UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIA E TECNOLOGIA

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

JULIANA OLIVEIRA DE CARVALHO

SISTEMA DE APOIO A DECISÃO PARA IMPLANTAR UMA APLICAÇÃO BASEADA EM MICROSSERVIÇOS EM UM AMBIENTE MULTI-CLOUD

FORTALEZA

2019

JULIANA OLIVEIRA DE CARVALHO

SISTEMA DE APOIO A DECISÃO PARA IMPLANTAR UMA APLICAÇÃO BASEADA EM MICROSSERVIÇOS EM UM AMBIENTE MULTI-CLOUD

Tese apresentada ao Programa de Pós-graduação em Ciências da Computação do Centro de Ciência e Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciências da Computação. Área de Concentração: Engenharia de Software

Orientador: Prof. Dr. Fernando Trinta.
Coorientador: Prof. Dr. Dario Vieira.

FORTALEZA

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C324s Carvalho, Juliana Oliveira de.
Sistema de apoio a decisão para implantar uma aplicação baseada em microsserviços em um ambiente multi-cloud / Juliana Oliveira de Carvalho. – 2019.
199 f. : il. color.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação , Fortaleza, 2019.
Orientação: Prof. Dr. Fernando Trinta.
Coorientação: Prof. Dr. Dario Vieira.

1. Seleção de múltiplos provedores de nuvem. 2. multi-cloud. 3. multi-requisitos. I. Título.
CDD 005

JULIANA OLIVEIRA DE CARVALHO

SISTEMA DE APOIO A DECISÃO PARA IMPLANTAR UMA APLICAÇÃO BASEADA EM MICROSSERVIÇOS EM UM AMBIENTE MULTI-CLOUD

Tese apresentada ao Programa de Pós-graduação em Ciências da Computação do Centro de Ciência e Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do título de doutor em Ciências da Computação. Área de Concentração: Engenharia de Software

Aprovada em: 30 de julho de 2019

BANCA EXAMINADORA

Prof. Dr. Fernando Trinta (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Dario Vieira (Coorientador)
EFREI-Paris

Prof. Dr. Vinícius Cardoso Garcia
Universidade Federal de Pernambuco (UFPE)

Prof. Dr. Nelson Souto Rosa
Universidade Federal de Pernambuco (UFPE)

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará (UFC)

Aos meus filhos, Ana Paula e Arthur e ao esposo Rayner.

AGRADECIMENTOS

Durante os cinco e meio anos, além dos desafios inerentes a um doutorado, eu passei junto com a minha família por vários desafios, dentre eles: mudança de cidade, mudança de escola, mudança de país, mudança de estilo de vida. Vários desafios foram superados e outros não, mas quero levar comigo desta experiência o aprendizado e a gratidão. Primeiro agradeço à Deus por tudo permitir e orientar. Depois, eu quero agradecer aqui as pessoas que foram fundamentais durante estes anos de doutorado.

Ao Prof. Dr. Fernando Trinta por me orientar em minha tese de doutorado, pela amizade, pela paciência, por me incentivar a enfrentar os desafios e por sempre ouvir minhas lamúrias.

Ao Prof. Dr. Dario Vieira por me co-orientar em minha tese de doutorado, pela amizade, pela paciência, pelo apoio diante dos problemas, pelo acolhimento e principalmente por acreditar no meu projeto.

Agradeço também a todos os professores do MDCC por me proporcionarem a oportunidade de compartilhar do vossos conhecimentos.

Aos meus colegas, professores do curso de Sistemas de Informação do campus de Picos da Universidade Federal da Piauí (UFPI), pelo apoio as minhas necessidades de afastamento.

As amizades feitas durante o doutorado que continuarão pela vida, Rainara, Jaminny, Christiano e Adyson. Cada um de vocês teve uma grande importância em momentos distintos do doutorado. Obrigada por tudo. E através da Rainara gostaria de agradecer aos demais colegas de doutorado que de forma direta ou indireta colaboraram para chegar ao final desta trajetória.

À família do professor Dario Vieira pelo apoio e acolhimento. Aos amigos Luiz Reis e Letícia Carvalho pelo incentivo e apoio emocional. Aos MegaAmigos, a nossa família no Piauí: Maria Carolina, Marta Rochelly, Medyna, Renata, Vieira, Ismael, Drumond, Márcio, obrigada por sempre estarem juntos nos dias bons e ruins.

Aos meus familiares, irmãs (Flávia e Cárthea), cunhada(Franciane), cunhados(Carlos, Klelcion e Cleber), sobrinhos(Sofia, Gabriel e Otávio), sogra (Nilta) e sogro (Walter) por todo apoio. A minha Tia Jurany e toda sua família, que sempre me apoiaram em tudo, meu muito obrigada. Além destes, as minhas tia, tios e primos que sempre torceram e oraram por mim. E a minha querida mãe, Wilma, que sempre está pronta para nos ajudar no que for preciso, e sempre fazendo tudo que pode para que possamos ser felizes, muito obrigada por tudo!

E por último, eu gostaria de agradece aos meus filhos, Ana Paula e Arthur, que passaram por amadurecimentos diante do contexto de vida que nos submetemos durante os anos de doutorado e ao meu esposo, Rayner, pelo companheirismo, por estarmos sempre juntos desde os momentos mais duros e os mais leves. Eu gostaria de me desculpar às vezes que eu não soube entendê-los, bem como por estar ausente, mesmo estando presente fisicamente e de agradecer imensamente por toda compreensão.

Além das pessoas aqui citadas, eu gostaria de agradecer as instituições que colaboraram para alcançar este objetivo.

Ao Programa de Mestrado e Doutorado em Ciência da Computação MDCC da Universidade federal do Ceará (UFC), por toda infraestrutura ofertada.

À EFREI-Paris por me receber como pesquisadora e pelo financiamento da pesquisa de doutorado via bolsa de estudos.

À Fundação Cearense de Apoio ao Desenvolvimento (Funcap) pelo financiamento da pesquisa de doutorado via bolsa de estudos.

À Universidade Federal do Piauí (UFPI) por permitir o afastamento das atividades acadêmicas para me dedicar ao doutorado.

"Lute com garra e determinação por seus objetivos. Tenha consciência de seus obstáculos. Não fuja e não se apavore diante deles. Enfrente os desafios e combata com coragem seus medos. Tenha, cultive, conserve e propague sempre pensamentos otimistas e positivos."

(Mábio S. Moreira Viana)

RESUMO

A computação em nuvem tornou-se um modelo popular de entrega de serviços, trazendo vários benefícios. Todavia, para aplicar o modelo de nuvem em determinados cenários alguns desafios devem se superados. Um desses problemas é implantar e executar aplicações em diferentes provedores, pois estes são vários, e oferecem diversos serviços com as mesmas funcionalidades e diferentes recursos. Assim, lidar com problemas de distribuições de aplicações em diferentes provedores é uma tarefa complexa para um arquiteto de *software*, pois os componentes de uma aplicação possuem diferentes características. Soluções foram propostas para lidar com o problema, mas a maioria delas foca em atender aos provedores. Portanto, esta pesquisa propõe um sistema de tomada de decisão baseado em economicidade para implantar uma aplicação distribuída em múltiplos provedores de nuvem. Este trabalho considera somente aplicações baseadas em microsserviços, pois estes oferecem maior flexibilidade. Assim, a solução proposta deve selecionar provedores que melhor atendam aos requisitos dos microsserviços e de um arquiteto de *software*, de forma que os microsserviços possam ser implantados em provedores distintos. A fim de que o sistema possa atender a uma diversidade de cenários, propõe-se três modelos de seleção. Para alcançar os objetivos, propõe-se também uma definição, uma classificação e taxonomias para o gerenciamento de recursos em múltiplos provedores da perspectiva de um arquiteto de *software* e uma definição de microsserviços no contexto de múltiplas nuvens. Além disso, uma arquitetura para o gerenciamento da implantação e execução de aplicações baseadas em microsserviços distribuídas em *multi-cloud* é proposta e nomeada de PacificClouds. Ao final, uma análise comparativa é realizada entre os três modelos propostos, a qual mostra a viabilidade em relação à performance das soluções propostas para cada um dos modelos.

Palavras-chave: Seleção de múltiplos provedores. Múltiplos requisitos. Microsserviços.

ABSTRACT

Cloud computing has become a trendy model of service delivery, bringing various benefits. However, to apply the cloud model in specific scenarios, some challenges must be overcome. One of these problems is to deploy and run applications in various providers, and each one comprises several services with similar functionalities and different capabilities. Thus, dealing with issues of application distributions in multiple providers is a complex task for a software architect, since the components of an application have different characteristics. Solutions have been proposed to face this problem, but most of them focus on service providers. Therefore, we propose a cost-effective decision-making system to deploy a distributed application across multiple cloud providers. We consider in this work applications based on microservices, for offering greater flexibility. Thus, the proposed solution select providers that best meet the microservices and software architect requisites, in a manner that the microservices can be deployed in many providers. We propose three selection models for the system to serve a variety of scenarios. To reach the objectives, we also offer a definition, a classification and taxonomies for the management of resources in multiple providers from the perspective of a software architect, and a definition of microservices in the context of multi-cloud. Further, we propose PacificClouds, an architecture for managing the deployment and execution of applications based on microservices distributed multi-cloud. In the end, we accomplished a comparative analysis of the three proposed models, which one shows the feasibility concerning the performance of the solutions applied in each of the models.

Keywords: Multiple Clouds Selection. Multi-requirements. Microservices.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
CDO	*cloud deployment option*
CMCR	Classificação de Recursos em Múltiplas Nuvens
GA	*Genetic Algorithm*
IaaS	*Infrastracture as a Service*
IDC	*International Data Corporation*
MCKP	*Multi-Choice Knapsack Problem*
MCR	*multiple clouds resources*
MCRM	*multiple clouds resources management*
PaaS	*Platform as a Service*
PSG	*Provider and Service Granularity*
QoS	*Quality of Service*
SaaS	*Service as a Service*
SAW	*Simple Additive Weighting*
SLA	*Service Level Agreement*
TIC	Tecnologia de Informação e Comunicação
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO

A computação em nuvem é um modelo de prestação de serviços que foi empregado com sucesso nos últimos anos. Segundo Vaquero *et al.* (2009), a computação em nuvem é vista como um grande conjunto de recursos virtualizados, facilmente acessíveis e utilizáveis (como *hardware*, plataformas de desenvolvimento e/ou serviços). Esses recursos podem ser dinamicamente reconfigurados para se ajustar a uma carga variável (escala), permitindo também uma utilização otimizada dos recursos. Esse conjunto de recursos é tipicamente explorado através de um modelo de pagamento por uso, no qual as garantias são oferecidas pelo provedor de infraestrutura através de acordos de nível de serviço (*Service Level Agreement*s (SLAs)) personalizados. De certa forma, esse modelo materializa a visão da computação utilitária defendida por pesquisadores como John McCarthy e Douglas Parkhill, que na década de 1960 previram que futuros recursos computacionais seriam comprados de forma semelhante aos serviços públicos comuns, como água, gás ou eletricidade. De fato, hoje é possível facilmente "adquirir" vários recursos de computação, desde servidores a aplicações completas, de acordo com as necessidades do usuário.

Empresas de várias áreas adotaram a computação em nuvem em razão das vantagens por ela ofertadas. Uma das vantagens é que os recursos em nuvem permitem às empresas otimizarem seus gastos em ativos de Tecnologia de Informação e Comunicação (TIC), proporcionando maior agilidade ao gerenciamento desses recursos e, consequentemente, melhorando sua competitividade. Além dessa vantagem, por meio da computação em nuvem, pequenas e novas empresas podem reduzir a barreira de entrada no mercado, permitindo que os custos operacionais sejam proporcionais ao crescimento da receita.

As vantagens oferecidas pela computação em nuvem são confirmadas através dos investimentos mundiais em serviços e infraestrutura de nuvem pública, pois a *International Data Corporation* (IDC) prevê um aumento anual em investimentos de 22,5% entre 2017-2022 (SMITH; SHIRER, 2019). Além disso, Smith e Shirer (2019) preveêm que mais da metade de todos os investimentos em nuvem pública em 2019 é em *Service as a Service* (SaaS).

As aplicações criadas com base nos conceitos de nuvem obtém vantagens em relação ao crescimento elástico, facilidade de atualização e manutenção e melhor uso de recursos, tal como o uso de abordagens *multitenancy*. No entanto, do ponto de vista da engenharia de *software*, ainda há muitos desafios para a criação de aplicações centradas na nuvem que precisam ser abordados. Alguns desses obstáculos incluem questões como privacidade de dados, migração

de aplicações para a nuvem e bloqueio de aplicações para um provedor específico, problema comumente mencionado como *vendor lock-in* (OPARA-MARTINS *et al.*, 2015), (MEZGÁR; RAUSCHECKER, 2014), (SILVA *et al.*, 2013) e (PETCU, 2013a).

Nos últimos anos, vários trabalhos propuseram soluções para esses e outros desafios relacionados ao desenvolvimento de aplicações nativas para nuvem. Um dos cenários estudados tem cada componente de um sistema distribuído em mais de um provedor de nuvem, e a distribuição desses componentes leva em consideração requisitos como o desempenho do serviço de um provedor, a disponibilidade e o custo operacional (SOUSA *et al.*, 2016), (WANG *et al.*, 2017) e (JATOTH *et al.*, 2019). Esse cenário é conhecido na literatura como ambiente de múltiplos provedores de nuvem e oferece aos arquitetos de *software* uma infinidade de possibilidades para configurar suas aplicações. Assim, um ambiente de múltiplos provedores traz como um dos principais benefício o uso de serviços que melhor atendam as necessidades de uma aplicação, independentemente, de qual provedor oferta o serviço (PETCU, 2013b) e (PETCU, 2014b).

O ambiente de múltiplos provedores de nuvem engloba diferentes visões sobre como as aplicações os utilizam. Desta maneira, taxonomias e classificações foram propostas por diferentes trabalhos, a exemplo de Petcu (2014a), Toosi *et al.* (2014) e Grozev e Buyya (2014). De acordo com Petcu (2014c) existem três modelos de entrega de serviços de nuvem: *multi-cloud*, *cloud federation* e *inter-cloud*. A principal diferença entre eles é que o primeiro modelo não possui acordo entre os provedores envolvidos, diferentemente do segundo, o qual possui acordo entre os provedores, já no terceiro, por sua vez, pode ou não haver um acordo entre os provedores, mas obrigatoriamente deve existir um *broker* para intermediar os provedores de nuvem. Assim, esta pesquisa adota o modelo de entrega de serviço *multi-cloud*, por não depender de acordo entre provedores e, consequentemente, por oferecer maior oferta de provedores e de serviços de nuvem.

1.1 Motivação

Diante de um contexto em que há um grande crescimento do número de provedores de nuvem, bem como do número de recursos ofertados por eles, o uso de múltiplos provedores de nuvem pode trazer muitos benefícios econômicos, como descritos no preâmbulo deste capítulo. Além disso, o desenvolvimento de aplicações para serem implantadas em múltiplos provedores de nuvem é considerado um problema recente. Todavia, há muitos desafios a serem enfrentados,

sendo alguns deles: segurança, privacidade, confiança, questões legais, gerenciamento de recursos e serviços de SLA (TOOSI *et al.*, 2014). Esta tese de doutorado procura enfrentar os desafios da seleção de provedores de nuvem para hospedar uma aplicação distribuída em um ambiente *multi-cloud*, com vistas a facilitar as tomadas de decisões de um arquiteto de *software*, de forma a oferecer um sistema de tomada de decisão baseado em economicidade em termos de tempo e custo.

Um aspecto importante para alcançar esses objetivos é entender como as aplicações nativas para nuvem são projetadas recentemente. Ultimamente, um estilo arquitetônico que se destacou no *design* de aplicações para nuvem é baseado em microsserviços. De acordo com Sethi (2017), microsserviços são a decomposição de um sistema de negócios monolítico em serviços implantáveis independentes. Assim, eles nos permitem construir aplicações cujo código é organizado em várias partes distintas, que são executadas em processos separados, conforme descrito por Ziade (2017). A adequação desse estilo arquitetural para aplicações em nuvem o torna um candidato natural também para aplicações distribuídas em um ambiente *multi-cloud*. Desta forma, esta pesquisa foca em aplicações baseadas em microsserviços.

Além disso, a implantação de aplicações em nuvem, quer sejam estas únicas ou múltiplas, envolve diferentes atores os quais variam desde o usuário final, passando pelo arquiteto de *software* que desenvolve as aplicações, e chegando até os operadores em nuvem responsáveis por dar suporte a execução dessas aplicações. Um arquiteto de *software* exerce várias funções, dentre elas, projetar uma solução que atenda aos requisitos do cliente/empresa, e que tenha flexibilidade para suportar mudanças ou novos requisitos ao longo do tempo. Assim, no contexto desta pesquisa, o usuário-alvo é o arquiteto de *software* que precisa tomar decisões sobre a melhor forma de selecionar provedores de nuvem para hospedar aplicações baseadas em microsserviços em um ambiente *multi-cloud*.

1.2 Questões de Pesquisa

De acordo com o mencionado na seção anterior, a questão de pesquisa primária (PRQ) a ser investigada neste trabalho de doutorado é:

Como selecionar provedores de nuvem para hospedar os microsserviços de uma aplicação em um ambiente *multi-cloud*, de acordo com a perspectiva de um arquiteto de *software*?

Para auxiliar na investigação de propostas de soluções para a questão de pesquisa

central, propõe-se três perguntas de pesquisa secundária (SRQ), as quais orientam o presente estudo:

- **SRQ1: Quais são as demandas de um arquiteto de *software* em relação ao gerenciamento de aplicações em um ambiente *multi-cloud*?** Esta questão visa descobrir as necessidades de gerenciamento dos arquitetos de *software* para uma aplicação distribuída em múltiplos provedores de nuvem, a fim de ajudar a obter informações necessárias para a seleção de provedores para hospedar microsserviços;
- **SRQ2: Quais recursos podem ser gerenciados em múltiplos provedores de nuvem? Quando e como eles devem ser gerenciados?** Para tomar decisões, os arquitetos de *software* precisam entender o quê são e quais são os recursos que podem ser gerenciados em um ambiente de múltiplos provedores de nuvem, como: processadores, banco de dados, *softwares*, rede, capacidades de nuvem e localização de *data centers*. Assim, essa questão tem como objetivo investigar quais recursos podem ser gerenciados em um ambiente de múltiplos provedores de nuvem e ajudar a descobrir como e quando gerenciá-los;
- **SRQ3: Como uma aplicação deve ser desenvolvida para ser implantada em diferentes provedores de nuvem?** Para facilitar o processo de distribuição de uma aplicação em um ambiente de múltiplos provedores e, consequentemente, do processo de seleção de provedores, o desenvolvimento de uma aplicação deve ser realizado com foco em computação em nuvem. Portanto, essa questão contribui na compreensão das características de uma aplicação desenvolvida para ser implantada em computação em nuvem, bem como o estilo arquitetural mais usado recentemente para atender a essas características denominados de microsserviços.

1.3 Objetivos da Pesquisa

Este trabalho tem como objetivo principal propor soluções para a seleção de múltiplos provedores de nuvem para hospedar uma aplicação baseada em microsserviços da perspectiva de um arquiteto de *software*, com enfoque em economicidade. A fim de alcançar esse objetivo maior, foram definidos quatro objetivos mais específicos, a saber:

- **Obj01: Propor as demandas de um arquiteto de *software* para o gerenciamento de recursos em múltiplos provedores de nuvem.** (SRQ1) A fim de propor um processo de seleção que possa atender às necessidades de um arquiteto de *software*.
- **Obj02: Propor uma definição e uma classificação para recursos em múltiplos prove-**

dores de nuvem (MCR) e taxonomias de quando e como gerenciar tais recursos da perspectiva de um arquiteto de *software*. (SRQ2) O intuito é compreender o quê são e quais são os recursos em um ambiente de múltiplos provedores, além de identificar quando e como eles podem ser gerenciados para melhor compreender o processo de seleção de múltiplos provedores para hospedar uma aplicação distribuída.

