. ; ESTRELLA, J. C. ; BRUSCHI, S. M. . Basoper: a Web-based Open Educational Resource for the Teaching of Scheduling Algorithms for Batch Systems. In: TISE: Congreso Internacional de Informática Educativa, 2015, Santiago. XX Conferência Internacional sobre Informática na Educação. Santiago: Universidad de Chile, 2015. v. 11. p. 498-503. (Artigo em conferência/Qualis B4).

Foi iniciada a implantação do *framework* de algoritmos de otimização em um ambiente real, considerando um protótipo desenvolvido por Nakamura (2017). Contudo, não foi possível a realização de uma experimentação robusta, devido a necessidade da refatoração e de algumas parametrizações da arquitetura. Alguns testes foram executados, entretanto, constatou-se a necessidade de levar outros fatores em consideração pelo *framework* de otimização, como por exemplo o uso da CPU. Além disso, durante a fase de monitoramento, constatou-se a necessidade de técnicas de escalonamento e gerenciamento de clientes, pois em um ambiente com recursos limitados os primeiros clientes da fila alocam muitos recursos, enquanto outros não são atendidos. Outras situações, como, acrescentar outros atributos de QoS ao SLA e novas restrições ao problema, como por exemplo, um limite máximo de custo que o cliente consegue pagar, também será objeto de estudo. Em trabalhos futuros, pretende-se concluir o desenvolvimento do *framework* no protótipo e implantar políticas para o gerenciamento de clientes.

REFERÊNCIAS

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

ALMEIDA, V. A.; MENASCÉ, D. Capacity planning an essential tool for managing web services. **IT professional**, IEEE, v. 4, n. 4, p. 33–38, 2002. Citado na página 13.

AMAZON. **Dynamic Scaling**. 2015. Disponível em: <<http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/as-scale-based-on-demand.html>>. Último acesso em: 01/20/2017. Citado na página 35.

_____. **Scaling Based on Metrics**. 2015. Disponível em: <http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html>. Último acesso em: 01/20/2017. Citado na página 34.

_____. **Scaling the Size of Your Auto Scaling Group**. 2015. Disponível em: <http://docs.aws.amazon.com/autoscaling/latest/userguide/scaling_plan.html>. Último acesso em: 01/20/2017. Citado na página 34.

APACHE. **Apache Benchmarking**. 2017. Available from: <<https://openbenchmarking.org/test/pts/apache>>. Accessed: 06-27-2017. Citado nas páginas 47 e 66.

ARANTES, J. da S.; ARANTES, M. da S.; TOLEDO, C. F. M.; JÚNIOR, O. T.; WILLIAMS, B. C. Heuristic and genetic algorithm approaches for UAV path planning under critical situation. **International Journal on Artificial Intelligence Tools**, v. 26, n. 1, p. 1–30, 2017. Citado na página 53.

ARANTES, M. d. S. **Ambiente para desenvolvimento de métodos aplicados a problemas de otimização**. 116 p. Dissertação (Mestrado) — Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2014. Citado na página 51.

BALLARD, D. H. **An introduction to natural computation**. [S.l.]: Mit Press, 1999. Citado na página 22.

BATISTA, B. G. **Modelos de negócio para ambientes de computação em nuvem que consideram atributos de qos relacionados a desempenho e a segurança**. 142 p. Tese (Doutorado) — Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016. Citado nas páginas 15, 43, 48, 49 e 66.

BATISTA, B. G.; ESTRELLA, J. C.; FERREIRA, C. H. G.; FILHO, D. M. L.; NAKAMURA, L. H. V.; REIFF-MARGANIEC, S.; SANTANA, M. J.; SANTANA, R. H. C. Performance evaluation of resource management in cloud computing environments. **PloS one**, Public Library of Science, v. 10, n. 11, p. e0141914, 2015. Citado nas páginas 33, 35, 36, 37, 80 e 95.