- **Obj03: Propor uma arquitetura de referência para suporte à implantação, monitoramento e atualização de aplicações em ambientes *multi-cloud*.** (SRQ1, SRQ2 e SRQ3) Esta arquitetura, batizada no trabalho como PacificClouds (CARVALHO *et al.*, 2018b), tem como objetivo gerenciar todo o processo de implantação e execução de uma aplicação baseada em microsserviços distribuída em um ambiente *multi-cloud* da perspectiva de um arquiteto de *software*. Desta forma, o processo de seleção de múltiplos provedores pode ser identificado de acordo como as necessidades da arquitetura e, consequentemente, de uma aplicação e de um arquiteto de *software*.
- **Obj04: Propor modelos, soluções e avaliações para a seleção de múltiplos provedores de nuvem que possam ser adotados pelo PacificClouds.** (SRQ3) Com vistas a promover uma maior flexibilidade em relação ao número de provedores disponíveis, ao tamanho de uma aplicação e a grau de comunicação entre os microsserviços, esta pesquisa propõe três modelos para a seleção de múltiplos provedores de nuvem, bem como soluções para cada um deles. Com estes modelos, o PacificClouds pode escolher o modelo de seleção mais adequado à aplicação e às necessidades de um arquiteto de *software*.

1.4 Metodologia de Pesquisa

A realização deste trabalho de doutorado segue as metas descritas na Seção 1.3. Assim, a metodologia de pesquisa está dividida em quatro fases, e cada uma delas incluem um conjunto de tarefas como apresentado na Figura 1. Na primeira fase, descreve-se o processo de revisão da literatura e a definição dos objetivos deste trabalho. Na segunda fase, apresenta-se as propostas e, na terceira fase, propõe-se as soluções a fim de alcançar os objetivos aqui delimitado. Por fim, apresenta-se o processo de avaliação das soluções propostas para cada um dos modelos.

Na primeira fase da Figura 1, a revisão da literatura apresenta as tarefas relacionadas à investigação dos desafios do uso de múltiplos provedores para melhor usufruir dos benefícios da computação em nuvem e, ao mesmo tempo, para ajudar na definição dos objetivos desta tese. Além disso, uma revisão de literatura secundária foi realizada para investigar temas relacionados

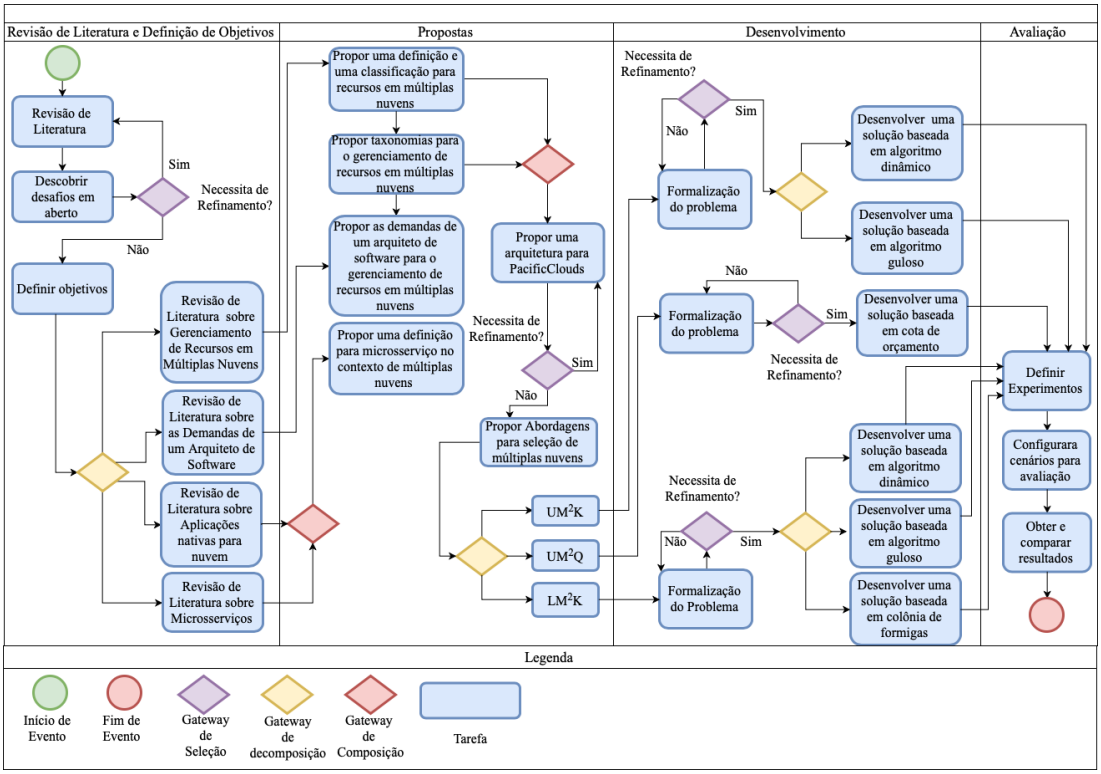


Figura 1 – Metodologia de Pesquisa para o projeto de Doutorado.

ao objetivo principal desta pesquisa, ou seja, que ajudam a propor soluções para o processo de seleção de múltiplos provedores para hospedar os microsserviços de uma aplicação distribuída em um ambiente *multi-cloud*. Essas revisões de literatura utilizaram bases de dados *online*, como IEEE Xplore[1], ACM DL[2], Science Direct[3] e Google Scholar[4]. A fim de delimitar este trabalho, vários refinamentos foram realizados nos resultados obtidos, como: exclusão pelo título, pelo resumo e pela conclusão dos trabalhos.

Na segunda fase, a metodologia de pesquisa teve por objetivo descrever um *background* para a seleção de múltiplos provedores de nuvem. Primeiro, foi proposta uma definição e uma classificação para MCR (CARVALHO *et al.*, 2018a). Além disso, foram propostas três taxonomias para o gerenciamento de recursos em múltiplos provedores de nuvem (*multiple clouds resources management* (MCRM)), que focam em como e quando gerenciar o MCR e em como configurar os requisitos de um arquiteto de *software* para uma aplicação distribuída em múltiplos provedores. Também foram propostas as demandas de um arquiteto de *software* para o MCRM, as quais descrevem suas necessidades para o gerenciamento de uma aplicação

[1] https://ieeexplore.ieee.org/
[2] https://dl.acm.org/
[3] https://www.sciencedirect.com/
[4] https://scholar.google.com/intl/en-US/scholar/about.html

hospedada em múltiplos provedores. Como complemento, uma definição para microsserviços no contexto de *multi-cloud* (CARVALHO *et al.*, 2018b) foi formulada. De acordo com as definições, classificação e taxonomias, a arquitetura para o PacificClouds foi projetada com objetivo de gerenciar o processo de implantação e execução de uma aplicação baseada em microsserviços em um ambiente *multi-cloud*. Para que seja possível alcançar os objetivos do PacificClouds, é fundamental a seleção de provedores de nuvem para hospedar uma aplicação distribuída baseada em microsserviços. Assim, três modelos para o processo de seleção de múltiplos provedores para hospedar os microsserviços de uma aplicação são propostos. O primeiro é chamado de microsserviços desvinculados mapeados para o problema da mochila (*Unlinked Microservice Mapped to Knapsack - UM^2K*); o segundo é nomeado de microsserviços desvinculados mapeados para cotas de orçamento (*Unlinked Microservice Mapped to Quota - UM^2Q*), e o último é chamado de microsserviços vinculados mapeados para o problema da mochila (*Linked Microservice Mapped to Knapsack - LM^2K*). Os modelos propostos neste trabalho colocam um microsserviço em um único provedor. Desta forma, para cada microsserviço é selecionado um conjunto de serviços em um provedor com vistas a atender às necessidades do microsserviço. O principal objetivo é selecionar o provedor que oferece o melhor conjunto de serviços de acordo com os requisitos do microsserviço e com os requisitos de um arquiteto de *software* (usuário do PacificClouds).

Na terceira fase, denominada Desenvolvimento, são apresentadas as soluções relacionadas aos modelos propostos para a seleção de múltiplos provedores de nuvem. Primeiro são descritas matematicamente cada modelo proposto. Depois, as soluções propostas para cada um dos modelos são implementadas. Para o modelo, UM^2K, duas soluções são propostas: uma baseada em um algoritmo dinâmico e a outra em um algoritmo guloso. Uma solução para o modelo UM^2Q também é proposta, a qual é baseada em requisitos individuais, onde cada microsserviço recebe uma cota do orçamento total disponível para uma aplicação. Para o terceiro modelo, LM^2K, três soluções são propostas: a primeira baseada em um algoritmo dinâmico; a segunda em um algoritmo guloso e a terceira, em um algoritmo bioinspirado em colônia de formigas.

A última fase, nomeada de Avaliação, mostra as atividades referentes a avaliação das soluções propostas na terceira fase. Esta pesquisa concentra-se na avaliação de desempenho, cujos os experimentos são definidos baseados no número de provedores, no número de microsserviços de uma aplicação e na variação dos requisitos de um arquiteto de *software* para uma aplicação. Depois, cenários são configurados, nos quais vários requisitos de provedores e de aplicações

são definidos. Os conjuntos de provedores diferem um do outro pelo número de provedores ou pela quantidade de serviço por provedor. Os requisitos de uma aplicação variam também de acordo com o número de microsserviços. Por fim, uma comparação dessas soluções é realizada de acordo com os resultados obtidos nos experimentos.

1.5 Contribuições da Tese

A principal contribuição desta tese é um processo de seleção de múltiplos provedores para hospedar os microsserviços de uma aplicação em um ambiente *multi-cloud* da perspectiva de um arquiteto de *software* (CARVALHO *et al.*, 2018) e (CARVALHO *et al.*, 2019).

Além desta contribuição esta tese oferece outras seis contribuições para auxiliar no processo de adoção de um ambiente *multi-cloud*:

- Uma arquitetura de referência para o gerenciamento do processo de implantação e da execução de uma aplicação baseada em microsserviços distribuída em um ambiente *multi-cloud*, nomeado de PacificClouds (CARVALHO *et al.*, 2018b).
- Uma definição para microsserviços no contexto de um ambiente *multi-cloud* (CARVALHO *et al.*, 2018b).
- Uma definição e uma classificação para MCR (CARVALHO *et al.*, 2018a).
- Taxonomias referentes a quando e como gerenciar MCR e como configurar os requisitos de um arquiteto de *software* (CARVALHO *et al.*, 2018a).
- Uma taxonomia para identificar o foco de uma pesquisa na seleção de provedores de nuvem em relação ao número de provedores e serviços usados por eles antes e depois da implantação de uma aplicação, que denomina-se granularidade de provedores e serviços (*Provider and Service Granularity* (PSG)) (CARVALHO *et al.*, 2019).
- Uma avaliação de desempenho para as soluções propostas, bem como uma análise comparativa entre os resultados obtidos.

1.6 Restrições do Trabalho

Esta tese possui algumas contribuições para a adoção de um ambiente *multi-cloud*, mas está fora do seu escopo:

- Aplicações tradicionais que usam arquitetura monolítica e as que não foram projetas para nuvem.

- Ambiente com um único provedor de nuvem e ambientes de múltiplos provedores de nuvem cujo os modelos de entrega sejam: *cloud federation* ou *inter-cloud*.
- Como um arquiteto de *software* descobre das capacidades dos provedores de nuvem e informa os requisitos de uma aplicação.
- Seleção de provedores para uma aplicação em tempo de execução.

1.7 Organização do Documento

Este capítulo apresentou os problemas que motivam este trabalho de doutorado, os pontos de partida envolvidos, as questões de pesquisa e seus principais objetivos. Apresentou-se também a metodologia de pesquisa utilizada para alcançar os objetivos aqui delimitados. As demais partes constituídas deste documento estão organizadas da seguinte forma:

Capítulo 2 (Gerenciamento de Recursos em Múltiplas Nuvens): tem como objetivo fornecer informações para o processo de seleção de múltiplos provedores da perspectiva de um arquiteto de *software*. Para isto, apresenta as demandas de um arquiteto de *software* para o gerenciamento de MCR para uma aplicação distribuída. Além disso, propõe uma definição e uma classificação para recursos em múltiplos provedores de nuvem e três taxonomias para o MCRM. Por fim, apresenta o estado da arte em relação ao MCRM da perspectiva de um arquiteto de *software*.

Capítulo 3 (Aplicações Distribuídas em Múltiplas Nuvens): tem como objetivo mostrar a necessidade do desenvolvimento com foco na computação em nuvem para uma aplicação que deve ser implantada em um ambiente de computação em nuvem. Para isto, apresenta conceitos relacionados a aplicações nativas para nuvem e de microsserviços. Além disso, propõe uma definição para microsserviços no contexto de *multi-cloud*. Em seguida, apresenta e descreve detalhadamente a arquitetura do PacificClouds. Depois, apresenta um cenário que mostra a importância do PacificClouds. Por fim, apresenta e compara trabalhos relacionados ao PacificClouds.

Capítulo 4 (Modelos para a Seleção de Múltiplas Nuvens): apresenta e descreve três modelos para selecionar os múltiplos provedores de nuvem para hospedar os microsserviços de uma aplicação.

Capítulo 5 (Soluções para Seleção de Múltiplas Nuvens): apresenta as soluções projetadas para cada um dos modelos propostos no Capítulo 4. Para cada uma das soluções são apresentados um exemplo, um diagrama e um algoritmo.

Capítulo 6 (Experimentos e Resultados): apresenta, descreve os experimentos e os cenários

para avaliação de desempenho de cada uma das soluções propostas no capítulo 5. Além disso, apresenta e compara os resultados obtidos.

Capítulo 7 (Trabalhos Relacionados): este capítulo descreve alguns trabalhos relacionados à seleção de múltiplos provedores e compara-os aos três modelos propostos.

Capítulo 8 (Considerações Finais e Trabalhos Futuros): este capítulo apresenta as considerações finais deste trabalho de doutorado, bem como alguns trabalhos futuros.

2 GERENCIAMENTO DE RECURSOS EM MÚLTIPLAS NUVENS

A proliferação da computação em nuvem está ocorrendo devido às suas características essenciais: agrupamento de recursos, amplo acesso à rede, serviço medido, autoatendimento sob demanda e rápida elasticidade (SOSINSKY, 2011). Consequentemente, os recursos são apresentados aos usuários como se advindos de uma fonte inesgotável. O pagamento é feito de acordo com o uso dos recursos, e usuários não precisam se preocupar com atualizações de infraestrutura e serviços, o que leva à uma redução no custo de operação para as empresas. Diante deste contexto, muitos provedores de nuvem surgiram, e para conquistar os usuários, eles buscaram se especificar para atender às necessidades dos usuários. Porém, isso fez com que eles se tornassem bastante específicos na forma como seus recursos são utilizados pelos seus clientes. Assim, os usuários que desejam plena utilização de todas as vantagens da computação em nuvem, que seja por meio do uso de serviços de vários provedores para compor uma aplicação mais complexa e inovadora, ou migrando de um provedor para outro a fim de obter melhores qualidade de serviços, precisam reconstruir suas aplicações, tornando o usuário refém de um provedor de nuvem. Essa situação é comumente conhecida como o problema de *vendor lock-in* (OPARA-MARTINS *et al.*, 2015).

Uma maneira de melhor usufruir das vantagens da computação em nuvem é usar múltiplas nuvens. De acordo com NIST (2011) e Petcu (2014c), uma aplicação em múltiplas nuvens utiliza recursos de nuvens distintas geograficamente, sequencialmente ou simultaneamente. A forma sequencial ocorre quando move-se uma aplicação ou serviço de uma nuvem para outra, enquanto o método simultâneo acontece quando diferentes serviços de nuvens são usados em paralelo. Petcu (2014c) classifica diferentes cenários de aplicações em múltiplas nuvens como **modelos de entrega** (***delivery models***), sendo eles, *multi-cloud*, *cloud federation* e *inter-cloud*. *Multi-cloud* é um modelo de entrega que não inclui um acordo prévio entre os provedores de nuvem envolvidos na execução de uma aplicação. Por outro lado, existe um acordo de colaboração estabelecido entre provedores de nuvem para compartilhar recursos na *cloud federation*. No último modelo, a *inter-cloud* pode ser tanto uma *cloud federation* quanto uma *multi-cloud*, mas os serviços escalonáveis e oportunistas devem ser dinâmicos. Um recurso importante do modelo de entrega *inter-cloud* é a presença de um *broker*, que é um ator intermediário no relacionamento entre o provedor e o usuário. Portanto, de acordo com esses três modelos, o uso de múltiplas nuvens pode ajudar a melhorar o *Quality of Service* (QoS) dos serviços de nuvem utilizados por uma aplicação, visto que um usuário pode usar múltiplas nuvens sequencialmente ou em

paralelo, ou até mesmo através de um *broker*.

De acordo com as definições dos modelos de entrega, pode-se observar que a interação entre as nuvens é obrigatória. Portanto, a portabilidade e a interoperabilidade são essenciais para a distribuição de uma aplicação em múltiplas nuvens (SILVA; LUCRÉDIO, 2012). No entanto, a portabilidade e a interoperabilidade são atualmente desafios em múltiplas nuvens. Para resolver esses problemas, a comunidade acadêmica e a indústria propuseram algumas soluções, dentre eles: mOSAIC (PETCU *et al.*, 2013a), MODAClouds (ARDAGNA *et al.*, 2012) e OPTMIS (FERRER *et al.*, 2012), embora nenhuma delas seja amplamente adotada devido à falta de flexibilidade.

O primeiro desafio está relacionado à função de gerenciamento de recursos em múltiplas nuvens, a qual é responsável por (*i*) selecionar recursos em múltiplas nuvens para implantação de aplicações, por (*ii*) monitorar os recursos de uma aplicação distribuída em múltiplas nuvens em tempo de execução de acordo com os requisitos de uma aplicação e de um arquiteto de *software*, bem como por (*iii*) monitorar os recursos dos provedores de nuvem. O gerenciamento de recursos em múltiplas nuvens pode ser considerado da perspectiva de um arquiteto de *software* e do provedor. Diante deste contexto, o gerenciamento de recursos desempenha um papel importante na distribuição e execução de uma aplicação distribuída e, consequentemente, na interoperabilidade e na portabilidade, tornando o ambiente de múltiplas nuvens mais flexível ao oferecer os recursos mais adequados para atender aos requisitos de uma aplicação e de um arquiteto de *software*, tanto na implantação quanto na execução de uma aplicação.

Em relação à flexibilidade, a gestão de recursos ainda é um desafio em aberto. No entanto é possível abordá-lo observando três aspectos importantes: (*i*) o papel real do gerenciamento de recursos, (*ii*) a definição de recursos em múltiplas nuvens e (*iii*) as funções fornecidas pelo gerenciamento de recursos. Além disso, na literatura existem vários trabalhos relacionados à gestão de recursos em nuvem, porém quase sempre sob a perspectiva do provedor. Para que se possa distribuir uma aplicação em vários provedores de nuvem e, consequentemente, mitigar o *vendor lock-in*, é necessário tratar o gerenciamento de recursos da perspectiva do provedor e do arquiteto de *software*. Para isso, é necessário definir e classificar os recursos em múltiplas nuvem e determinar as funções para gerenciar recursos da perspectiva do usuário.

Além disso, de acordo com Zhan *et al.* (2015), tanto a academia quanto a indústria veem o problema de gerenciamento de recursos em nuvem como um problema de otimização de

NP-difícil. Assim, algoritmos evolutivos tem recebido considerável atenção nos últimos anos para promover o gerenciamento de recursos em nuvem por ofertar uma solução global para o problema NP-difícil, aceitável em um período de tempo proporcional a diversas variáveis. Outra razão para o sucesso de algoritmos evolutivos é no escalonamento de recursos para computação em grade. Além disso, mais de 17,5% dos trabalhos que abordam o gerenciamento de recursos utilizam computação evolutiva (ZHAN *et al.*, 2015).

O principal objetivo deste capítulo é mostrar o estado-da-arte do gerenciamento de recursos para múltiplas nuvens. Assim, a primeira contribuição deste trabalho é uma classificação proposta para as demandas de um arquiteto de *software* para implantação e execução de aplicações em múltiplas nuvens, permitindo que o gerenciador de recursos atenda melhor às necessidades dos usuários. A segunda e terceira contribuições estão relacionadas à identificação de recursos em múltiplas nuvens, para otimizar e atender a todos os recursos. Em seguida, a quarta, quinta e sexta contribuições são as taxonomias propostas para alcançar o gerenciamento de recursos em múltiplas nuvens (MCRM) e atender aos requisitos de um arquiteto de *software*. A sétima contribuição, por sua vez, é identificar os benefícios da computação evolucionária para o gerenciamento de recursos em múltiplas nuvens, além de verificar lacunas e determinar a possibilidade de tratamento utilizando computação evolucionária. Finalmente, a última contribuição é uma descrição dos principais *surveys* relacionados ao gerenciamento de recursos, bem como uma comparação com o trabalho aqui apresentado. Em resumo, as contribuições deste capítulo são dadas abaixo:

- A Seção 2.2 propõe uma classificação para as demandas dos arquitetos de *software* para implantação e execução de aplicações em múltiplas nuvens.
- Na Seção 2.3.1, propõe-se uma definição e uma classificação para recursos em múltiplas nuvens (MCR).
- Na Seção 2.3.2, três taxonomias relacionadas ao MCRM da perspectiva de um arquiteto de *software* são propostas. A primeira e a terceira taxonomia determinam quando e como gerenciar o MCR, respectivamente. A segunda determina como configurar os requisitos de um arquiteto de *software* para o MCR.
- Na Seção A no Apêndice A, investiga-se os benefícios da computação evolutiva para o gerenciamento de recursos em múltiplas nuvens e o preenchimento dos *gaps* de gerenciamento de recursos através de computação evolutiva.
- Na Seção A no Apêndice A, uma breve descrição é realizada sobre outros *surveys* rela-

cionados ao gerenciamento de recursos. Além disso, verifica-se quais deles focam nas demandas de um arquiteto de *software* no gerenciamento de recursos em múltiplas nuvens para a implantação e execução de uma aplicação. Nota-se que nenhum deles aborda o gerenciamento de recursos em múltiplas nuvens da perspectiva de um usuário. Portanto, até a escrita deste documento, nenhum trabalho na literatura havia considerado o gerenciamento de recursos em múltiplas nuvens a partir da perspectiva de um arquiteto de *software* usando computação evolucionária.

Este capítulo está organizado da seguinte maneira. A Seção 2.1 apresenta um cenário de múltiplas nuvens, oferecendo definições, características, vantagens e desafios, ou, mais precisamente, o desafio do gerenciamento de recursos em múltiplas nuvens. Além disso, propõe as demandas de um arquiteto de *software* para implantação e execução de aplicações em múltiplas nuvens. A seção 2.3 oferece uma definição e uma classificação para recursos em múltiplas nuvens. Em seguida, propõe três taxonomias relacionadas ao gerenciamento de recursos para múltiplas nuvens. Além disso, descreve algumas taxonomias descritas na literatura, destacando suas limitações relacionadas às demandas dos usuários por gerenciamentos de recursos em múltiplas nuvens. A Seção A descreve alguns dos trabalhos mais importantes relacionados à promoção do gerenciamento de recursos em múltiplas nuvens através da computação evolucionária. Além disso, discuti esses trabalhos para verificar se eles atendem a todas as demandas de um usuário para implantação e execução de aplicações em múltiplas nuvens. A Seção A descreve alguns trabalhos que apresentam um *survey* sobre gerenciamento de recursos de computação em nuvem e mostra as diferenças relacionadas à esta abordagem. A seção A discute os desafios e as direções futuras relacionadas ao gerenciamento de recursos em múltiplas nuvens da perspectiva de um arquiteto de *software*.

2.1 Múltiplas Nuvens

O ambiente de múltiplas nuvens permite que as aplicações aproveitem os melhores recursos de diferentes componentes oferecidos por vários provedores de nuvem. Esses benefícios de múltiplas nuvens aumentaram o interesse da comunidade acadêmica em investigar seus desafios para gerenciar, implantar e monitorar uma aplicação distribuída e provedores de nuvem do ponto de vista do arquiteto de *software* e do provedor. Como a indústria começou a desenvolver pesquisas na área antes da academia, ela já oferece algumas soluções proprietárias. Embora já existam algumas soluções, não existe ainda uma taxonomia padronizada para o assunto, e

diferentes pesquisadores usam termos distintos para se referir a conceitos iguais. Nesta tese de doutorado adota-se as definições propostas por Petcu (2014c), que conduziu uma pesquisa de taxonomia de múltiplas nuvens, e mostrou que a comunidade acadêmica criou vários termos para descrever soluções arquitetônicas específicas.

As principais razões para o uso de múltiplas nuvens foram descritas por vários pesquisadores, dentre eles Petcu (2014c), Grozev e Buyya (2014) e Toosi *et al.* (2014): escalabilidade, ampla disponibilidade de recursos, recuperação de desastres, distribuição geográfica, baixa latência de acesso, questões legais, regulamentação, eficiência de custos, economia de energia, interoperabilidade e combate ao *vendor lock-in*. Estes motivos permitem redimensionar dinamicamente sua capacidade e a cooperação entre vários provedores para tratar os problemas relacionados a padrões de uso de serviço e implantação personalizada de uma aplicação na nuvem. O problema associado aos padrões de uso de serviço estabelece que estes podem variar com o tempo e, na maioria das vezes, de maneira imprevisível, o que pode sobrecarregar um único provedor de nuvem, levando à interrupção do serviço ou tornando-os não confiáveis. Na implantação de uma aplicação na nuvem, a existência de interoperabilidade entre provedores de nuvem elimina a obrigação de personalização de padrões de uso. Além disso, o desenvolvimento e implantação de aplicações em múltiplas nuvens altamente disponíveis são permitidos garantindo qualidade de serviço e desempenho. As múltiplas nuvens também por ser uma solução que atenda às demandas de usuários por serviços dispersos geograficamente, que exigem baixo tempo de resposta e uso simultâneo de múltiplas nuvens. O cenário de múltiplas nuvens ainda fornece oportunidade para provedores de nuvem identificarem outros provedores capazes de cumprir com as regulamentações relacionadas à localização do *data center*. Por fim, través de um ambiente de múltiplas nuvens é possível reduzir o consumo de energia promovendo a eficiência do uso da infraestrutura de computação.

O uso de múltiplas nuvens traz várias vantagens que permitem alcançar vários benefícios da computação em nuvem, já que os usuários podem escolher os serviços mais adequados às suas necessidades, como serviços com maior elasticidade ou melhor preço. No entanto, múltiplas nuvens também trazem vários desafios, como problemas relacionados à interoperabilidade e à portabilidade. Para resolver os problemas associados à portabilidade e interoperabilidade é necessário conhecer os vários recursos da nuvem e entender como e quando gerenciá-los (MEZGÁR; RAUSCHECKER, 2014), (SILVA *et al.*, 2013) e (PETCU, 2011).

Segundo Rehman *et al.* (2015), para os desafios relacionados a interoperabilidade e

portabilidade os arquitetos de *software* precisam tomar decisões importantes de gerenciamento de serviços em nuvem com base nos aspectos de qualidade de serviço (QoS) e outros critérios, como o custo total para alocar os recursos necessários para executar aplicações e dados gerais de desempenho de acordo com a distribuição dos componentes da aplicação em diferentes provedores de serviços. Além disso, existem desafios da perspectiva do arquiteto de *software* que devem ser tratados. Ainda de acordo com Rehman *et al.* (2015), o gerenciamento de serviços de nuvem da perspectiva do arquiteto de *software* possui duas fases. A primeira fase se inicia com a seleção de um provedor de nuvem para instanciar um serviço pela primeira vez. A segunda fase monitora o desempenho dos serviços tanto para aqueles alocados quanto para os disponíveis, de modo a avaliar o nível de QoS e decidir se serviços continuam ou não no mesmo provedor de nuvem.

Além disso, para tratar problemas relacionados ao gerenciamento de recursos para várias nuvens faz-se necessário definir as demandas que um arquiteto de *software* exige para implantar uma aplicação em múltiplas nuvens. Por este motivo, Petcu (2014b) definiu uma classificação de requisitos funcionais para o *Cloudware Multi-Cloud*, um *software* que permite construir, implantar, executar e gerenciar aplicações em um ambiente de computação em nuvem. A classificação proposta é apresentada na Tabela 1 e enfoca na perspectiva de um desenvolvedor e de um usuário. Os requisitos da perspectiva de um desenvolvedor estão na forma de restrições de princípios e ferramentas. No entanto, os da perspectiva de um usuário estão associados às fases de desenvolvimento, implantação e execução.

TABELA 1. REQUISITOS FUNCIONAIS PARA O AMBIENTE DE *Multi-Cloud* (PETCU, 2014B).

	Desenvolvimento	Implantação			Execução
Ferramentas	Portal/Serviço como ponto de entrada	Meta-alocador serviço/recurso	Implantador genérico	Mecanismo de pesquisa	Meta-monitor para aplicações
	Serviços extras agnósticos a nuvem	Meta-escalonador	Implantador semi-automatizado	Serviço de correspondência	Meta-monitor para serviços/recursos
	Interface para requisitos do usuário	Meta-auto-scaler e balanceador de carga	Mecanismos de rede virtual	Serviço seletivo	Driver de ciclo de vida de uma aplicação/serviço
	Serviço de integração	Depurador e testador	Gerenciamento de credenciais	Sistema de recomendação	Controle de QoS e mecanismos de aviso
Princípios	Suporte à portabilidade	Preservação de particularidades	Junção perfeita entre nuvens	Sem restrições nas nuvens	Usar protocolos padrão
	Interfaces abstratas de controle de serviço	Usar interfaces padrão	Suporte para os principais provedores de nuvem	Permitir alocação dinâmica de recursos	*Overhead* pequeno

2.2 Demandas de um Arquiteto de *Software* para Múltiplas Nuvens

Como mencionado anteriormente na seção 2.1, Petcu (2014b) descreve os requisitos gerais para múltiplas nuvens e Rehman *et al.* (2015) descreve as fases do gerenciamento de serviços da perspectiva de um usuário. Tais propostas buscam unificar conceitos sobre aplicações em múltiplas nuvens. Com base nestes trabalhos, propõe-se aqui uma classificação das demandas de um arquiteto de *software* para o gerenciamento de aplicações em múltiplas nuvens, como mostra a Figura 2. O gerenciamento de aplicações é responsável por verificar todos os requisitos de uma aplicação e de um arquiteto de *software* e por abordá-los de maneira mais próxima ao ideal, usando funcionalidades, como gerenciamento de recursos, monitoramento e o serviço de SLA. Ressalta-se que estas funcionalidades de gerenciamento de aplicações podem ser solicitadas antes ou depois da implantação de uma aplicação.

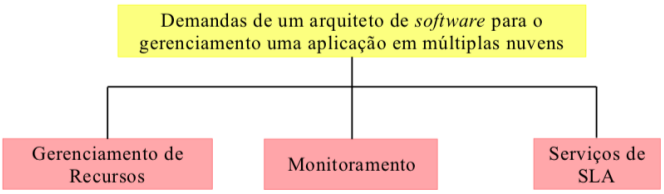


Figura 2. As demandas de um Arquiteto de *Software* para Gerenciamento de Aplicações em Múltiplas Nuvens.

O gerenciamento de recursos destina-se principalmente (*i*) a verificar quais recursos atendem melhor aos requisitos das aplicações e dos arquitetos de *software*, e (*ii*) a adquirir ou a liberar recursos quando necessário. O monitoramento, como o nome sugere, é responsável pelo monitoramento dos requisitos tanto de uma aplicação quanto de um arquiteto de *software*, bem como pelo monitoramento dos recursos de um provedor de nuvem, além de solicitar uma redistribuição, se necessário. Outra função do monitoramento é a análise de dados relacionada aos recursos e à aplicação, a qual deve ser enviada ao serviço de SLA. Finalmente, o Serviço de SLA é responsável por atualizar os requisitos de um arquiteto de *software* de uma aplicação antes da implantação e em tempo de execução, além de relatar o desempenho de uma aplicação e de uso de recursos.

2.3 Classificação e Taxonomias

De acordo com as demandas de um arquiteto de *software* mencionadas previamente e outras identificadas na literatura (GUZEK *et al.*, 2015), propõe também neste trabalho uma

definição e uma classificação para CMCR, a ser melhor detalhada na na Subseção 2.3.1. Três taxonomias são também propostas para atender os requisitos de uma aplicação da perspectiva de um arquiteto de *software*, melhor detalhadas na Subseção 2.3.1. Essas três taxonomias visam apresentar (*i*) quando , (*ii*) como gerenciar e (*iii*) como configurar recursos em múltiplas nuvens de acordo com os requisitos de um arquiteto de *software* para uma aplicação. Além disso, na Subseção 2.3.3, apresenta-se algumas classificações e taxonomias reunidas a partir de uma revisão bibliográfica, onde destaca-se as limitações relacionadas às demandas de um arquiteto de *software* para o gerenciamento de recursos em múltiplas nuvens.

2.3.1 Definição e Classificação de Recursos em Múltiplas Nuvens

Até a escrita deste documento, nenhuma classificação de recursos de múltiplas nuvens na literatura já havia contemplado os requisitos de arquiteto de *software* em um ambiente de múltiplas nuvens.

Quando um arquiteto de *software* deseja distribuir uma aplicação em ambiente de múltiplas nuvens, ele necessita identificar quais são os recursos que podem ser alocados em cada provedor. Diante deste contexto, a fim de facilitar o gerenciamento de recursos em múltiplas nuvens propõe-se aqui uma definição para recursos em múltiplas nuvens a partir da perspectiva do arquiteto de *software*.

> **Definição (Recursos em Múltiplas Nuvens)**: recursos em múltiplas nuvens representam os componentes (recursos/serviços) necessários para implantação e execução de uma aplicação no ambiente de múltiplas nuvens, observando os requisitos de uma aplicação e de um arquiteto de *software*.

Já a classificação para recursos em múltiplas nuvens se faz necessária, pois para cada tipo de recurso existem diferentes tipos de requisitos. Esta classificação tem como objetivo auxiliar no gerenciamento de recursos, principalmente sobre mapeamento de requisitos e análise de dados.

> **Classificação(Recursos em Múltiplas Nuvens)**: a classificação de recursos em múltiplas nuvens é composta por dois níveis. O primeiro é o nível abstrato com três tipos de recursos: (*i*) de *hardware*, (*ii*) de *software* e (*iii*) de nuvem. O segundo é o nível concreto e possui todos os componentes que podem ser exigidos por uma aplicação e por um usuário.

A Figura 3 ilustra a classificação dos recursos em múltiplas nuvens e apresenta os níveis abstrato e concreto.

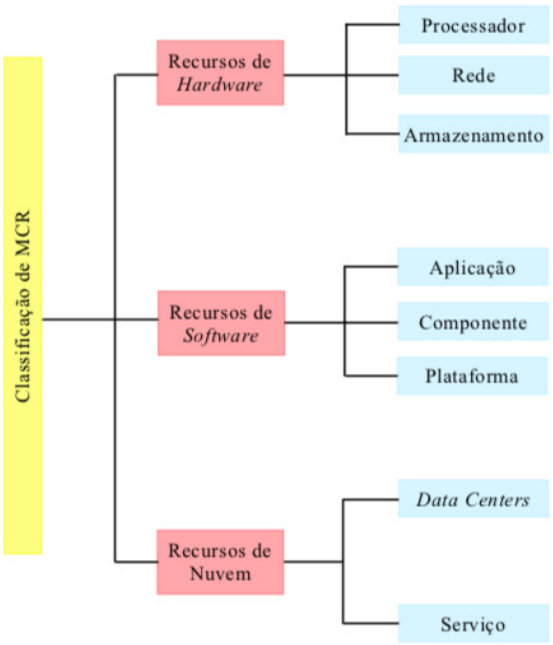


Figura 3. Classificação de Recursos em Múltiplas Nuvens (CMCR).

2.3.2 Taxonomias para o Gerenciamento de Recursos em Múltiplas Nuvens

Durante as pesquisas realizadas nesta tese, nenhuma taxonomia de recursos contempla os requisitos de um arquiteto de *software* em um ambiente de múltiplas nuvens. Nesta subseção, três taxonomias são propostas. A primeira determina quando gerenciar os recursos em múltiplas nuvens. Já a segunda determina como requisitos de um arquiteto de *software* em múltiplas nuvens podem ser configurados. Por fim, a última taxonomia proposta determina como gerenciar recursos em múltiplas nuvens, de acordo com as demandas de um arquiteto de *software* descritas na Subseção 2.2.

A Figura 4 ilustra a taxonomia proposta para se determinar quando gerenciar recursos em múltiplas de nuvens, tanto da perspectiva de um arquiteto de *software* quanto da perspectiva de um provedor.

No segundo nível da taxonomia, a perspectiva de um arquiteto de *software* é dividida em duas fases:pre-implantação e pós-implantação. O terceiro nível da taxonomia apresenta as situações nas quais os recursos podem ser gerenciados em múltiplas nuvens, tanto da perspectiva de um arquiteto de *software* quanto da perspectiva de um provedor. Da perspectiva de um

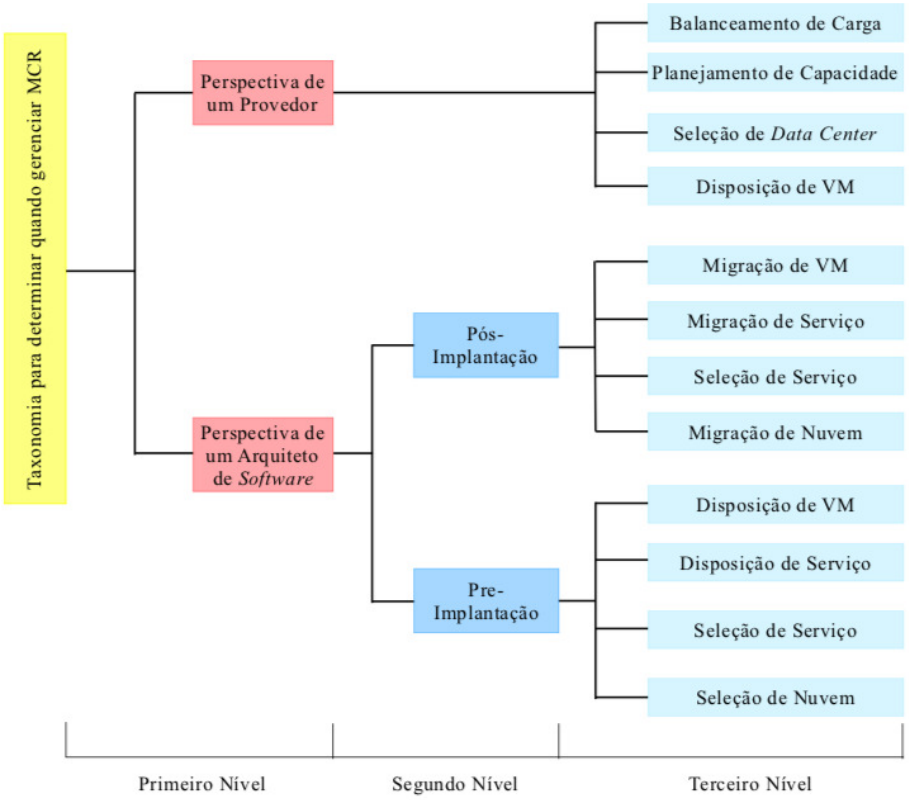


Figura 4. Taxonomia para determinar quando gerenciar MCR.

arquiteto de *software*, na pré-implantação, as situações que podem gerenciar recursos são: seleção de Nuvem, seleção de Serviço, posicionamento de serviço e de máquinas virtuais (*Virtual Machine* (VM)). Na seleção de nuvem um arquiteto de *software* precisa determinar a modularização de uma aplicação. A Figura 5 apresenta duas opções: única nuvem e múltiplas nuvens. No primeiro caso, a aplicação é implantada em apenas um provedor e, no segundo caso, ela deve ser implantada em múltiplas nuvens. Na taxonomia proposta, um arquiteto de *software* possui três opções para modularizar uma aplicação. A opção Grupo de Tarefas representa uma aplicação modularizada em vários grupos de tarefas (microsserviços, por exemplo), onde cada grupo de tarefas pode ser implantado em um provedor. As outras duas opções, baseiam-se em Tarefa e Serviço de uma aplicação, e podem ser implantadas em provedores de nuvem distintos. Neste trabalho, Tarefa representa um conjunto de serviços de nuvem. Em seguida, a seleção da nuvem deve basear-se nos requisitos de um arquiteto de *software* para um Grupo de Tarefas, Tarefa ou Serviço. Para as demais situações da perspectiva de um arquiteto de *software* tanto na pré-implantação quanto na pós-implantação observa-se os requisitos de um arquiteto de *software*, além dos funcionais, para selecionar e colocar os serviços e/ou máquinas virtuais (VMs) para executar uma aplicação em um provedor de nuvem.

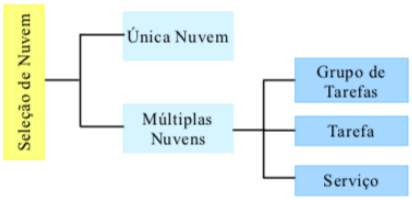


Figura 5. Modularização de uma Aplicação para implantação.