- BELLMAN, R. **Dynamic programming**. [S.l.]: Courier Corporation, 2013. Citado na página [20](#).
- BELOGLAZOV, A.; ABAWAJY, J.; BUYYA, R. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. **Future Generation Computer Systems**, v. 28, n. 5, p. 755 – 768, 2012. Citado na página [37](#).
- BLACK, P. E. Manhattan distance. **Dictionary of Algorithms and Data Structures**, US National Institute of Standards and Technology. Retrieved May, v. 18, p. 2012, 2006. Citado na página [53](#).
- BUI, L. T.; ALAM, S. An introduction to multi-objective optimization. **Multi-Objective Optimization in Computational Intelligence: Theory and Practice**, IGI Global Snippet, p. 1–19, 2008. Citado nas páginas [15](#), [29](#), [30](#) e [31](#).
- BURIOL, P. M. F. L.; MOSCATO, P. A new memetic algorithm for the asymmetric traveling salesman problem. **Journal of Heuristics**, Kluwer Academic Publishers, Hingham, MA, USA, v. 10, p. 1381–1231, 2004. Citado na página [53](#).
- BYUN, E.-K.; KEE, Y.-S.; KIM, J.-S.; MAENG, S. Cost optimized provisioning of elastic resources for application workflows. **Future Generation Computer Systems**, v. 27, n. 8, p. 1011 – 1026, 2011. Citado na página [37](#).
- CALHEIROS, R. N.; RANJAN, R.; BUYYA, R. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In: IEEE. **Parallel Processing (ICPP), 2011 International Conference on**. [S.l.], 2011. p. 295–304. Citado na página [12](#).
- CARISSIMI, A. Virtualização: da teoria a soluções. **Minicursos do Simpósio Brasileiro de Redes de Computadores–SBRC**, v. 2008, p. 173–207, 2008. Citado nas páginas [7](#) e [9](#).
- CHARD, K. Drive: A distributed economic meta-scheduler for the federation of grid and cloud systems. Victoria University of Wellington, 2011. Citado na página [10](#).
- CHEN, Y.; LI, X.; CHEN, F. Overview and analysis of cloud computing research and application. In: IEEE. **E-Business and E-Government (ICEE), 2011 International Conference on**. [S.l.], 2011. p. 1–4. Citado na página [1](#).
- CHIEU, T. C.; MOHINDRA, A.; KARVE, A. A.; SEGAL, A. Dynamic scaling of web applications in a virtualized cloud computing environment. In: IEEE. **e-Business Engineering, 2009. ICEBE'09. IEEE International Conference on**. [S.l.], 2009. p. 281–286. Citado na página [6](#).
- COUTINHO, E. F.; SOUSA, F. R. de C.; REGO, P. A. L.; GOMES, D. G.; SOUZA, J. N. de. Elasticity in cloud computing: a survey. **annals of telecommunications-Annales des télécommunications**, Springer, p. 1–21, 2015. Citado na página [34](#).
- DARWIN, C. **On the origin of species**. [S.l.]: London: John Murray, 1859. Citado na página [22](#).
- DEB, K.; AGRAWAL, S.; PRATAP, A.; MEYARIVAN, T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. **Lecture notes in computer science**, Springer, v. 1917, p. 849–858, 2000. Citado nas páginas [26](#) e [28](#).

DEB, K.; PRATAP, A.; AGARWAL, S.; MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. **Evolutionary Computation, IEEE Transactions on**, IEEE, v. 6, n. 2, p. 182–197, 2002. Citado nas páginas 15, 60 e 62.

EDGEWORTH, F. Y. **Mathematical psychics: An essay on the application of mathematics to the moral sciences**. [S.l.]: Kegan Paul, 1881. v. 10. Citado na página 29.

EIBEN, A.; SMITH, J. **Introduction to Evolutionary Computing**. [S.l.]: Springer Berlin Heidelberg, 2007. (Natural Computing Series). Citado na página 55.