Da perspectiva de um provedor as situações que podem ser gerenciadas são: posicionamento de VM, seleção de *data center*, planejamento de capacidade e balanceamento de carga. Elas precisam ser gerenciadas pelo provedor de nuvem, através da observação dos requisitos de um arquiteto de *software*, de maneira a aumentar a lucratividade do provedor.

Conforme descrito nesta seção, para tornar possível o gerenciamento de recursos em múltiplas nuvens, é necessário configurar os requisitos de um arquiteto de *software*. Para isso, existem alguns aspectos que devem ser analisados: o estado,a multiplicidade e a granularidade dos requisitos. Assim, uma taxonomia é proposta para determinar como configurar os requisitos de um arquiteto de *software*, a qual é ilustrada na Figura 6. O estado de um requisito pode ser: estático, dinâmico e/ou híbrido. O estado estático indica que os requisitos são conhecidos antes da implantação de uma aplicação. O estado dinâmico indica que os requisitos aparecem ao longo do tempo (GUZEK *et al.*, 2015). Por fim, os requisitos podem ser configurados nos estados estático e dinâmico, indicando que os requisitos podem ser conhecidos antes da implantação de uma aplicação e alterados após a implantação.

No aspecto de multiplicidade, um arquiteto de *software* pode configurar um ou vários requisitos para uma aplicação. Se vários requisitos forem configurados, as prioridades, a otimização ou ambas devem ser definidas. A prioridade determina o peso de cada requisito e a otimização determina quais requisitos devem ser maximizados ou minimizados. A granularidade significa a definição dos requisitos de um arquiteto de *software* para uma aplicação, grupo de tarefas, tarefa ou serviço. Se a definição dos requisitos for feita para uma aplicação, isso indicará que todos os componentes de uma aplicação têm os mesmos requisitos, mas se os requisitos forem definidos para um Grupo de tarefas, por Tarefa ou por Serviço, isso indicará que cada Grupo de tarefas, Tarefa ou Serviço de uma aplicação possui requisitos distintos.

Por fim, uma taxonomia é proposta para determinar como gerenciar os recursos em múltiplas nuvens, a qual é ilustrada na Figura 7. O aspecto modo indica se existe ou não a necessidade de intervenção humana durante o processo de gerenciamento de recursos. O modo automático indica que todo gerenciamento de recursos é feito sem intervenção de um arquiteto

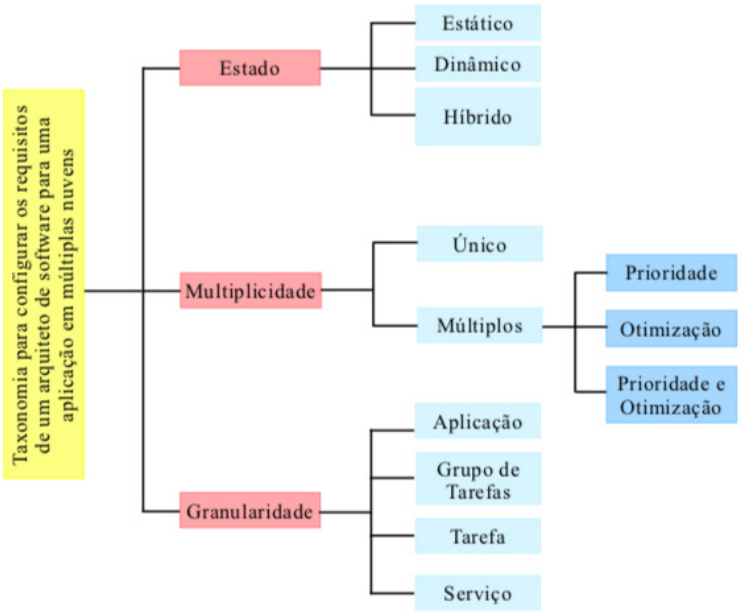


Figura 6. Taxonomia para configurar os requisitos de um Arquiteto de *Software* para uma Aplicação em Múltiplas Nuvens.

de *software*, mas este pode definir se as soluções devem ser ótimas ou se devem ser viáveis. O modo interativo indica que o sistema de gerenciamento deve interagir com um arquiteto de *software* para tomar uma decisão. No aspecto de granularidade o gerenciamento de recursos deve ser definido de acordo com a definição de requisitos de um arquiteto de *software*. As definições de requisito feitas por um arquiteto de *software* podem ser para a aplicação, ou para um Grupo de Tarefas, ou por Tarefa e ainda por Serviço.

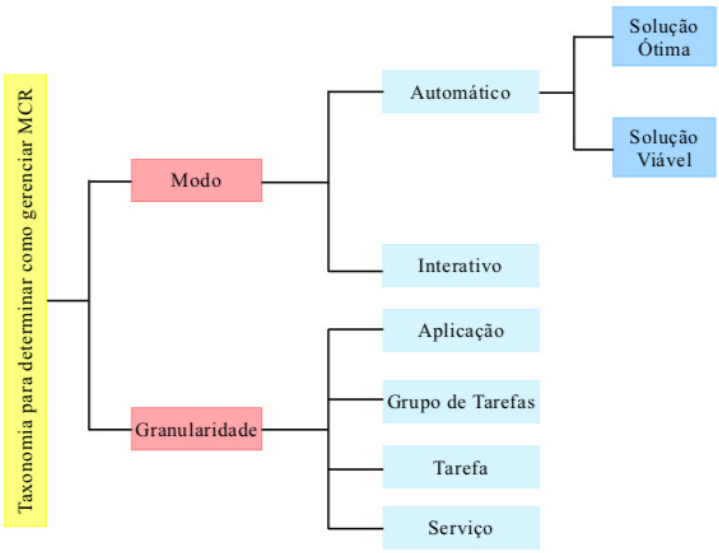


Figura 7. Taxonomia para determinar como gerenciar recursos em múltiplas nuvens.

2.3.3 Taxonomias de Gerenciamento de Recursos em Nuvens na Literatura

Como mencionado anteriormente, considera-se que para gerenciar recursos em múltiplas nuvens, deve-se determinar quando e como gerenciar, além de como configurar os requisitos de acordo com as metas de um arquiteto de *software*.

A fim de apresentar as taxonomias descritas na literatura para o gerenciamento de recursos de múltiplas nuvens da perspectiva de um arquiteto de *software*, uma pesquisa foi realizada em algumas das principais bases de pesquisa, a saber, ScienceDirect, ACM DL e IEEE. Como resultado, obteve-se oito taxonomias. Nesta subseção, as principais características destas taxonomias são apresentadas na Tabela 2. Em seguida, discuti-se suas limitações relacionadas às demandas do arquiteto de *software* para o gerenciamento de recursos em múltiplas nuvens.

A Tabela 2 apresenta três características relacionadas às oito taxonomias: perspectiva, modelo de entrega e meta para identificar se elas atendem ou não às demandas de um arquiteto de *software* para o ambiente de múltiplas nuvens. A primeira característica indica a partir de qual perspectiva a taxonomia trata o gerenciamento de recursos. As perspectivas de gerenciamento de recursos podem ser a partir de um provedor, de um arquiteto de *software*, ou de ambos. Já a segunda característica tem por objetivo apresentar o modelo de entrega abordado pela taxonomia. Os modelos de entrega para ambientes de múltiplas nuvens são: *cloud provider*, *cloud federation*, *multi-cloud* ou *inter-cloud*. Por fim, a última característica descreve o objetivo principal da taxonomia, pois percebe-se que não há padronização de termos relacionados ao gerenciamento de recursos. Alguns termos são empregados de maneira distinta por diferentes pesquisadores, indicando que o gerenciamento de técnicas de recursos pode significar o gerenciamento de recursos para alguns pesquisadores, e gerenciamento de contexto para outros. Assim, também existem trabalhos que propõe uma taxonomia de gerenciamento de recursos com diferentes pontos de vista.

Nota-se que apenas uma das taxonomias lida com a perspectiva de um arquiteto de *software*, três delas lidam com múltiplas nuvens, e a maioria delas trata apenas de um aspecto de gerenciamento de recursos, como gerenciamento de contexto e gerenciamento de serviços. Nenhuma delas aborda aspectos relativos a quando e como gerenciar recursos em múltiplas nuvens, conforme descrito no início desta subseção.

Embora nenhuma dessas taxonomias apresente as necessidades do MCRM a partir da perspectiva de um arquiteto de *software*, a Tabela 3 mostra os aspectos contemplados por eles observando as taxonomias descritas nesta tese. Objetiva-se com isso verificar quais demandas de

TABELA 2. RESUMO DAS TAXONOMIAS PARA GERENCIAMENTO DE RECURSOS NA NUVEM

Taxonomia	Perspectiva	Modelo de Entrega	Objetivo da Taxonomia
Guzek *et al.* (2015)	Provedor	*Cloud*	Uma classificação para as questões dinâmicas de CRM.
Zhan *et al.* (2015)	Provedor	*Cloud*	Escalonamento de recursos para computação em nuvem.
Mustafa *et al.* (2015)	Provedor	*Hybrid* e *Mobile*	Contexto de CRM.
Rehman *et al.* (2015)	Usuário	*Cloud*	Fasse e processo de gerenciamento de serviços de nuvem.
Singh e Chana (2016b)	Provedor	*Cloud*	Funções do CRM.
(PARIKH *et al.*, 2017)	Provedor	*Cloud*	Processo sequencial do gerenciamento de recursos.
Liaqat *et al.* (2017)	Provedor	*Federated Cloud*	Problemas e soluções das funções de gerenciamento de recursos.
Gonzalez *et al.* (2017)	Provedor	*Multiple clouds*	Requisitos para múltiplas nuvens para *workflows* intensividade de dados e The multiple cloud requirements for data-intensive workflows and alto nível de mecanismos dinâmicos.

um arquiteto de *software* para a implantação e execução de aplicações em múltiplas nuvens são atendidas por estas taxonomias.

Na Tabela 3, é possível observar que a maioria das taxonomias propostas na literatura aborda alguns aspectos relacionados a quando a gerenciar recursos, mas as taxonomias que atendem a estes aspectos focam na perspectiva do provedor, e em um único provedor de nuvem. A taxonomia que lida com gerenciamento de recursos da perspectiva de um arquiteto de *software* aborda apenas aspectos relacionados à seleção e migração de serviços. Quanto à configuração dos requisitos de um arquiteto de *software*, a maioria delas configura tanto o estado estático quanto o dinâmico, define multiplicidade, embora neste caso não definam prioridades e otimização e não definem a granularidade dos requisitos. E em relação a como gerenciar nenhuma delas trata deste aspecto.

De acordo com os trabalhos encontrados, até a escrita deste documento as demandas de um arquiteto de *software* no gerenciamento de recursos para implantação e execução de aplicações distribuídas em múltiplas nuvens não são atendidas por nenhuma taxonomia descrita na literatura como proposto neste trabalho.

2.4 Resumo do Capitulo

De acordo com as demandas dos arquitetos de *software* para o gerenciamento de aplicações em múltiplas nuvens, este capítulo propõe três taxonomias relacionadas ao gerenciamento de recursos em múltiplas nuvens da perspectiva de um arquiteto de *software*, para que fosse possível definir essas taxonomias, antes propôs-se uma definição e uma classificação de

TABELA 3. RESUMO DAS TAXONOMIAS DA LITERATURA EM RELAÇÃO AO MCRM DESCRITO NAS FIGURAS 4, 6, E 7.

Taxonomia	Gerenciamento de Recursos em Múltiplas Nuvens (MCRM)												
	Quando Gerenciar								Como Configurar Requisitos			Como Gerenciar	
	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XI
Guzek *et al.* (2015)	✘	✘	✔	✔	✘	✘	✘	✘	Híbrido	Múltiplo	✘	✘	✘
Zhan *et al.* (2015)	✘	✔	✘	✘	✘	✘	✘	✘	✘	Múltiplo	✘	✘	✘
Mustafa *et al.* (2015)	✘	✘	✘	✘	✘	✘	✘	✘	✘	Múltiplo	✘	✘	✘
Rehman *et al.* (2015)	✘	✔	✘	✘	✘	✔	✔	✘	Híbrido	Múltiplo	Serviço	✘	✘
Singh e Chana (2016b)	✘	✔	✘	✘	✘	✔	✘	✘	Híbrido	Múltiplo	✘	✘	✘
Parikh *et al.* (2017)	✔	✔	✘	✘	✘	✔	✔	✔	Híbrido	Múltiplo	✘	✘	✘
Liaqat *et al.* (2017)	✘	✔	✘	✘	✘	✔	✘	✘	Híbrido	Múltiplo	✘	✘	✘
Gonzalez *et al.* (2017)	✔	✘	✘	✘	✘	✘	✘	✘	✘	Múltiplo	Workflow	✘	✘
Taxonomias propostas por esta Tese Carvalho *et al.* (2018a)	✔	✔	✔	✔	✔	✔	✔	✔	Híbrido	Múltiplo	✔	✔	✔

I:SELEÇÃO DE NUVEM; II: SELEÇÃO DE SERVIÇO NA PRE-IMPLANTAÇÃO; III: DISPOSIÇÃO DE SERVIÇO; IV: DISPOSIÇÃO DE VM; V: MIGRAÇÃO DE NUVEM; VI: SELEÇÃO DE SERVIÇO PÓS-IMPLANTAÇÃO; VII: MIGRAÇÃO DE SERVIÇO; VIII: MIGRAÇÃO DE VM; IX: ESTADO; X: MULTIPLICIDADE; XI: GRANULARIDADE; XII:MODO

recursos em múltiplas nuvens.

A partir das taxonomias propostas, investigou-se trabalhos que propõem uma solução para o problema de gerenciamento de recursos em múltiplas nuvens a partir da perspectiva de um arquiteto de *software* usando computação evolucionária. Ao perceber que poucos trabalhos tratam do gerenciamento de recursos sob esta perspectiva, investigou-se os trabalhos que propõem uma solução para o problema de gerenciamento de recursos em múltiplas nuvens usando computação evolutiva independentemente da perspectiva, mas poucos aspectos propostos nas taxonomias são atendidos pelo soluções pesquisadas. Portanto, como descrito neste capítulo, vários aspectos precisam ser abordados para ter o gerenciamento de recursos da perspectiva de um arquiteto de *software*, e existem vários desafios, porque o problema de gerenciamento de recursos em múltiplas nuvens é um problema NP difícil. As direções futuras para o gerenciamento de recursos da perspectiva de um arquiteto de *software* que está de acordo com as taxonomias propostas neste trabalho e que utilizam computação evolutiva foram apontadas, pois através da computação evolutiva é possível obter soluções aceitáveis em um tempo proporcional a diversas variáveis.

Por fim, investigou-se trabalhos que abordam o gerenciamento de recursos em nuvem e pode-se observar que embora existam vários deles, poucos focam em múltiplas nuvens, bem como poucos focam na perspectiva de um arquiteto de *software* de forma abrangente, como nas taxonomias propostas nesta tese.

3 APLICAÇÕES DISTRIBUÍDAS EM MÚLTIPLAS NUVENS

Para uma aplicação obter os benefícios da computação em nuvem de forma que cada componente de uma aplicação possa ser implantado em um provedor de nuvem distinto, é necessário projetar a aplicação para ser implantada na computação em nuvem. Para isso, é importante que as aplicações sejam desenvolvidas usando os conceitos e tecnologias relacionadas à criação de aplicações com pensamento em características intrínsecas da nuvem. Um estilo arquitetural que tem sido utilizado com bastante êxito é o baseado em microsserviços. Embora o desenvolvimento de uma aplicação nativa para nuvem baseada em microsserviços traga muitos benefícios em relação ao gerenciamento da implantação e monitoramento em um ambiente *multi-cloud*, um arquiteto de *software* deve tomar várias decisões em relação à implantação e monitoramento de uma aplicação distribuída em múltiplos provedores de nuvem. Assim, com o objetivo de facilitar as tomadas de decisões de um arquiteto de *software*, as contribuições deste capítulo são:

- A proposição de uma nova abordagem para gerenciar a implantação e a execução de uma aplicação baseada em microsserviços distribuída em um ambiente *multi-cloud* da perspectiva de um arquiteto de *software* nomeada de PacificClouds.
- A realização de uma comparação entre PacificClouds e outros trabalhos similares encontrados na literatura. Esta comparação tem por objetivo observar quais outros benefícios o PacificClouds deve ter em relação a iniciativas similares para o gerenciamento da implantação, execução de uma aplicação em ambiente *multi-cloud*.

A fim de alcançar de mostrar os benefícios e desafios relacionados ao desenvolvimento de aplicações nativas para nuvem, bem como de alcançar os objetivos este capítulo está organizado da seguinte maneira. Na Seção 3.1, descreve conceitos relacionados e características de uma aplicação nativa para nuvem. Na Seção 3.2, apresenta conceitos, benefícios, características e desafios de microsserviços, além de propor uma definição para microsserviços, a qual consideremos ser mais completa dentre as definições que encontramos até a escrita deste documento. Na Seção 3.3 descreve um cenário, no qual mostra as principais tomadas de decisões de um arquiteto de *software* em relação a implantação e monitoramento de uma aplicação em um ambiente *multi-cloud*. Na Seção 3.4 oferece uma descrição da arquitetura do PacificClouds, e a relação da arquitetura com um arquiteto de *software*, e com os provedores de nuvem disponíveis. Finalmente, na Seção 3.5 compara PacificClouds com outros trabalhos na literatura.

3.1 Aplicação Nativa para Nuvem

De acordo com Sosinsky (2011), a computação em nuvem é descrita como um modelo de entrega de serviços que compreende cinco características essenciais, três modelos de serviços principais e quatro modelos de implantação principais, ilustrados na Figura 8. As características essenciais, como o reservatório de recursos, o amplo acesso à rede, o serviço medido, o autoatendimento sob demanda e a rápida elasticidade podem reduzir a infraestrutura e o custo pessoal. Os três modelos de serviço amplamente adotados são *Infrastracture as a Service* (IaaS), *Platform as a Service* (PaaS) e SaaS. Um provedor de IaaS gerencia toda a infraestrutura, enquanto os usuários são responsáveis pelos outros aspectos da implantação. Um provedor de PaaS gerencia a infraestrutura, os sistemas operacionais e o *software*, que estão habilitados para a plataforma em questão, enquanto os usuários são responsáveis pelo gerenciamento da aplicação que eles implantam. No modelo SaaS, a responsabilidade do cliente é inserir e gerenciar seus dados e interações, e a responsabilidade do provedor é gerenciar todas as aplicações que estão na infraestrutura. Os modelos de implantação podem ser comunitários, híbridos, privados e públicos. Eles especificam como uma infraestrutura de nuvem é construída, gerenciada e acessada, e definem a finalidade e a localização da nuvem (SCHMADER, 2015).

FONTE: BASEADA EM SOSINSKY (2011).

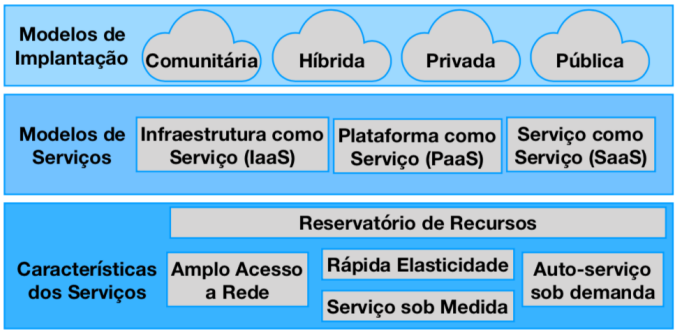


FIGURA 8. CARACTERÍSTICAS DA COMPUTAÇÃO EM NUVEM.

Segundo Fehling *et al.* (2014), as aplicações nativas para nuvem são criadas com foco na integração com o modelo de nuvem, para obter as vantagens da computação em nuvem. Uma aplicação nativa para nuvem garante outras características além daquelas derivadas da computação em nuvem. Nesta seção, apresenta-se as razões para desenvolver aplicações nativas para nuvem.

Uma aplicação nativa para nuvem de acordo com Fehling *et al.* (2014) deve incorpo-

rar as características rotuladas como IDEAL (***I**solated state, **D**istribution, **E**lasticity, **A**utomated Management, **L**oose coupling*). A característica estado isolado (*Isolated State*) está relacionado a como uma aplicação manipula um estado. Cada componente de uma aplicação pode optar tanto pelo *stateful* (com estado) quanto pelo *stateless* (sem estado). Componentes com estado carregam o estado se o componente, a plataforma ou a infraestrutura falhar. Componentes sem estado podem ser adicionados e facilmente removidos, em virtude de seu estado não transitório ser atribuído a componentes com estado ou a armazenamento externo. A distribuição também é uma importante característica em virtude de que muitos recursos de TI estejam possivelmente distribuídos globalmente. A elasticidade (*Elasticity*) é uma característica na qual é possível aumentar ou reduzir os recursos destinados a uma aplicação ou cliente para lidar com a variação da carga de trabalho. Gerenciamento automatizado (*Automated Management*) é uma característica na qual as tarefas de adicionar e remover recursos em tempo de execução devem ser automatizadas. O baixo acoplamento (*Loose coupling*) é uma característica na qual as dependências entre os componentes de uma aplicação devem ser minimizadas (FEHLING *et al.*, 2014).