EIBEN, A. E.; SMITH, J. E. **Introduction to evolutionary computing**. [S.l.]: Springer, 2003. v. 53. Citado nas páginas 20, 22, 27 e 41.

EMEAKAROHA, V. C.; BRANDIC, I.; MAURER, M.; BRESKOVIC, I. Sla-aware application deployment and resource allocation in clouds. In: IEEE. **Computer Software and Applications Conference Workshops (COMPSACW), 2011 IEEE 35th Annual**. [S.l.], 2011. p. 298–303. Citado nas páginas 36 e 37.

EYRAUD-DUBOIS, L.; LARCHEVÊQUE, H. Optimizing resource allocation while handling sla violations in cloud computing platforms. In: **Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on**. [S.l.: s.n.], 2013. p. 79–87. Citado nas páginas 36 e 37.

FANIYI, F.; BAHSOON, R. A systematic review of service level management in the cloud. **ACM Computing Surveys (CSUR)**, ACM, v. 48, n. 3, p. 43, 2015. Citado nas páginas 15, 19, 16, 17, 33, 40 e 41.

FERGUSON, P.; HUSTON, G. **Quality of service: delivering QoS on the Internet and in corporate networks**. [S.l.]: John Wiley & Sons, Inc., 1998. Citado na página 9.

FRANCA, P. M.; MENDES, A.; MOSCATO, P. A memetic algorithm for the total tardiness single machine scheduling problem. **European Journal of Operational Research**, v. 132, p. 377–2217, 2001. Citado na página 53.

GARG, S. K.; VERSTEEG, S.; BUYYA, R. Smicloud: A framework for comparing and ranking cloud services. In: IEEE. **Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on**. [S.l.], 2011. p. 210–218. Citado na página 10.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & operations research**, v. 13, n. 5, p. 533–549, 1986. Citado nas páginas 21 e 58.

GUO, L.; MCGOUGH, A. S.; AKRAM, A.; COLLING, D.; MARTYNIAK, J.; KRZYNARIC, M. Enabling qos for service-oriented workflow on grid. In: IEEE. **Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on**. [S.l.], 2007. p. 1077–1082. Citado na página 10.

HASSAN, S.; AL-JUMEILY, D.; HUSSAIN, A. J. Autonomic computing paradigm to support system's development. In: IEEE. **Developments in eSystems Engineering (DESE), 2009 Second International Conference on**. [S.l.], 2009. p. 273–278. Citado na página 11.

HO, W.; XU, X.; DEY, P. K. Multi-criteria decision making approaches for supplier evaluation and selection: A literature review. **European Journal of Operational Research**, v. 202, n. 1, p. 16 – 24, 2010. Citado na página 36.

HOLLAND, J. H. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. U Michigan Press, 1975. Citado na página 26.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing—degrees, models, and applications. **ACM Computing Surveys (CSUR)**, v. 40, n. 3, p. 7, 2008. Citado na página 36.

HUU, T. T.; MONTAGNAT, J. Virtual resources allocation for workflow-based applications distribution on a cloud infrastructure. In: IEEE. **Cluster, Cloud and Grid Computing (CC-Grid), 2010 10th IEEE/ACM International Conference on**. [S.l.], 2010. p. 612–617. Citado na página 39.

JAEN-CUELLAR, A. Y.; MORALES-VELAZQUEZ, L.; ROMERO-TRONCOSO, R. de J.; MORIÑIGO-SOTELO, D.; OSORNIO-RIOS, R. A. Micro-genetic algorithms for detecting and classifying electric power disturbances. **Neural Computing and Applications**, Springer, v. 28, n. 1, p. 379–392, 2017. Citado na página 56.

JAIN, R. **The art of computer systems performance analysis**. [S.l.]: John Wiley & Sons, 2008. Citado na página 13.

JIANG, J.; LU, J.; ZHANG, G.; LONG, G. Optimal cloud resource auto-scaling for web applications. In: IEEE. **Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on**. [S.l.], 2013. p. 58–65. Citado na página 36.

JING, S.-Y.; ALI, S.; SHE, K.; ZHONG, Y. State-of-the-art research study for green cloud computing. **The Journal of Supercomputing**, Springer, v. 65, n. 1, p. 445–468, 2013. Citado na página 2.

KESSACI, Y.; MELAB, N.; TALBI, E.-G. A multi-start local search heuristic for an energy efficient {VMs} assignment on top of the opennebula cloud manager. **Future Generation Computer Systems**, v. 36, p. 237 – 256, 2014. ISSN 0167-739X. Special Section: Intelligent Big Data ProcessingSpecial Section: Behavior Data Security Issues in Network Information PropagationSpecial Section: Energy-efficiency in Large Distributed Computing ArchitecturesSpecial Section: eScience Infrastructure and Applications. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S0167739X13001520>>. Citado nas páginas 36 e 37.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. *et al.* Optimization by simulated annealing. **science**, v. 220, n. 4598, p. 671–680, 1983. Citado nas páginas 21 e 57.

KO, J. M.; KIM, C. O.; KWON, I.-H. Quality-of-service oriented web service composition algorithm and planning architecture. **Journal of Systems and Software**, Elsevier, v. 81, n. 11, p. 2079–2090, 2008. Citado nas páginas 40 e 41.

KOUSIOURIS, G.; CUCINOTTA, T.; VARVARIGOU, T. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. **Journal of Systems and Software**, Elsevier, v. 84, n. 8, p. 1270–1291, 2011. Citado nas páginas 34 e 37.

KRISHNAKUMAR, K. Micro-genetic algorithms for stationary and non-stationary function optimization. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Intelligent control and adaptive systems**. [S.l.], 1990. v. 1196, p. 289–297. Citado na página 56.

- KUMBHARE, A.; SIMMHAN, Y.; PRASANNA, V. K. Exploiting application dynamism and cloud elasticity for continuous dataflows. In: ACM. **Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. [S.l.], 2013. p. 57. Citado na página 41.
- KUMBHARE, A. G.; SIMMHAN, Y.; FRINCUI, M.; PRASANNA, V. K. Reactive resource provisioning heuristics for dynamic dataflows on cloud infrastructure. **Cloud Computing, IEEE Transactions on**, IEEE, v. 3, n. 2, p. 105–118, 2015. Citado na página 41.
- L.D., D. B.; KRISHNA, P. V. Honey bee behavior inspired load balancing of tasks in cloud computing environments. **Applied Soft Computing**, v. 13, n. 5, p. 2292 – 2303, 2013. Citado na página 37.
- LI, Y.; LI, W.; JIANG, C. A survey of virtual machine system: Current technology and future trends. In: IEEE. **Electronic Commerce and Security (ISECS), 2010 Third International Symposium on**. [S.l.], 2010. p. 332–336. Citado na página 9.
- MASDARI, M.; NABAVI, S. S.; AHMADI, V. An overview of virtual machine placement schemes in cloud computing. **Journal of Network and Computer Applications**, v. 66, p. 106–127, 2016. Citado na página 37.
- MELL, P.; GRANCE, T. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011. Citado nas páginas 5, 6 e 7.
- MENASCÉ, D. A. Virtualization: Concepts, applications, and performance modeling. In: **Int. CMG Conference**. [S.l.: s.n.], 2005. p. 407–414. Citado na página 9.
- MENKEN, I.; BLOKDIJK, G. **Cloud Computing Foundation Complete Certification Kit-Study Guide Book and Online Course**. [S.l.]: Emereo Pty Ltd, 2010. Citado na página 7.
- MITCHELL, M. **An introduction to genetic algorithms**. [S.l.]: MIT press, 1998. Citado na página 24.
- MONKS, E. **Planejamento de Capacidade em Redes Corporativas para Implementação de Serviços VoIP**. Tese (Doutorado) — Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Instituto de Informática. Universidade Federal do Rio Grande do Sul, 2006. Citado nas páginas 13 e 14.
- MOSCATO, P.; MENDES, A.; BERRETTA, R. Benchmarking a memetic algorithm for ordering microarray data. **Biosystems**, v. 88, n. 1–2, p. 56 – 75, 2007. Citado na página 53.
- NAKAMURA, L. H. V. **Mecanismos de autoconfiguração e auto-otimização para arquiteturas virtualizadas que visam a provisão de qualidade de serviço**. 217 p. Tese (Doutorado) — Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2017. Citado nas páginas 15, 40, 43, 50, 51 e 98.
- NARENDRA, P. M.; FUKUNAGA, K. A branch and bound algorithm for feature subset selection. **IEEE Transactions on Computers**, v. 9, n. C-26, p. 917–922, 1977. Citado na página 20.
- PARETO, V. **Cours d'économie politique**. [S.l.]: Librairie Droz, 1964. v. 1. Citado na página 29.