As características descritas no parágrafo anterior estão de acordo com um tipo de manifesto mantido para aplicações nativas para nuvem e é chamado de aplicação de doze fatores. Muitos desses fatores interagem uns com os outros, sendo descritos a seguir conforme Sethi (2017). O primeiro fator, é chamado **base de código centralizado**, e especifica que todo o código implantado é rastreado no controle de revisão, e que seus códigos podem ter várias instâncias em várias plataformas. No segundo fator, chamado então de **gerenciamento de dependências**, define que uma aplicação deve declarar as dependências e isolá-las. **Definição de configuração** é o terceiro fator, no qual a variável de ambiente deve ser definida no nível do sistema operacional. O quarto fator é o **serviço de apoio** e indica que todo recurso é considerado parte de uma aplicação em si. No quinto fator, **isolamento na construção, atualização e ciclo de execução**, os artefatos devem ser construídos separadamente e, em seguida, combinados com a configuração. Em seguida, deve ser iniciada uma ou mais instâncias da combinação de artefato e configuração. No sexto fator, **processos sem estado**, uma aplicação deve executar uma ou mais instâncias que não compartilham nada. O sétimo fator, **associação de porta de serviços**, diz que todos os serviços de uma aplicação autônoma precisam ser expostos por meio de associação de porta. **Escalação de processos sem estado** é o oitavo fator, no qual a arquitetura deve se concentrar no gerenciamento de processos sem estado. No nono fator, em **gerenciamento de estado do processo**, os processos devem aumentar muito rapidamente e encerrar normalmente

dentro de um pequeno período de tempo. No décimo fator, **entrega contínua à produção**, os diferentes ambientes devem ser mantidos semelhantes. O décimo primeiro fator, ***logs* como fluxos de eventos**, ajudam a entender a tarefa da aplicação. Por fim, o décimo segundo fator, **tarefas *ad hoc* como processos *on-off***, as tarefas de gerenciamento executadas como parte de uma atualização devem ser executadas como processos únicos no ambiente.

De acordo com o descrito nesta seção, Sethi (2017) apresenta alguns pontos que devem ser considerados no desenvolvimento de uma aplicação nativa para nuvem, como segue. Um projeto informativo deve ser usado para aumentar o uso da aplicação com tempo e custo mínimos para os clientes que usam automação. A portabilidade de uma aplicação deve ser usada em diferentes ambientes e plataformas. A adequação de uma aplicação deve ser usada em plataformas de nuvem, e a alocação e o gerenciamento de recursos devem ser compreendidos. Ambientes idênticos devem ser usados para reduzir *bugs* e obter uma agilidade máxima de liberação de *software*. Por último, a alta disponibilidade deve ser ativada escalonando a aplicação com supervisão mínima e projetando arquiteturas de recuperação de desastre.

Como descrito nesta seção, aplicações nativas para nuvem são resultado de uma abordagem para criar e executar aplicações que exploram as vantagens do modelo de entrega de computação em nuvem. Assim, o objetivo de aplicações nativas para nuvem é como implementar e implantar, mas não onde implantar. Na Tabela 4 compara-se oito aspectos de aplicação nativa para nuvem com aplicações tradicionais, a qual está de acordo com Pivotal Software[1].

Em resumo, pode-se perceber que a grande diferença está no processo de desenvolvimento da aplicação. As aplicações nativas para nuvens focam no desacoplamento entre aplicação e infraestrutura, por meio de serviços independentes, na qual a entrega contínua é fundamental. O processo desenvolvimento de ser colaborativo e ter um comportamento previsível. Em decorrência destas características, promove-se uma escalabilidade automatizada e um recuperação rápida das falhas. Já as aplicações tradicionais possuem um processo de desenvolvimento e implantação lentos, com entregas periódicas, as equipes não integradas com dependências entre seus serviços, levando à comportamento imprevisível. Além disso, a escalabilidade é manual e a recuperação de falhas é lenta.

[1] https://pivotal.io/cloud-native
[2] https://pivotal.io/cloud-native

TABELA 4. DIFERENÇAS ENTRE APLICAÇÕES NATIVAS PARA NUVEM E APLICAÇÕES TRADICIONAIS.

Aspectos	Aplicações Nativas para Nuvem	Aplicações Tradicionais
Comportamento	**Previsível**: são projetadas para maximizar a resiliência e possui uma infraestrutura altamente automatizada.	**Imprevisível**: pela forma como e desenvolvida e implantada, o comportamento das mesmas não pode ser previsível.
Sistema Operacional	**Abstração**: através de plataformas abstraem a dependências da infraestrutura.	**Dependência**: constroem dependências entre a aplicação e o sistema operacional.
Recursos	**Adequado**: automatiza o provisionamento e a configuração da infraestrutura, alocando e realocando recursos dinamicamente no tempo de implantação, com base nas necessidades contínuas da aplicação.	**Excessivo**: projeta uma solução de infraestrutura personalizada e dedicada, atrasando a implantação de uma aplicação.
Desenvolvimento	**Colaborativo**: trabalha com uma combinação de pessoas, processos e ferramentas, resultando em uma colaboração próxima entre as funções de desenvolvimento e operações.	**Silos**: trabalha com transferência excessiva de código finalizado, dos desenvolvedores para as operações.
Entrega de serviços	**Continua**: disponibilizam atualizações de *software* individuais e lançam assim que estão prontos.	**Períodos**: lançam softwares periodicamente.
Dependência entre serviços	**Independente**: decompõe os aplicativos em serviços operacionais independentes pequenos e fracamente acoplados.	**Dependente**: agrupam muitos serviços diferentes em um único pacote de implantação, causando dependências desnecessárias entre os serviços e levando a uma perda de agilidade durante o desenvolvimento e a implantação.
Escalabilidade	**Automática**: a automação da infraestrutura em escala elimina o tempo de inatividade devido a erro humano.	**Manual**: inclui operadores humanos que criam e gerenciam manualmente configurações de servidor, rede e armazenamento.
Recuperação a falhas	**Rápida**: a orquestração gerencia dinamicamente o posicionamento de *containers* em um *cluster* de VMs para fornecer dimensionamento elástico e reinicialização em caso de falha de uma aplicação ou de infraestrutura.	**Lenta**: VMs individuais são lentas para a inicialização e apresentam uma grande sobrecarga mesmo antes de implantar o código de uma aplicação nelas.

FONTE: PIVOTAL SOFTWARE[2]

3.2 Microsserviços

Quando pretende-se implantar uma aplicação na nuvem o ideal é que a aplicação seja projetada como uma aplicação nativa para nuvem, e o estilo arquitetural que tem sido usado com êxito é microsserviços. Assim, nesta seção, propõe-se uma definição para microsserviços no contexto de *multi-cloud* e descreve-se seus princípios e benefícios. Além disso, compara-se microsserviços com arquiteturas tradicionais. Por fim, apresenta-se os principais desafios na adoção de microsserviços.

Na literatura, os microsserviços possuem várias definições. De acordo com Newman (2015), microsserviços são pequenos serviços autônomos que trabalham juntos. Na segunda definição, dada por Pautasso *et al.* (2017), os microsserviços representam um estilo arquitetônico como uma abordagem para desenvolver uma única aplicação como um conjunto de pequenos serviços, cada um executando seu processo e se comunicando com mecanismos leves, geralmente uma *Application Programming Interface* (API) de recursos HTTP. Na definição fornecida por

RV (2016), microsserviços são um estilo arquitetural ou uma abordagem para a construção de sistemas de TI como um conjunto de recursos de construção autônomos e fracamente acoplados. Por fim, Dragoni *et al.* (2017) propuseram duas definições relacionadas a microsserviço: uma para microsserviço em si e outra para arquitetura de microsserviço. Para os autores, um microsserviço é um processo coeso e independente que interage por meio de mensagens, enquanto uma arquitetura de microsserviço é uma aplicação distribuída, em que todos os seus módulos são microsserviços.

Apesar destes conceitos, optou-se neste trabalho por uma definição própria, por entender que é necessário uma visão mais ampla que as definições encontradas até a escrita desta tese. Portanto, neste trabalho define-se **Microsserviços** como um conjunto de serviços autônomos, independentes e autocontidos, nos quais cada serviço tem um único objetivo, representando uma lógica de negócio, e eles devem ser fracamente acoplados e interagem para construir uma aplicação distribuída.

No contexto da definição proposta aqui, um microsserviço necessita de vários serviços de nuvem para executar suas tarefas. Além disso, como cada microsserviço representa uma lógica de negócios, entende-se que existe uma comunicação entre os serviços de nuvem utilizado por um microsserviço maior que a comunicação entre dois microsserviços, os quais tratam lógica de negócio distintas.

A fim de construir microsserviços, os princípios de sua arquitetura precisam ser conhecidos. De acordo com RV (2016), os princípios da arquitetura de microsserviços são autonomia e responsabilidade única. Os princípios devem estar presentes no projeto e desenvolvimento de microsserviços. O princípio responsabilidade única significa ter total responsabilidade por uma lógica de negócios e sua execução. A autonomia implica que os microsserviços sejam autônomos, independentemente implantáveis.

Além dos princípios faz-se necessário conhecer os benefícios dos microsserviços por trás dos sistemas distribuídos. De acordo com Newman (2015) e RV (2016) existem sete benefícios principais. O primeiro, **heterogeneidade da tecnologia**, permite decidir qual é a ferramenta mais adequada para cada serviço de uma aplicação composta por múltiplos serviços e com múltipla colaboração. Isto traz uma flexibilidade para projetar soluções com maior custo benefício. A **resiliência** é outro benefício, permite que um componente do sistema falhe, contanto que não esteja em cascata e permaneça isolado do resto do sistema, que pode continuar funcionando. O terceiro, a **escalabilidade**, permite expandir somente os serviços que

precisam de redimensionamento, sem alterar os demais serviços. A **facilidade de implantação** é outro benefício, no qual um microsserviço pode ser implantado independentemente dos outros microsserviços de uma aplicação. Isto permite atualizações e correções apenas de um serviço, bem como implantação rápida. Em relação ao **alinhamento organizacional**, os microsserviços permitem alinhar melhor a arquitetura à organização, ajudando a minimizar o número de pessoas trabalhando no código. Ele também permite trabalhar com equipes menores distribuídas em vários microsserviços. A **composibilidade** significa que a funcionalidade pode ser consumida de forma diferente para diferentes propósitos. Por fim, a **otimização por substituição** significa que é muito mais fácil gerenciar o custo de substituir serviços por uma implantação melhor ou até excluí-los completamente. Desta maneira, torna-se mais fácil atualizar e melhorar serviços em uma aplicação.

Para melhor apresentar o descrito nesta seção, e de acordo com RV (2016) a Figura 9 ilustra a arquitetura de uma aplicação baseada em microsserviços. Através da arquitetura é possível observar que cada microsserviço possui três camadas: apresentação, negócio e banco de dados. Portanto, como cada microsserviço representa uma lógica de negócios, os microsserviços estão alinhados às características de negócios e seus ciclos de vida, os quais podem ser gerenciáveis de maneira independente.

Fonte: RV (2016).

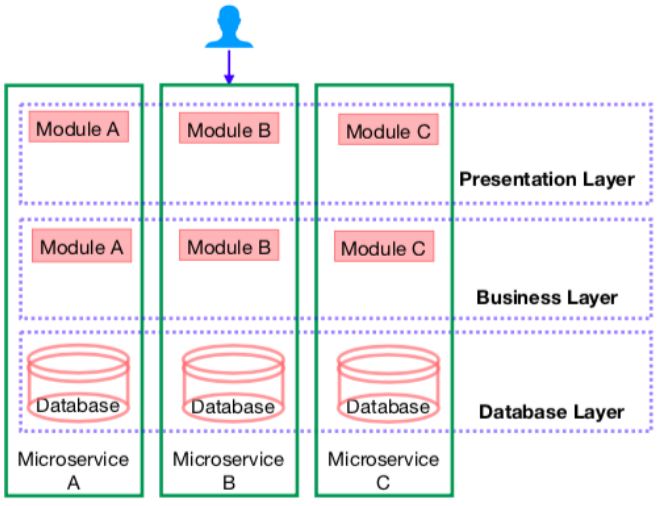


Figura 9. Arquitetura de uma Aplicação baseada em Microsserviços.

A Figura 10 apresenta a arquitetura tradicional de uma aplicação, e observa-se que, diferentemente da arquitetura baseada em microsserviços, todos os módulos representam todas as camadas de negócios, ou seja, cada camada de negócio possui todos os módulos.

Fonte: RV (2016).

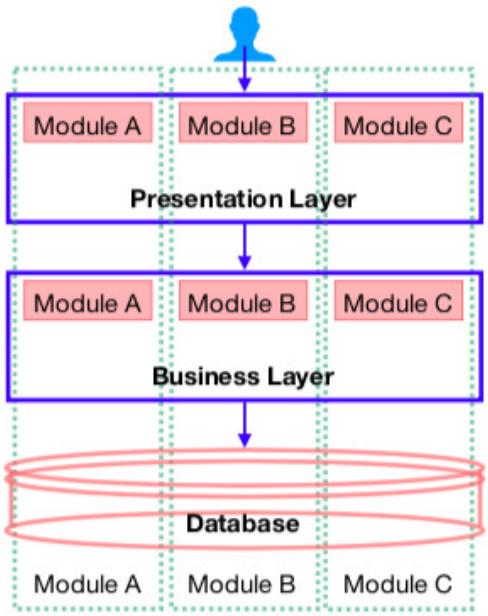


Figura 10. Arquitetura de uma Aplicação Tradicional.

Além de comparar as arquiteturas, neste presente trabalho, apresenta-se uma comparação entre o desenvolvimento monolítico e o baseado em microsserviço em conformidade com RV (2016), a qual é ilustrada pelas Figuras 11a e 11b. O gráfico da Figura 11a compara a agilidade, velocidade de entrega e escala. Pode-se observar que as aplicações baseadas em microsserviços possuem maior agilidade, velocidade de entrega e escala que as aplicações monolíticas. O gráfico da Figura 11b compara o tempo resposta e o custo. Nota-se que as aplicações baseadas em microsserviços têm menor tempo de resposta e custo do que as aplicações monolíticas. Diante deste contexto e das características necessárias para uma aplicação ser distribuída em múltiplos provedores de nuvem, e das características e benefícios constata-se que o estilo arquitetural de microsserviços são adequados para aplicações nativas para múltiplos provedores de nuvem.

Embora existam vários benefícios quanto ao uso de microsserviços, eles possuem alguns desafios. Os principais desafios relacionados à adoção de microsserviços na construção de um sistema são: (*i*) os arquitetos de *softwares* devem mudar a maneira de pensar o projeto dos sistemas e devem decidir sobre os limites certos para cada microsserviço que compõe o *software*; (*ii*) as complexidades dos sistemas distribuídos; (*iii*) a necessidade de escalar os sistemas de maneira diferente e garantir que os microsserviços sejam resilientes (NEWMAN, 2015) e (DRAGONI *et al.*, 2017).

(a) Agilidade, Velocidade de Entrega e Escalabilidade.

(b) Custo e Tempo de Resposta.

Figura 11. Uma comparação entre o desenvolvimento tradicional e o baseado em microsserviços.

3.3 Cenário Motivador

Esta seção descreve uma loja virtual baseada em microsserviços nomeada de AnaGomesModas. Além disso, esta seção também descreve as principais tomadas de decisões de um arquiteto de *software* em relação ao processo de implantação e monitoramento da aplicação em um ambiente *multi-cloud*.

3.3.1 Descrição

AnaGomesModas é uma loja virtual que vende moda praia, íntima e *fitness*. Cada segmento de moda possui diversos fornecedores distribuídos nos estados brasileiros. Ela também oferece serviços especializados aos seus clientes por meio de parcerias com nutricionistas, fisioterapeutas e *personal trainers*. A loja online AnaGomesModas permite o acesso aos seus clientes através de computadores pessoais e dispositivos móveis. Conforme ilustrado na Figura 12, a aplicação possui três tipos de usuários: Cliente, Profissional e AnaGomesModas.

O cliente usa a aplicação para comprar produtos, bem como para obter os serviços dos profissionais oferecidos pela loja. Os usuários do Profissional são profissionais que possuem parceria com a loja para oferecer serviços aos clientes da AnaGomesModas. O usuário AnaGomesModas é responsável por registrar produtos e fornecedores, bem como monitorar vendas, serviços e finanças. Assim, a loja de moda virtual AnaGomesModas oferece três tipos de serviços: serviços de venda de três segmentos de moda, prestação de serviços em parceria com três tipos de profissionais e serviço de monitoramento de venda e de pós-venda de produtos e/ou serviços.

Na Figura 12 pode-se notar que a aplicação AnaGomesModas possui quatro mi-

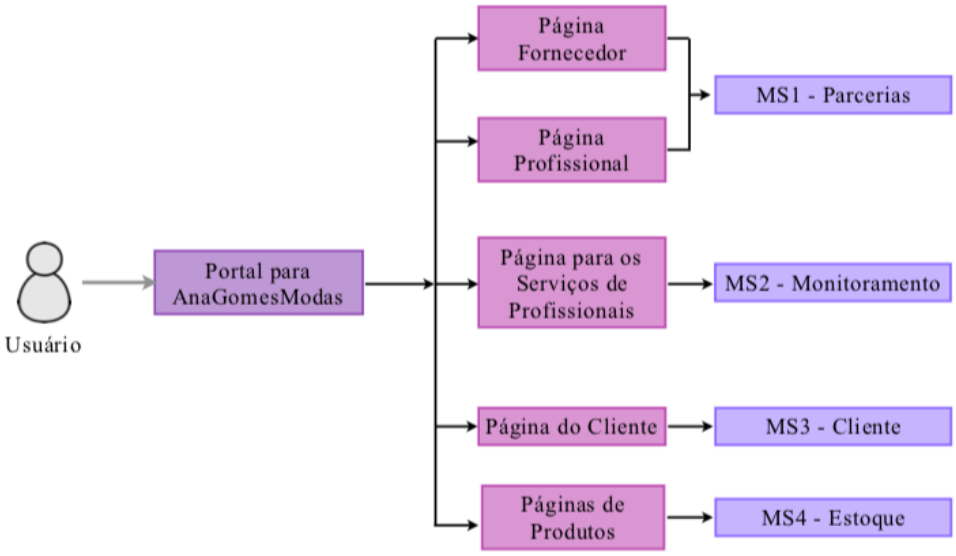


Figura 12. Os microsserviços do portal AnaGomesModas.

crosserviços. Cada microsserviço possui características e necessidades diferentes para serviços distintos. Assim, se um arquiteto de *software* desejar obter os melhores serviços de acordo com as necessidades da aplicação, e consequentemente, obter mais benefícios da computação nuvem, ele necessita usar um ambiente de múltiplos provedores de nuvem. Embora traga vários benefícios o uso de *multi-cloud*, um arquiteto de *software* precisa tomar várias decisões em relação a implantação e execução de uma aplicação distribuída em um ambiente *multi-cloud*. Este trabalho considera quatro tomadas de decisões essenciais como mostrado na Figura 13: (1) definição dos requisitos não-funcionais para cada microsserviço; (2) obtenção das capacidades dos provedores de nuvem; (3) seleção dos provedores para hospedar cada microsserviço; (4) implantação de cada microsserviços, observando os requisitos e características de cada provedor; (v) monitoramento dos dados referentes aos recursos utilizados pela aplicação, bem como das capacidades dos demais provedores de nuvem para migrar microsserviços caso obtenha melhor QoS.

Pode-se notar que um arquiteto *software* tem que tomar várias decisões, as quais são complexa, pois as aplicações necessitam de muitos serviços com requisitos diferentes e existem muitas ofertas de serviços por vários provedores de nuvem. Além disso, cada provedor de nuvem possui sua especificidade no processo de implantação e monitoramento das informações. Assim, este trabalho propõe PacificClouds, uma abordagem para facilitar as tomadas de decisões de um arquiteto de *software* no gerenciamento da implantação e do monitoramento de uma aplicação baseada em microsserviços distribuída em um ambiente *multi-cloud*.