- RAFIQ, M.; MATHEWS, J.; BULLOCK, G. Conceptual building design-evolutionary approach. **Journal of computing in civil engineering**, American Society of Civil Engineers, v. 17, n. 3, p. 150–158, 2003. Citado na página 23.
- RIMAL, B. P.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. In: IEEE. INC, IMS and IDC, 2009. NCM'09. Fifth International Joint Conference on. [S.l.], 2009. p. 44–51. Citado nas páginas 7 e 9.
- RITTINGHOUSE, J. W.; RANSOME, J. F. **Cloud computing: implementation, management, and security**. [S.l.]: CRC press, 2009. Citado na página 12.
- _____. **Cloud computing: implementation, management, and security**. [S.l.]: CRC press, 2016. Citado na página 2.
- ROSE, R. Survey of system virtualization techniques. 2004. Citado na página 9.
- SAIT, S. M.; YOUSSEF, H. **Iterative computer algorithms with applications in engineering: solving combinatorial optimization problems**. [S.l.]: IEEE Computer Society Press, 1999. Citado na página 23.
- SONI, G.; KALRA, M. A novel approach for load balancing in cloud data center. In: IEEE. **Advance Computing Conference (IACC), 2014 IEEE International**. [S.l.], 2014. p. 807–812. Citado nas páginas 34 e 37.
- STANTCHEV, V.; SCHRÖPFER, C. Negotiating and enforcing qos and slas in grid and cloud computing. In: **Advances in grid and pervasive computing**. [S.l.]: Springer, 2009. p. 25–35. Citado na página 9.
- SUITE, P. T. **Smallpt Benchmarking**. 2017. Available from: <<https://openbenchmarking.org/test/pts/smallpt>>. Accessed: 06-27-2017. Citado nas páginas 47 e 67.
- TALBI, E.-G. **Metaheuristics: from design to implementation**. [S.l.]: John Wiley & Sons, 2009. v. 74. Citado nas páginas 15, 19, 20 e 21.
- TEUSCHER, C.; MANGE, D.; STAUFFER, A.; TEMPESTI, G. Bio-inspired computing tissues: towards machines that evolve, grow, and learn. **Biosystems**, Elsevier, v. 68, n. 2, p. 235–244, 2003. Citado na página 22.
- TOLEDO, C. F. M.; ARANTES, M. da S.; OLIVEIRA, R. R. D.; ALMADA-LOBO, B. Glass container production scheduling through hybrid multi-population based evolutionary algorithm. **Applied Soft Computing**, v. 13, n. 3, p. 1352–1364, 2013. Citado nas páginas 53 e 68.
- TOSCANI, L. V.; VELOSO, P. A. Complexidade de algoritmos. **ARTMED Editora SA, Porto Alegre-RS**, 2008. Citado na página 22.
- TRUONG, V.-H.; NGUYEN, P.-C.; KIM, S.-E. An efficient method for optimizing space steel frames with semi-rigid joints using practical advanced analysis and the micro-genetic algorithm. **Journal of Constructional steel research**, Elsevier, v. 128, p. 416–427, 2017. Citado na página 56.
- TSAI, C.-W.; WANG, S.-W.; HU, K.-C.; CHIANG, M.-C. A modified multiple-search multi-start framework-enhanced micro genetic algorithm for wlan load balancing. In: IEEE. **Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on**. [S.l.], 2015. p. 202–207. Citado na página 56.