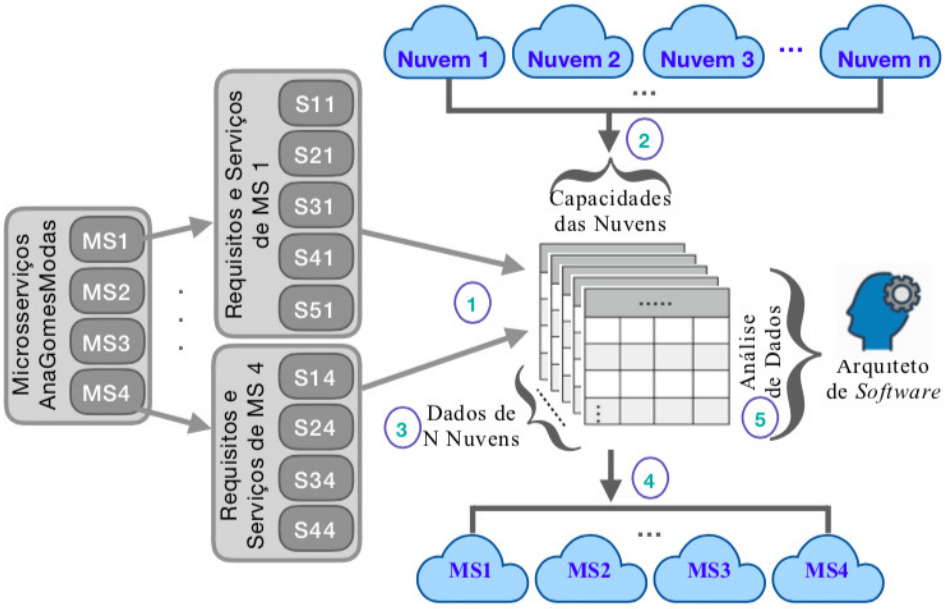


Figura 13. O gerenciamento de AnaGomesModas por um arquiteto de *software*.

3.4 PacificClouds

PacificClouds é uma nova abordagem com objetivo de gerenciar o processo de implantação e execução de uma aplicação distribuída em múltiplos provedores, mais especificamente para um ambiente *multi-cloud*, baseada em microsserviços da perspectiva de um arquiteto de *software*. Desta forma, PacificClouds busca facilitar as tomadas de decisões de um arquiteto de *software* e ajuda a mitigar o *vendor lock-in*. Esta seção apresenta e descreve a arquitetura do PacificClouds, bem como a relação da mesma com um arquiteto de *software* e com os provedores de nuvem. Depois, apresenta-se um cenário ao qual PacificClouds pode ser adequado, bem como descreve-se o processo de gerenciamento do PacificClouds para o cenário apresentado.

3.4.1 Arquitetura do PacificClouds

De acordo com os objetivos do PacificClouds mencionadas anteriormente, adota-se o modelo de entrega *multi-cloud* devido à sua maior flexibilidade, pois este ambiente não exige um acordo prévio entre os provedores de nuvem. Como PacificClouds pretende migrar microsserviços caso apareça um provedor de nuvem com melhor oferta, ou porque ocorreu violação dos requisitos, assim, ele precisa lidar com a portabilidade. Neste trabalho assume-se as três categorias de nível de portabilidade usadas por (PETCU *et al.*, 2013b): funcional, dados e aprimoramento de serviços nos níveis de IaaS e PaaS. A portabilidade de nível funcional é alcançada definindo as funcionalidades de uma aplicação independentemente do provedor. A

portabilidade no nível de dados é alcançada quando um usuário é capaz de recuperar dados de uma aplicação de um provedor e importá-los para uma aplicação equivalente hospedada em outro provedor. E o último nível, o aprimoramento de serviços, é alcançado através de metadados por meio de anotações e as APIs de controle que são permitidas pela infraestrutura serem adicionadas, reconfiguradas ou removidas em tempo real, seja automaticamente ou não, com base no tráfego, interrupções ou outros fatores.

A interoperabilidade é uma característica essencial para uma aplicação distribuída. Assim, este trabalho considera-se os seguintes níveis de interoperabilidade, os quais estão em conformidade com Nogueira *et al.* (2016): sintático e semântico, associados ao nível de concordância. A interoperabilidade sintática ocorre quando os sistemas podem se comunicar e trocar dados com um formato de dados específico e protocolos de comunicação e a semântica ocorre quando há uma interpretação automática significativa e precisa dos dados trocados para produzir resultados úteis. Em relação ao nível de adoção, assume-se os mecanismos de comunicação adotados para microsserviços. Relacionado ao nível de implantação, considera-se a interoperabilidade horizontal e vertical nos modelos de serviço IaaS e PaaS. A interoperabilidade horizontal ocorre quando os serviços trabalham juntos no mesmo nível de implantação enquanto que a vertical ocore quando os serviços trabalham juntos em diferentes níveis de implantação. A comunicação assíncrona é assumida nos padrões de interações entre as aplicações em nuvens. Finalmente, associada ao nível do usuário final, adota-se a interoperabilidade centrada no usuário, pois ajudam os usuários de diferentes serviços em nuvem a encontrar um denominador comum entre os serviços.

PacificClouds pretende abordar a interoperabilidade e a portabilidade em *multi-cloud* com foco na perspectiva de um arquiteto de *software* (usuário do PacificClouds) por meio de microsserviços e aplicações nativas para nuvem. Desta forma, também aborda uma questão importante relacionada à adoção da computação em nuvem, a flexibilidade. Assim, esta subseção apresenta e descreve a arquitetura PacificClouds.

Para que a arquitetura cubra todos os objetivos do PacificClouds mencionados nesta seção, deve-se incluir as seguintes funcionalidades: (i) identificar os microsserviços de uma aplicação, bem como os requisitos dos mesmos; (ii) identificar os requisitos de um arquiteto de *software*; (iii) descobrir e monitorar as capacidades dos provedores de nuvem disponíveis; (iv) gerenciar a implantação e execução de aplicações; (v) monitorar a implantação e execução de cada microsserviço; (vi) monitorar os requisitos de um arquiteto de *software* e de uma aplicação,

bem como os recursos dos provedores de nuvem envolvidos; (vii) prover a implantação de cada microsserviço de uma aplicação; (viii) gerenciar os recursos dos provedores de nuvem de cada microsserviço; (ix) fornecer informações de performance de uma aplicação e de violação de requisitos a um arquiteto de *software*, bem como informações das capacidades dos outros provedores disponíveis, para que um arquiteto de *software* possa tomar decisões sobre migrar um microsserviço ou de modificar algum requisito. Para dar suporte às funcionalidades previstas, a Figura 14 ilustra a arquitetura do PacificClouds, a qual consiste em três partes: PacificClouds API, Adapter, PacificClouds Core.

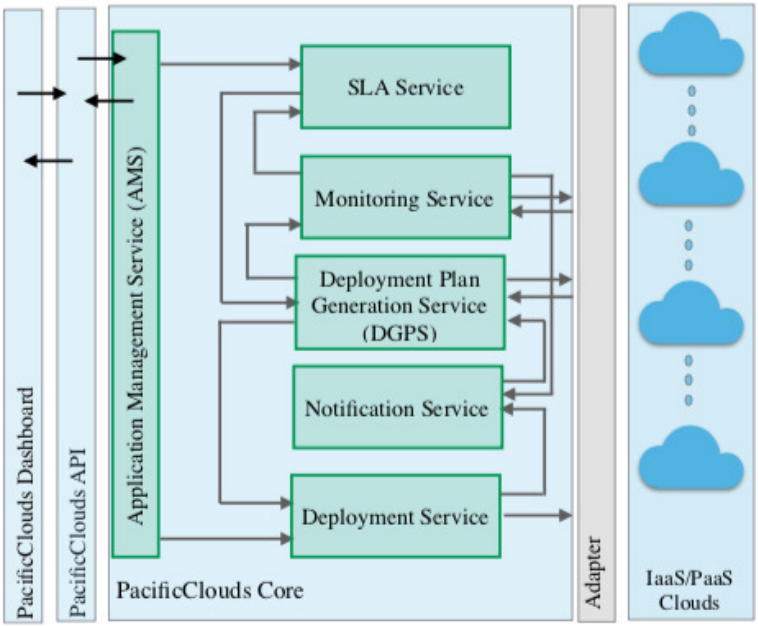


Figura 14. Arquitetura do PacificClouds.

PacificClouds API é a interface de comunicação entre um arquiteto de *software* e o PacificClouds Core. PacificClouds API recebe dois artefatos de um arquiteto de *software*: os microsserviços de uma aplicação e os requisitos dos microsserviços e de um arquiteto de *software*. Em seguida, PacificClouds API envia as informações recebidas ao PacificClouds Core. O PacificClouds Core fornece informações sobre os recursos dos provedores de nuvem usados por uma aplicação, bem como sobre os requisitos de uma aplicação e de um usuário, para um arquiteto de *software* via PacificClouds API.

Adapter é a interface de comunicação entre os provedores de nuvem disponíveis e o PacificClouds Core. O Adapter envia para o PacificClouds Core as características dos recursos dos provedores de nuvem disponíveis, bem como dos provedores de nuvem que estão hospedando uma aplicação. O Adapter envia para os provedores de nuvem selecionados os microsserviços de

uma aplicação.

PacificClouds Core é a parte central da arquitetura. Ela é responsável por identificar os microsserviços de uma aplicação e seus respectivos requisitos; determinar os provedores de nuvem para implantar cada microsserviço; implantar os microsserviços; gerenciar os recursos dos provedores de nuvem que hospedam os microsserviços de uma aplicação; monitorar informações sobre os recursos utilizados por cada microsserviço de uma aplicação em tempo de execução; monitorar todas as capacidades dos provedores de nuvem disponíveis; e verificar os requisitos de um arquiteto de *software*. Assim, o PacificClouds Core é composto por seis microsserviços para realizar cada uma destas tarefas. Esses microsserviços se comunicam por meio de chamadas síncronas e eventos assíncronos. As próximas subseções descrevem cada um destes microsserviços.

3.4.1.1 Application Management Service - AMS

O serviço de gerenciamento da aplicação recebe uma aplicação do PacificCloud API e é responsável por registrá-la no AppData, através de serviço de registro de aplicações (*Application Register Service* - ARS). Em seguida, ele envia a aplicação para o serviço de identificação de microsserviços (*Microservice Identification Service* - MIS), o qual extrai cada microsserviço da aplicação. Depois, ele extrai os requisitos de cada microsserviço e do arquiteto de *software*, através do serviço de descoberta de requisitos (*Requirements Discovery Service* - RDS). Além disso, o AMS registra o perfil de um arquiteto de *software* no USrPfData via o serviço de registro de usuário (*User Register Service*-URS), conforme Figura 15.

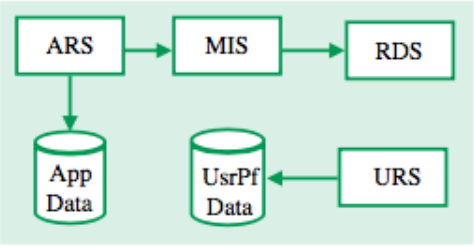


Figura 15. Microsserviço para o gerenciamento de uma aplicação - AMS.

3.4.1.2 Deployment Plan Generation Service

A Figura 16 ilustra o serviço de geração do plano de implantação (*Deployment Plan Generation Service* - DPGS), que determina os provedores de nuvem para implantar os

microsserviços sem violar os requisitos de uma aplicação e os requisitos de um arquiteto de *software* por meio do serviço de seleção de nuvens (*Cloud Selection Service* - CSS). Para isso, ele recebe do SLA Service todos os requisitos. Depois, ele deve gerar um plano de implantação de microsserviços com base nos requisitos e nos recursos das nuvens obtidos através do serviço de descoberta das capacidades dos provedores de nuvem (*Cloud Capabilities Discovery Service* - CCDS), bem como registra o plano de implantação no PlanData. O CCDS envia as capacidades dos provedores de nuvem para o serviço de mapeamento dos recursos dos provedores de nuvem (*Cloud Resources mapping Service* - (CRMS)), o qual é responsável por mapear todas as capacidades dos provedores e em seguida registra-os ou atualiza-os no ClData. Além disso, analisa as notificações recebidas do Notification Service e os requisitos recebidos do SLA Service. Em seguida, se necessário, gera um plano de reimplantação de microsserviços.

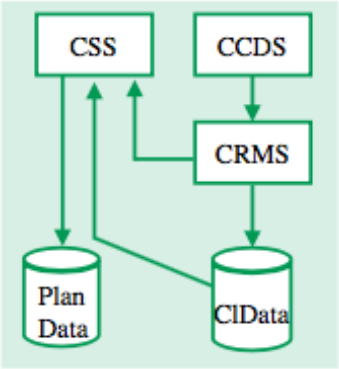


Figura 16. Microsserviço para geração do plano de implantação - DPGS.

3.4.1.3 Monitoring Service

O serviço de monitoramento (*Monitoring Service*), mostrado na Figura 17, é responsável por monitorar os recursos dos provedores de nuvem que hospedam os microsserviços de uma aplicação em tempo de execução através do Adapter. O Monitoring Service é composto por três serviços: (i) o serviço de registro de recursos de provedores de nuvem (*Cloud Resource Register Service* - CRRS), o qual recebe do DPGS os dados do plano de implantação. Depois, registra-os em MsRsData e o envia para o CRDAS; (ii) O serviço de detecção de recursos de provedores de nuvem (*Cloud Resource Detection Service* - CRDetS) obtém todas as capacidades dos provedores de nuvem, nos quais estão implantados os microsserviços. Em seguida, envia-os para o CRDAS; (iii) O serviço de análise de dados dos recursos dos provedores de nuvem (*Cloud Resource Data Analysis Service* - CRDAS) é responsável por comparar o desempenho de cada

microsserviço de uma aplicação recebido do CRDetS com os dados do plano de implantação. Em seguida, ele envia os dados dos recursos dos provedores de nuvem que hospedam um aplicação para o SLA Service e envia uma notificação para o Notification Service, caso ocorra alguma violação.

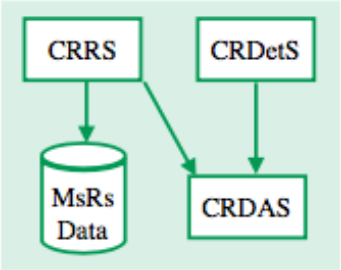


Figura 17. Microsserviço de monitoramento de Provedores de Nuvem - Monitoring Service.

3.4.1.4 SLA service

A Figura 18 ilustra o serviço de SLA (*SLA Service*), o qual é responsável por mapear os requisitos de um arquiteto de *software* e dos microsserviços recebidos do AMS, bem como os dados dos recursos dos provedores de nuvem usados pela aplicação, recebida do Monitoring Service, através do serviço de mapeamento de dados (*Data Mapping Service* - DMS) e em seguida, registra-los no UsrReqData, McsReqData e PerfrData, respectivamente. Além disso, o SLA Service fornece procedimentos e métodos para avaliar e relatar informações sobre a performance de uma aplicação, bem como sobre a violação de requisitos para um arquiteto de *software*, via serviço de análise de dados (*Data Analysis Service*- DAS).

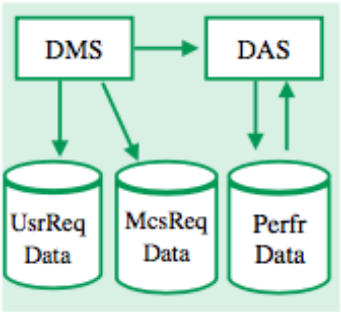


Figura 18. Microsserviço para mapeamento de requisitos - SLA Service.

3.4.1.5 Notification Service

O serviço de notificação (*Notification Service*), mostrado na Figura 19, é responsável por notificar o microsserviço DPGS através do serviço de notificação de implantação (*Deployment Notification Service* - DplNS) sobre mudanças nos requisitos de um arquiteto de *software* ou nos requisitos de um microsserviço, ou a atualização de uma aplicação; também informa a DPGS sobre a violação dos requisitos pelo provedor de nuvem, registra os dados de notificação no DplNData e no UNData, e notifica o arquiteto de *software* para implantar ou reimplantar uma aplicação via serviço de notificação do usuário (*User Notification Service* - UsrNS).

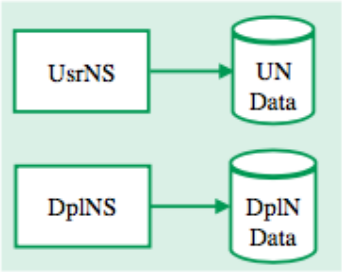


Figura 19. Microsserviço para notificação - Notification Service.

3.4.1.6 Deployment Service

O serviço de implantação (*Deployment Service*), ilustrado na Figura 20, recebe os microsserviços do AMS através do serviço de recepção de microsserviços de uma aplicação (*Application Microservice Reception Service* - AMRS), além de cadastrar cada microsserviço no McsData. Em seguida, o serviço de implantação identifica o provedor de nuvem para cada microsserviço de acordo com o plano de implantação por meio do serviço de associação de microsserviço (*Microservice Association Service* - MAS); depois, implanta os microsserviços por meio do serviço de implantação de microsserviço (*Microservice Deployment Service* - MDplS).

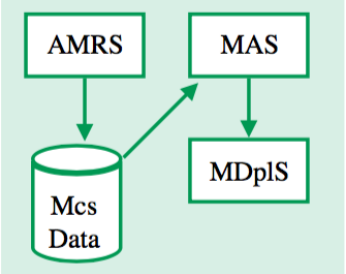


Figura 20. Microsserviço para implantação de uma aplicação - Deployment Service.

Observa-se que o PacificClouds pretende gerenciar o processo de implantação e execução de uma aplicação distribuída em múltiplos provedores de nuvem para melhor atender às necessidades de uma aplicação e, consequentemente, melhorar a satisfação do usuário final com o desempenho da aplicação. Assim, o PacificClouds será capaz de ajudar um arquiteto de *software* no projeto e desenvolvimento de uma aplicação, pois um arquiteto de *software* não deve se preocupar com o processo de implantação e execução da aplicação. Para isso, o PacificClouds deve identificar cada um dos serviços oferecidos por uma aplicação, bem como seus requisitos. Além disso, o PacificClouds deve selecionar os provedores de nuvem mais adequados para cada um dos microsserviços de uma aplicação e implantar cada um dos microsserviços de uma aplicação de acordo com o processo de seleção de provedores de nuvem. PacificClouds também deve monitorar a execução de aplicações observando os provedores de nuvem envolvidos na implantação da aplicação para monitorar e detectar violações de requisitos, atualizações dos microsserviços de uma aplicação e alterações nos recursos dos provedores para identificar a necessidade de reimplantar microsserviços.

3.5 Trabalhos relacionados ao PacificClouds

Esta seção descreve primeiro uma visão geral dos trabalhos mais relevantes relacionados ao PacificClouds em relação ao gerenciamento de um aplicação distribuída em múltiplos provedores de nuvem, comparando-os ao PacificClouds.

O Cloud4SOA (DANDRIA *et al.*, 2012) introduz uma arquitetura baseada em *broker* cujo objetivo principal é abordar desafios de interoperabilidade semântica na camada de PaaS, com base na arquitetura SOA. O mOSAIC (PETCU *et al.*, 2013b) visa oferecer acesso a recursos heterogêneos de múltiplos provedores de nuvem. A partir de uma perspectiva de alto nível, uma API e um PaaS compõem o mOSAIC, o PaaS permite implementar, configurar e gerenciar aplicações em execução em IaaS.

O MODAClouds (ARDAGNA *et al.*, 2012) oferece um conjunto de técnicas para desenvolvimento e gerenciamento de operações em tempo de execução de aplicações em múltiplos provedores de nuvem. Ele oferece um ambiente integrado de desenvolvimento de código aberto para o *design* de alto nível, seleção de serviços de nuvem, prototipagem inicial, avaliações de QoS, geração de código semiautomático e implantação automática de aplicações em múltiplos provedores de nuvem. O RASIC (LOUTAS *et al.*, 2010) se concentra na resolução dos problemas de interoperabilidade semântica em IaaS e na introdução de uma abordagem centrada no usuário

para aplicações que usam múltiplos recursos de nuvem. Isso facilita a alternância entre os provedores de nuvem e permite a integração de composição e serviços de diferentes nuvens.