- VAZIRANI, V. V. **Approximation algorithms**. [S.l.]: Springer Science & Business Media, 2013. Citado na página 20.
- VEGESNA, S. **IP quality of service**. [S.l.]: Cisco Press, 2001. Citado na página 9.
- VELEV, D.; ZLATEVA, P. Cloud infrastructure security. In: **Open Research Problems in Network Security**. [S.l.]: Springer, 2011. p. 140–148. Citado nas páginas 6 e 7.
- WANG, X.; DU, Z.; CHEN, Y. An adaptive model-free resource and power management approach for multi-tier cloud environments. **Journal of Systems and Software**, v. 85, n. 5, p. 1135–1146, 2012. Citado na página 36.
- WEBER, T. S. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. **Relatório técnico, Instituto de Informática UFRGS**, 2007. Citado na página 10.
- WOEGINGER, G. J. Exact algorithms for np-hard problems: A survey. In: **Combinatorial Optimization—Eureka, You Shrink!** [S.l.]: Springer, 2003. p. 185–207. Citado na página 41.
- WU, L.; GARG, S. K.; BUYYA, R. Sla-based admission control for a software-as-a-service provider in cloud computing environments. **Journal of Computer and System Sciences**, Elsevier, v. 78, n. 5, p. 1280–1299, 2012. Citado nas páginas 35 e 37.
- YAN, Q.; YU, F. R.; GONG, Q.; LI, J. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. **IEEE Communications Surveys & Tutorials**, IEEE, v. 18, n. 1, p. 602–622, 2016. Citado na página 2.
- YAZIR, Y. O.; AKBULUT, Y.; FARAHBOD, R.; GUITOUNI, A.; NEVILLE, S. W.; GANTI, S.; COADY, Y. Autonomous resource consolidation management in clouds using impromptu extensions. In: **Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on**. [S.l.: s.n.], 2012. p. 614–621. Citado na página 37.
- YAZIR, Y. O.; MATTHEWS, C.; FARAHBOD, R.; NEVILLE, S.; GUITOUNI, A.; GANTI, S.; COADY, Y. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In: **Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on**. [S.l.: s.n.], 2010. p. 91–98. Citado na página 37.
- YUCHAO, Z.; BO, D.; FUYANG, P. An adaptive qos-aware cloud. In: **Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on**. [S.l.: s.n.], 2012. p. 160–163. Citado na página 36.
- ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. **Journal of internet services and applications**, Springer, v. 1, n. 1, p. 7–18, 2010. Citado na página 1.
- ZHAO, J.; ZENG, W.; LIU, M.; LI, G. Multi-objective optimization model of virtual resources scheduling under cloud computing and it's solution. In: **IEEE. Cloud and Service Computing (CSC), 2011 International Conference on**. [S.l.], 2011. p. 185–190. Citado nas páginas 34 e 37.

GLOSSÁRIO

Framework: é uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica. *Frameworks* são projetados com a intenção de facilitar o desenvolvimento de *software*, habilitando designers e programadores a gastarem mais tempo determinando as exigências do *software* do que com detalhes de baixo nível do sistema.

Template: é um documento sem conteúdo, com apenas a apresentação visual (apenas cabeçalhos por exemplo) e instruções sobre onde e qual tipo de conteúdo deve entrar a cada parcela da apresentação.

Web: Sinônimo mais conhecido de *World Wide Web* (WWW). É a interface gráfica da Internet que torna os serviços disponíveis totalmente transparentes para o usuário e ainda possibilita a manipulação multimídia da informação.