O ASMEMA (SOUSA *et al.*, 2016) é um trabalho de configuração automatizada de ambientes *multi-cloud* para aplicações baseadas em microsserviços, o qual é uma abordagem automatizada para a seleção e configuração de provedores de nuvem para aplicações baseados em microsserviços. Mas os autores não tratam diretamente com a implantação de aplicações e não consideram custos e qualidade de serviço para otimizar a seleção. O SeaClouds (BROGI *et al.*, 2015) se concentra em implantar e gerenciar aplicações complexas de múltiplos componentes em nuvens heterogêneas. A abordagem é baseada no conceito de orquestração de serviço e é projetada para atender às propriedades funcionais e não funcionais de uma aplicação. Além disso, os serviços podem ser implantados, replicados e administrados usando APIs padronizadas, como especificação *Cloud Application Management for Platforms* (CAMP)(KARMARKAR, 2014) e Cloud4SOA.

A fim de comparar os trabalhos descritos nesta seção com PacificClouds, primeiro apresenta-se um resumo das características das obras relacionadas à promoção da interoperabilidade em um ambiente de múltiplas nuvens, incluindo o PacificClouds, apresentadas na Tabela 5. Dez aspectos foram considerados para ser observar o nível de flexibilidade de cada solução.

Na Tabela 5 pode-se perceber que a interoperabilidade semântica é o único aspecto que todos os trabalhos buscam prover. Alguns trabalhos usam a interoperabilidade semântica na portabilidade de aplicações, enquanto outros usam a interoperabilidade entre módulos de aplicações distribuídas em múltiplos provedores de nuvem e incluem soluções para tratar a interoperabilidade vertical entre modelos de serviço.

Uma das características descritas na Tabela 5 refere-se ao modelo de serviço usado pelas soluções. O MODAClouds e o ASMEMA tratam a interoperabilidade nos três modelos de serviços principais: IaaS, PaaS e SaaS. Apesar de abordar os três níveis, MODAClouds apenas aborda a interoperabilidade horizontal para *cloud federation*. O ASMEMA não aborda nenhum dos níveis de interoperabilidade vertical e horizontal para o modelo de entrega *multi-cloud*. Duas outras soluções, Cloud4SOA e RASIC, tratam da interoperabilidade em apenas um dos modelos de serviço. O Cloud4SOA aborda a interoperabilidade no nível de PaaS; ele não aborda nenhum nível de interoperabilidade vertical e horizontal e usa o modelo de entrega híbrida, que é uma *cloud federation* que tem uma nuvem privada e pelo menos uma nuvem pública (PETCU, 2014a). O RASIC trata a interoperabilidade no modelo de serviço IaaS para

TABELA 5. RESUMO DAS CARACTERÍSTICAS DOS TRABALHOS RELACIONADOS AO PACIFICCLOUDS.

Carcterísticas dos Projetos	Soluções para múltiplas nuvens						
	Cloud4 SOA	mOSAIC	MODAClouds	RASIC	ASMEMA	SeaClouds	**PacificClouds**
Modelo de Serviço	PaaS	IaaS/PaaS	IaaS/PaaS/SaaS	IaaS	IaaS/PaaS/SaaS	IaaS/PaaS	IaaS/PaaS/SaaS
Modelo de Entrega	Hybrid	Hybrid	Cloud Federation	Multi-Cloud	Multi-Cloud	Multi-Cloud	Multi-Cloud
Portabilidade	✔	✔	✔	✘	✘	✔	✔
Interoperabilidade Vertical	✘	✔	✘	✘	✘	✔	✔
Interoperabilidade Horizontal	✘	✘	✔	✔	✘	✔	✔
Distribuição da Aplicação	✘	✘	✘	✔	✔	✔	✔
Base Tecnológica Diferente	✘	✔	✔	✔	✔	✘	✔
Semânticas	✔	✔	✔	✔	✔	✔	✔
Flexibilidade	✘	✘	✘	✘	✔	✘	✔
Governância Descentralizada	✘	✘	✘	✘	✘	✘	✔

✔ O TRABALHO POSSUI A CARACTERÍSTICA ✘ O TRABALHO NÃO POSSUI A CARACTERÍSTICA

multi-cloud, mas atende apenas ao nível de interoperabilidade horizontal. Por fim, mOSAIC, SeaClouds e PacificClouds tratam a interoperabilidade nos modelos de serviço IaaS e PaaS. O mOSAIC usa o modelo de entrega *hybrid*, além de tratar a portabilidade da aplicação. O SeaClouds e o PacificClouds abordam os níveis de interoperabilidade vertical e horizontal para o modelo de entrega *multi-cloud*, além de promover a portabilidade de aplicações.

RASIC, ASMEMA, SeaClouds e PacificClouds abordam a interoperabilidade para aplicações distribuídas em múltiplos provedores de nuvem geograficamente dispersos, enquanto as outras soluções abordam apenas a portabilidade de aplicações no modelo de entrega *inter-cloud*. Em relação às bases tecnológicas para os provedores de nuvem, apenas Cloud4SOA e SeaClouds não usam provedores de nuvem com diferentes bases tecnologias, ou seja, todos os provedores de nuvem envolvidos na portabilidade ou interoperabilidade de aplicações devem ter a mesma base tecnológica.

As soluções ASMEMA e PacificClouds são flexíveis porque permitem implantar cada módulo de uma aplicação em um provedor de nuvem diferente e permitem que os módulo usem tecnologias diferentes entre eles. Além disso, elas permitem que um módulo seja atualizado independentemente de outros módulos. Eles possuem algumas diferenças, dentre elas, o módulo do ASMEMA é um serviço, enquanto que o módulo do PacificClouds é um microsserviço. Desta forma, o processo de seleção do PacificClouds é mais complexo, mas gera menos tráfego na rede. Outra diferença entre ele é que o o ASMEMA não faz governança descentralizada por não tratar interoperabilidade ou portabilidade. Portanto, o PacificClouds é a única abordagem que pretende promover a governança descentralizada de aplicações até a escrita deste documento, permitindo que os módulos de aplicações tenham um acoplamento flexível, além de gerar menos tráfego na rede e, portanto, permitir maior flexibilidade.

De acordo com a análise realizada nesta subseção, pode-se observar que o PacificClouds propõe maior flexibilidade em relação à interoperabilidade em um ambiente *multi-cloud*. Assim, PacificClouds pretende aumentar a flexibilidade através da distribuição de uma aplicação nativa para nuvem baseada em microsserviços em um ambiente *multi-cloud* de acordo com os requisitos da aplicação e de um arquiteto de *software*, independentemente da tecnologia usada.

3.6 Resumo do Capítulo

Conforme mencionado neste capítulo, a aplicação nativa para nuvem é uma abordagem que permite a implantação e execução de uma aplicação em diferentes provedores de nuvem.

Além disso, de acordo com Sethi (2017), microsserviços não apenas ajudam a gerenciar cada módulo eficientemente, mas também funcionam de maneira semelhante, na qual atuam como um *link* entre serviços diferentes e manipulam o fluxo de dados para uma transação específica com base no tipo de solicitações. Portanto, os conceitos e aspectos relacionados a aplicação nativa para nuvem e microsserviços apresentados aqui são importantes para desenvolver aplicações para múltiplas nuvens.

Este capítulo apresentou PacificClouds (CARVALHO *et al.*, 2018b), o qual é uma abordagem baseada em microsserviços que foca na perspectiva de um arquiteto de *software* e os ajuda a tomar decisões sobre o desenvolvimento de aplicações a serem implantadas em um ambiente *multi-cloud*. Portanto, um arquiteto de *software* não precisa se preocupar com a seleção de provedores de nuvem, com a implantação de uma aplicação e com o monitoramento de uma aplicação em tempo de execução. A arquitetura do PacificClouds foi também descrita, a qual oferece maior flexibilidade, governança descentralizada, além do seu objetivo principal.

4 MODELOS PARA SELEÇÃO DE MÚLTIPLAS NUVENS

A proliferação de provedores de nuvem fez com que cada provedor desejasse oferecer infraestrutura, plataforma e serviços para satisfazer as necessidades de seus usuários, de tal maneira que eles não desejassem usar outros provedores. Consequentemente, a infraestrutura e os serviços de um provedor diferem uns dos outros, e os provedores se tornaram consideravelmente específicos. Assim, torna-se difícil uma tomada de decisão sobre a seleção de um provedor de nuvem para um arquiteto de *software* que precisa implantar aplicações que exigem vários serviços de nuvem, e esses serviços são diferentes entre si e possuem diferentes requisitos. Como essas aplicações são mais complexas, o uso de múltiplas nuvens permite selecionar os provedores de nuvem que podem melhor atender a uma aplicação e a um arquiteto de *software*.

Como mencionado no capítulo 3, para facilitar a tomada de decisão dos arquitetos de *softwares*, Carvalho *et al.* (2018b) propuseram PacificClouds, que é uma nova abordagem para gerenciar o processo de implantação e execução de uma aplicação baseada em microsserviços em ambientes *multi-cloud*, a partir da perspectiva de um arquiteto de *software*. PacificClouds usa microsserviços como o estilo arquitetural porque eles fornecem suporte para criar aplicações nas quais cada parte é escalável e implantada independentemente. Além disso, o ambiente *multi-cloud* permite que um arquiteto de *software* distribua uma aplicação para aproveitar as vantagens oferecidas pela computação em nuvem e também ajuda a mitigar o *vendor lock-in*.

Embora um ambiente de múltiplas nuvens possibilite aproveitar ao máximo a computação em nuvem, pois traz uma maior flexibilidade, para fazer uso destas faz-se necessário selecionar provedores de nuvem que possam melhor atender a uma aplicação e aos requisitos de um arquiteto de *software* (usuário). Assim, a escolha de provedores de nuvem para implantar uma aplicação distribuída é uma tarefa complexa, porque existem múltiplos provedores. Para que os arquitetos de *software* possam escolher os requisitos e a arquitetura de uma aplicação sem precisar se preocupar com a implantação de uma aplicação em múltiplas nuvens, PacificClouds precisa selecionar os provedores de nuvem para hospedar os microsserviços de uma aplicação. O processo de seleção verifica entre os provedores de nuvem disponíveis os que melhor atendem às restrições de um arquiteto de *software* e de cada microsserviço de uma aplicação, observando que cada microsserviço necessita de vários serviços de nuvem e cada um deles possui diferentes funcionalidades e restrições. PacificClouds deve implantar um microsserviço de uma aplicação no provedor de nuvem que melhor atenda às suas necessidades. Assim, é possível perceber que uma aplicação possui vários microsserviços e que um microsserviço precisa usar vários serviços

de nuvem. Além disso, existem muitos provedores disponíveis e cada um deles oferece vários serviços com as mesmas funcionalidades, mas com diferentes capacidades.

Portanto, uma seleção de provedores de nuvem é realizado observando um conjunto de serviços de nuvem candidatos de cada provedor para otimizar os requisitos dos microsserviços de uma aplicação e de uma arquiteto de *software*. Este trabalho considera que um serviço candidato é um serviço de nuvem que atende a todos os requisitos de um microsserviço. Além disso, um microsserviço necessita de `r` serviços de nuvem. O processo de seleção de múltiplos provedores de nuvem nesta tese pode ser representado em quatro etapas como mostra a Figura 21. Na primeira etapa, o PacificClouds deve receber todos os requisitos de um arquiteto de *software*. Na segunda, PacificClouds deve obter as capacidades dos provedores de nuvem disponíveis. Na terceira, PacificClouds deve selecionar todos os provedores de nuvem que atendam aos requisitos de um arquiteto de *software* de cada microsserviço de uma aplicação. Finalmente, na quarta etapa, ele deve selecionar um provedor de nuvem para cada microsserviço dentre os provedores selecionados no terceiro passo. Este trabalho assumi que os requisitos de uma aplicação e as capacidades dos provedores já estão disponíveis.

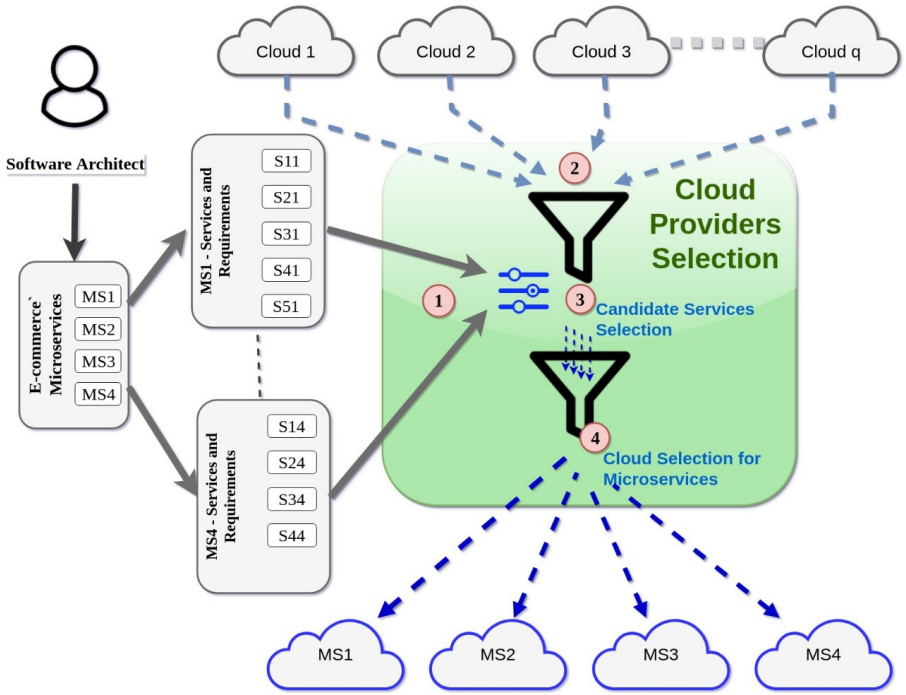


FIGURA 21. PROCESSO DE SELEÇÃO *Multi-Cloud* PARA O PACIFICCLOUDS.

De acordo com o descrito acima, cada microsserviço de uma aplicação é implantado em um único provedor de nuvem. De maneira que o provedor selecionado atenda pelo menos aos requisitos solicitados, sendo que o provedor selecionado deve ser o que melhor atenda aos

requisitos exigidos. Portanto, o ambiente de múltiplas nuvens traz um importante ganho, pois permite que cada microsserviço encontre a melhor oferta de acordo com seus requisitos.

Portanto, neste capítulo, três modelos são descritos para selecionar múltiplas nuvens, que são baseados nos requisitos de um arquiteto de *software* e dos microsserviços de uma aplicação, considerando as capacidade dos provedores de nuvem disponíveis, as quais deverão ser usadas pelo PacificClouds para implantar microsserviços de uma aplicação. Para os três modelos, apresenta-se uma descrição formal do processo de seleção *multi-cloud*, no qual as duas primeiras abordagens possuem o mesmo modelo de serviços. O primeiro modelo, descrito na Seção 4.1, é chamado de modelo para microsserviços desvinculados mapeado para mochila (*Model Unlinked Microsservices Mapped to the Knapsack - UM^2K*). Neste modelo, os microsserviços de uma aplicação devem ser desvinculados um do outro, ou seja, este modelo seleciona múltiplos provedores para aplicações onde a comunicação entre os microsserviços podem ser desprezada, e o processo é mapeado para o problema da mochila de múltiplas escolhas. O segundo modelo é chamado de modelo para microsserviços desvinculados mapeado para cota (*Model Unlinked Microsservices Mapped to Quota - UM^2Q*), o qual é descrito na Seção 4.2. No UM^2Q, como no UM^2K os microsserviços de uma aplicação são desvinculados e o processo de seleção é baseado em cota do orçamento para cada microsserviço de uma aplicação. Na na Seção 4.2, o terceiro modelo é descrito, o qual é chamado de modelo para microsserviços vinculados mapeado para mochila (*Model for Linked Microservices Mapped to Knapsack - LM^2K*). Neste modelo a comunicação entre os microsserviços de uma aplicação é tratada, e o processo é mapeado para o problema da mochila de múltiplas escolhas como em UM^2K.

Os três modelos apresentam contribuições significativas em comum:

1. O processo de seleção de nuvens para implantar microsserviços de uma aplicação é inovador, porque estas propostas selecionam múltiplos provedores de nuvem para hospedar os microsserviços de uma aplicação, sendo um para cada microsserviço. Ele é baseado nos requisitos de um arquiteto de *software* e de cada microsserviço.
2. Uma descrição formal para o processo de seleção de nuvens de cada modelo, que descreve como as aplicações e os provedores são modelados e como selecionar cada serviço de nuvem e cada provedor para hospedar cada microsserviço.
3. A seleção de nuvens usa *Simple Additive Weighting* (SAW) para classificar os serviços de nuvem em cada provedor disponível e, consequentemente, provedores de nuvem candidatos (AL-FAIFI *et al.*, 2017), nos quais os requisitos de um arquiteto de *software*

são considerados, e cada um deles têm peso e prioridade.

Além dessas contribuições, dois modelos tratam de aplicações onde os microsserviços são independentes. O primeiro e o terceiro modelos mapeiam o processo de seleção para o problema da mochila de múltiplas escolhas (*Multi-Choice Knapsack Problem* (MCKP)). O segundo trata individualmente os requisitos em cada microsserviço. E o terceiro trata a comunicação entre os microsserviços de uma aplicação.

4.1 Modelo para Microsserviços Desvinculados Mapeado para Mochila - UM^2K

Nesta seção, um modelo é descrito para seleção de múltiplos provedores para hospedar uma aplicação baseada em microsserviços, nomeado de modelo para microsserviços desvinculados mapeado para mochila (*Model for Unlinked Microservices Mapped to the Knapsack - UM^2K*) (CARVALHO *et al.*, 2018) e (CARVALHO *et al.*, 2019). Este modelo trata de aplicações baseadas em microsserviços independentes, onde a comunicação entre os microsserviços pode ser desconsiderada. Ele também mapeia o processo de seleção para o problema da mochila de múltiplas escolhas. Primeiro, descreve-se a formalização do modelo de serviço. Depois, o processo de seleção de provedores de nuvem.

4.1.1 Formalização do Modelo de Serviço

A existência de vários serviços de nuvem com as mesmas funcionalidades, mas com diferentes capacidades pode ser notado. Além disso, pode-se observar provedores de nuvem que oferecem vários serviços com características semelhantes, mas que possuem diferentes capacidades. Assim, os arquitetos de *softwares* podem solicitar serviços com várias capacidades. Embora este trabalho trate o tempo de resposta de uma aplicação (tempo de execução + atraso), a disponibilidade dos serviços de nuvem e custo de execução de uma aplicação, outros requisitos podem ser incluídos neste modelo.

De acordo com o processo de seleção de provedores descrito, no preâmbulo deste capítulo, para atender as necessidades do PacificClouds para facilitar as tomadas de decisões de um arquiteto de *software* propõe-se:

- Definição 1: **modelo de serviço de nuvem** - como S(S.rt, S.a, S.c), em que S.rt, S.a e S.c representam tempo de resposta, disponibilidade e custo, respectivamente, como em (LIU *et al.*, 2011).

- Definição 2: **classe de serviços de nuvem** - como SC_k ={S_{k1}, S_{k2}, ..., S_{ko}} em que S_{k1}, S_{k2}, ..., S_{ko} são serviços da classe `k` de um mesmo provedor com as mesmas funcionalidades, mas com diferentes capacidades.
- Definição 3: **modelo de provedor de serviços** - como SP_j = {SC_{j1}, SC_{j2}, ..., SC_{jp}}, em que SC_{j1},SC_{j2}, ..., SC_{jp} são classes de serviços para o provedor de nuvem `j`.
- Definição 4: **conjunto de provedores de nuvem** - como CP = {SP_1, SP_2,..., SP_q}, em que SP_1, SP_2,..., SP_q são provedores de nuvem disponíveis.
- Definição 5: **modelo de microsserviço** - como MS_i = {$S_{i1}^{k_1}$, $S_{i2}^{k_2}$, ..., $S_{ir}^{k_r}$}, em que $S_{i1}^{k_1}$, $S_{i2}^{k_2}$, ..., $S_{ir}^{k_r}$ são serviços de nuvem indispensáveis para execução das funções do microsserviço `i`. Cada serviço necessário para composição do microsserviço `i` pode pertencer a diferentes classes de serviços de um provedor de nuvem. Desta forma, $1 \leqslant k_r \leqslant o$, onde `o` é o máximo de classes de um provedor de nuvem.
- Definição 6: **modelo de aplicação** - como AP = {MS_1, MS_2, ..., MS_t}, que MS_1,MS_2, ..., MS_t são microsserviços de uma mesma aplicação.

Todas as definições descritas para o processo de seleção de provedores de nuvem devem seguir o modelo de serviço de nuvem (definição 1). Exemplo: A definição X é modelada como X (X.rt, X.a, X.c). Um arquiteto de *software* que deseja implantar uma aplicação em múltiplas nuvem via PacificClouds deve especificar os limiares de requisitos para tempo de resposta, disponibilidade e custo. Além disso, ele deve definir a prioridade para cada um dos requisitos. O processo de seleção de provedores de nuvem tem como objetivo selecionar os provedores que melhor atendem aos requisitos de um arquiteto de *software* e otimizem os objetivos especificados por um arquiteto de *software*, da seguinte maneira:

4.1.1.1 Requisito de disponibilidade

A disponibilidade de um provedor de nuvem para uma aplicação deve atender pelo menos ao limiar definido por um arquiteto de *software*, e cada microsserviço de uma aplicação deve atender ao mesmo limiar. Assim, um arquiteto de *software* define MinAvbty como o limiar de disponibilidade mínima. Na Eq. 4.1, AP.a é definida como a disponibilidade de uma aplicação, que recebe o menor valor de disponibilidade entre seus microsserviços e deve ser maior ou igual a MinAvbty. Além disso, na Eq. 4.2, $MS_i.a$ representa a disponibilidade do microsserviço `i`, que recebe o menor valor de disponibilidade entre seus serviços e deve ser maior ou igual a MinAvbty. Na Eq. 4.2, $S_{ij}^{k_j}.a$ representa a disponibilidade de um provedor para um serviço, o

qual é o serviço `j` do microsserviço `i` de uma classe k_j. Uma aplicação possui no máximo `t` microsserviços e um microsserviço necessita no máximo de `r` serviços de nuvem.

$$AP.a = \min_{1 \leqslant i \leqslant t} (MS_i.a) \geqslant MinAvbty, \forall MS_i \mid MS_i \in AP \tag{4.1}$$

$$MS_i.a = \min_{1 \leqslant j \leqslant r} S_{ij}^{k_j}.a \geqslant MinAvbty, \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, MS_i \in AP, 1 \leqslant i \leqslant t \tag{4.2}$$

4.1.1.2 Requisito de Tempo de Resposta

O tempo de resposta de uma aplicação deve atender ao limiar definido por um arquiteto de *software*, e cada microsserviço de uma aplicação deve atender ao mesmo limiar. Um arquiteto de *software* define MaxRT como o limiar máximo de tempo de resposta. MaxRT é definido como o limiar máximo de tempo de execução (MaxExecTime) mais o limiar máximo de atraso (MaxDelay) como na Eq.4.3. Na Eq. 4.4, AP.rt é definido como o tempo de resposta de uma aplicação, que recebe o maior valor de tempo de resposta entre seus microsserviços e deve ser menor ou igual a MaxRT. Além disso, na Eq. 4.5, $MS_i.rt$ representa o tempo de resposta do microsserviço `i`, que é atribuído ao maior valor de tempo de resposta entre seus serviços e deve ser menor ou igual a MaxRT. O tempo de resposta para um serviço é representado por $S_{ij}^{k_j}.rt$ na Eq. 4.5, que é o serviço `j` do microsserviço `i` de uma classe k_j. Uma aplicação possui no máximo `t` microsserviços e um microsserviço necessita no máximo de `r` serviços de nuvem.

$$MaxRT = MaxExecTime + MaxDelay \tag{4.3}$$

$$AP.rt = \max_{1 \leqslant i \leqslant t} (MS_i.rt) \leqslant MaxRT, \forall MS_i \mid MS_i \in AP \tag{4.4}$$

$$MS_i.rt = \max_{1 \leqslant j \leqslant r} (S_{ij}^{k_j}.rt) \leqslant MaxRT, \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, MS_i \in AP, 1 \leqslant i \leqslant t \tag{4.5}$$

4.1.1.3 Requisito de Custo

O custo da execução de uma aplicação não deve ser superior ao Budget, que é o limiar de custo definido por um arquiteto de *software*. Na Eq. 4.6, AP.c é definido como o custo

de execução de uma aplicação, que é atribuído ao custo total de todos os seus microsserviços e deve ser menor ou igual ao Budget. Na Eq.4.7, $MS_i.c$ representa o custo do microsserviço `i`, ao qual é atribuída a soma dos custos dos serviços de nuvem necessários para a sua execução, e deve ser menor igual ao Budget. O custo para um serviço é representado por $S_{ij}^{k_j}.c$ na Eq. 4.7, que é o serviço `j` do microsserviço `i` de uma classe k_j. Uma aplicação tem `t` como o máximo de microsserviços, e um microsserviço necessita no máximo de `r` serviços de nuvem.

$$AP.c = \sum_{i=1}^{t} MS_i.c \leqslant Budget, \forall MS_i \mid MS_i \in AP \tag{4.6}$$

$$MS_i.c = \sum_{j=1}^{r} S_{ij}^{k_j}.c \leqslant Budget, \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, MS_i \in AP \tag{4.7}$$

4.1.2 Seleção de Provedores de Nuvem

Nesta subseção, o processo de seleção de provedores de nuvem é detalhado para o modelo UM^2K, e tem como base o modelo de serviço descrito acima. O processo proposto tem três níveis: o primeiro nível descobre os serviços de nuvem candidatos para um microsserviço de uma aplicação em todos os provedores disponíveis; o segundo determina os provedores candidatos para cada microsserviço; e o terceiro seleciona os provedores para hospedar todos os microsserviços de uma aplicação, mapeando o processo de seleção para o problema da mochila de múltiplas escolhas.

4.1.2.1 Primeiro Nível - Descoberta dos Serviços de Nuvem Candidatos

A descoberta dos serviços candidatos a compor um microsserviço de uma aplicação é feita observando os requisitos de um arquiteto de *software* em cada serviço de nuvem. Para isso, primeiro identifica-se os serviços de nuvem que atendem às necessidades de um microsserviço em todos os provedores de nuvem disponíveis. Depois, classifica-se os serviços de nuvem que atendem aos requisitos de um arquiteto de *software* com base no SAW (ZENG *et al.*, 2004), o qual possui duas fases: escala e ponderação.

- **Fase de escala** - Nesta fase, os requisitos de um arquiteto de *software* são numerados de 1 a 3, com 1 = disponibilidade, 2 = tempo de resposta, 3 = custo. Uma matriz $R = (R_{ij}; 1 \leqslant i \leqslant n; 1 \leqslant j \leqslant 3)$ é construída mesclando os vetores de requisito de todos os `n` serviços candidatos de um provedor de nuvem. Serviços candidatos referem-se aos serviços necessários para a execução de um microsserviço. Todo o processo deve ser

realizado para todos os tipos de serviços que um microsserviço necessita. Cada linha R_i corresponde a um serviço de nuvem $S_{ij}^{k_j}$ e cada coluna R_j corresponde a um requisito, no qual $R_j^{Max} = Max(R_j), R_j^{Min} = Min(R_j), 1 \leqslant i \leqslant n$ e R_{ij} é o requisito `j` para o serviço `i` e seu valor está entre R_j^{Max} e R_j^{Min}. Para isso, existem dois tipos de critérios:

- **Negativo**: quanto maior o valor do requisito, menor a qualidade do serviço.

$$V_{ij} = \begin{cases} \frac{R_j^{Max} - R_{ij}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \tag{4.8}$$

- **Positivo**: quanto maior o valor do requisito, maior a qualidade do serviço.

$$V_{ij} = \begin{cases} \frac{R_{ij} - R_j^{Min}}{R_j^{Max} - R_j^{Min}} & \text{if } R_j^{Max} - R_j^{Min} \neq 0 \\ 1 & \text{if } R_j^{Max} - R_j^{Min} = 0 \end{cases} \tag{4.9}$$

- **Fase de Ponderação** - O *score* geral dos requisitos é calculado para cada serviço de nuvem candidato, usando a equação 4.10. Na Eq. 4.10 w_j é o peso do requisito `j`, e a soma dos pesos de todos os requisitos deve ser igual a 1.

$$Score(S_i) = \sum_{j=1}^{3} (V_{ij} * W_j) \mid W_j \in [0,1], \sum_{j=1}^{3} W_j = 1 \tag{4.10}$$

Ao final do primeiro nível, todos serviços candidatos para um microsserviço em todos provedores de nuvem disponíveis devem ter sido identificados e classificados.

4.1.2.2 Segundo Nível - Determinação dos Provedores Candidatos

Todas as combinações de serviços de um provedor devem ser descobertos para compor um microsserviço. Para isso, os serviços candidatos para um microsserviço que são oferecidos pelo mesmo provedor devem ser feitas para identificar todas combinações. A Eq. 4.11 define SSP_{ik} como o conjunto de combinações candidatas do provedor `k` para o microsserviço `i`. Além disso, na Eq. 4.11, $comb_{ikj}$ indica a combinação `j` do provedor `k` para o microsserviço `i`. Uma combinação $comb_{ikj}$ é um conjunto de serviços de nuvem de classes diferentes para atender às necessidades do microsserviço `i` como mostrado na Eq. 4.12. Em que $S_{il_n}^{m_n}$ indica o serviço candidato l_n da classe m_n necessário para função `n` do microsserviço `i`. O microsserviço possui no máximo `r` funções e, consequentemente, necessita de no máximo `r` serviços de nuvem,

um provedor candidato tem no máximo `m` combinações e um microsserviço tem no máximo `q` provedores candidatos.

$$SSP_{ik} = \{comb_{ik1}, ..., comb_{ikm}\} \tag{4.11}$$

$$comb_{ikj} = \{S_{il_1}^{m_1}, ..., S_{il_r}^{m_r}\} \tag{4.12}$$

O *score* e o custo devem ser calculados para cada combinação em SSP_{ik} como mostrado pelas Eqs. 4.13 e 4.14. Na Eq. 4.13, $comb_{ikj}.score$ é definido como os *score* médios de cada serviço de uma combinação. Na Eq. 4.14, $comb_{ikj}.c$ é definido como o custo total de todos os serviços combinados.

$$comb_{ikj}.score = \frac{Score(S_{il_1}^{m_1}) + Score(S_{il_2}^{m_2}) + ... + Score(S_{il_r}^{m_r})}{r} \tag{4.13}$$

$$comb_{ikj}.c = S_{il_1}^{m_1}.c + S_{il_2}^{m_2}.c + ... + S_{il_r}^{m_r}.c \tag{4.14}$$

Ao final do segundo nível, tem-se um conjunto de todas as combinações candidatas em todos os provedores disponíveis para todos os microsserviços de uma aplicação (SAMS), e cada um delas tem seu *score*. A Eq. 4.15 define SMS_i o qual é um conjunto de provedores candidatos para o microsserviço `i`, em que SSP_{ik} indica o conjunto de combinações candidatas do provedor `k` para o microsserviço `i`. Na Eq. 4.16, SAMS é definido como o conjunto de provedores candidatos para cada microsserviço de uma aplicação, e uma aplicação tem no máximo `t` microsserviços.

$$SMS_i = \{SSP_{ik} \mid k \in [1, q]\} \tag{4.15}$$

$$SAMS = \{SMS_1, ..., SMS_t\} \tag{4.16}$$

4.1.2.3 Terceiro Nível - Seleção de Provedores para hospedar Microsserviços

Provedores de nuvem pertencente ao SAMS (resultado do segundo nível) devem ser selecionados para hospedar cada microsserviço de uma aplicação. Para isso, combinações candidatas de SAMS devem ser geradas, de forma que cada combinação tenha uma combinação candidata para cada microsserviço, e o conjunto dessas combinações é chamado de SAPP (4.17). Além disso, a Eq. 4.17, $comb_{ik_i j_{k_i}}$ indica a combinação `j` do provedor candidato `k` para o microsserviço `i`. Posteriormente, na Eq. 4.18, $SAPP_l.c$ é definido como o custo de execução de cada combinação em SAPP e então, ele é verificado se é menor ou igual ao custo de execução da aplicação, e excluído de outra forma. Em seguida, calcula-se o *score* de cada item em SAPP. Um *score* de uma combinação candidata ($SAPP_l.score$) é um valor entre $(0,1]$ e é a soma dos *scores* de cada item da combinação, que é mostrada na Eq. 4.19.

$$\begin{aligned} SAPP = \{(comb_{1k_1 j_{k_1}}, ..., comb_{tk_t j_{k_t}}) \mid comb_{ik_i jk_i} \in SSP_{ik}, SSP_{ik} \in SMS_i, \\ SMS_i \in SAMS, 1 \leqslant i \leqslant t, 1 \leqslant k_i \leqslant q_i, 1 \leqslant j_{k_i} \leqslant w_{k_i}\} \end{aligned} \tag{4.17}$$

$$SAPP_l.c = \sum_{i=1}^{t} comb_{ik_i j_{k_i}}.c \tag{4.18}$$

$$SAPP_l.score = \sum_{i=1}^{t} comb_{ik_i j_{k_i}}.score \tag{4.19}$$

O processo de seleção descrito para o modelo UM^2K é um problema de otimização combinatória, pois várias combinações devem ser realizadas para obter os provedores para hospedar os microsserviços de uma aplicação. Para isso, primeiro obtém-se as combinações candidatas para cada microsserviço em todos os provedores de nuvem disponíveis. Depois, as combinações para a aplicação, onde cada combinação contém um conjunto de combinações candidatas, às quais podem pertencer a provedores de nuvem distintos, sendo uma para cada microsserviço de uma aplicação. Ao final, uma das combinações candidatas é selecionada para uma aplicação, a qual contém um provedor para cada microsserviço.

Um dos problemas mais estudados na otimização combinatória é o problema da mochila (PISINGER, 1995). Dentre as variações do problema da mochila existe o problema da mochila de múltiplas escolhas (MCKP) (KELLERER *et al.*, 2004). O MCKP é uma variação do

problema da mochila, no qual o conjunto de itens é dividido em classes, e o objetivo é colocar na mochila um item de cada classe, de forma que não exceda a capacidade da mochila, e que maximize o lucro, ou seja, obter um item de cada classe que não ultrapasse o peso da mochila e que produza o maior lucro.

O processo de seleção é mapeado para o problema da mochila de múltipla escolha. De forma que o orçamento de uma aplicação representa o peso máximo da mochila e o custo de cada item em PP representa o peso do item. Cada item do AP representa o conjunto de serviços necessários para um microsserviço de uma aplicação. Assim, os microsserviços representam as classes, em que cada classe possui um conjunto de combinações candidatas para atender a um microsserviço. Diante deste contexto, o problema de seleção múltiplos provedores de nuvem deve maximizar os requisitos de um arquiteto de *software* através do *score* de cada item em SAPP. O *score* do item representa o lucro, a Eq. 4.20 ilustra a função objetivo para maximizar o *score*, que está sujeita as Eqs. 4.21, 4.22 e 4.23 por obedecer ao custo total da combinação. Nessas equações, $comb_{ikj}$ é a combinação candidata `j` do provedor `k` para o microsserviço `i`, SMS_i é um conjunto de provedores candidatos para o microsserviço `i`, conforme definido na Eq. 4.15 e SSP_{ik} é um conjunto de combinações do provedor `k` para o microsserviço `i`, conforme definido na Eq. 4.11.

$$max(\sum_{i=1}^{t} \sum_{k \in SMS_i} \sum_{j \in SSP_{ik}} comb_{ikj}.score * Comb_{ikj}) \tag{4.20}$$

$$\sum_{i=1}^{t} \sum_{k \in SMS_i} \sum_{j \in SSP_{ik}} comb_{ikj}.c * comb_{ikj} \leqslant AP.c \tag{4.21}$$

$$\sum_{k \in SMS_i} SSP_{ik} = 1, \forall 1 \leqslant i \leqslant t \tag{4.22}$$

$$\sum_{j \in SSP_{ik}} comb_{ikj} = 1, \forall 1 \leqslant i \leqslant t, k \in [1,q] \tag{4.23}$$

4.2 Modelo para Microsserviços Desvinculados Mapeado para Cota - UM^2Q

Nesta seção, o segundo modelo é proposto para selecionar múltiplos provedores de nuvem para hospedar os microsserviços de uma aplicação da perspectiva de um arquiteto de

software, que deve ser usado pelo PacificClouds. O modelo de seleção proposto trata de aplicações que usam microsserviços independentes, ou seja, a comunicação entre os microsserviços de uma aplicação pode ser desconsiderada. Além disso, o modelo trata individualmente cada requisito, de forma que não possuem requisitos gerais. Isto significa que cada microsserviço possui seus requisitos independentemente dos outros microsserviços da mesma aplicação. Para isso, o orçamento de uma aplicação é estipulado pelo arquiteto de *software* através de cota por microsserviço. Portanto, esse modelo é chamado de modelo para microsserviços desvinculados mapeado para cota (*Model for Unlinked Microservices Mapped to a Quota* - UM^2Q). Embora esse modelo possa atender aos requisitos necessários, este trabalho considera três requisitos: tempo de resposta, disponibilidade e custo de execução de uma aplicação.

4.2.1 Formalização do Modelo de Serviço

A formalização do modelo de serviço para o modelo UM^2Q é semelhante à formalização descrita na Seção 4.1.1. Eles diferem porque nesse modelo UM^2Q seleciona um provedor para cada serviço com base em requisitos de microsserviço individuais. Para isso, o arquiteto de *software* precisa definir um limiar de orçamento para cada microsserviço, nomeado de cota de orçamento. Os limiares para disponibilidade e tempo de resposta seguem as mesmas regras definidas nas subseções 4.1.1.1 e 4.1.1.2, respectivamente, para o modelo UM^2K.

4.2.1.1 Requisito de Custo

O custo de execução de uma aplicação não deve ser superior ao Budget, que é o limiar de custo definido por um arquiteto de *software*. Na Eq. 4.24, AP.c é definido como o custo de execução de uma aplicação, ao qual é atribuído o custo total de todos os seus microsserviços e deve ser menor ou igual ao Budget. Na Eq.4.25, $MS_i.c$ representa o custo do microsserviço `i`, ao qual é atribuído a soma dos custos dos serviços de nuvem necessários para a sua execução, e deve ser menor ou igual à cota do Budget. O custo para um serviço é representado por $S_{ij}^{k_j}.c$ na Eq. 4.25, que é o serviço `j` do microsserviço `i` de uma classe k_j. Uma aplicação tem `t` como o máximo de microsserviços, e um microsserviço necessita no máximo de `r` serviços de nuvem.

$$AP.c = \sum_{i=1}^{t} MS_i.c \leqslant Budget, \forall MS_i \mid MS_i \in AP \tag{4.24}$$

$$MS_i.c = \sum_{j=1}^{r} S_{ij}^{k_j}.c \leqslant Budget * Quota_i, \forall S_{ij}^{k_j} \mid S_{ij}^{k_j} \in MS_i, MS_i \in AP, Quota \in]0,1] \quad (4.25)$$

4.2.2 Seleção de Provedores de Nuvem

Nesse modelo, múltiplos provedores de nuvem são selecionados para hospedar cada microsserviço de uma aplicação e cada microsserviço é hospedado em um único provedor, levando em consideração apenas requisitos individuais por microsserviço definidos por um arquiteto de *software*. Este modelo, UM^2Q, apresenta três níveis de seleção: o primeiro nível determina os serviços candidatos de todos os provedores de nuvem disponíveis para todos os microsserviços de uma aplicação. O segundo nível determina os provedores candidatos para cada microsserviço. O terceiro seleciona dentre os provedores candidatos os que irão hospedar os microsserviços de uma aplicação.

4.2.2.1 Primeiro Nível- Determinação dos Serviços Candidatos

Serviços que atendem a todos os requisitos de um arquiteto de *software* são selecionados em todos os provedores disponíveis, o que resulta em um conjunto de serviços candidatos em cada provedor. Em seguida, classifica-se todos os serviços candidatos em cada provedor. Para isso, usa-se a técnica SAW como em Seghir e Khababa (2016) e como descrito na Subseção 4.1.2.1 para o modelo UM^2K, o qual, ao final, tem-se um conjunto de todos os serviços candidatos para cada microsserviço em todos os provedores disponíveis.

4.2.2.2 Segundo Nível - Determinação dos Provedores Candidatos

Todos os serviços necessários para compor cada microsserviço devem ser selecionados em cada provedor, nos quais os serviços candidatos foram selecionados no primeiro nível. Para isso, uma cota de orçamento para cada microsserviço de uma aplicação deve ter sido definida. Assim, na Eq. 4.26, $MS_i.c$ é definido como o custo de execução do microsserviço `i`, que deve ser menor ou igual a $(Budget * Quota_i)$. $Quota_i$ é definida como a cota de orçamento de execução de uma aplicação definida para microsserviço `i` e Budget como o limiar de custo de execução de uma aplicação. Em seguida, na Eq. 4.27, cada $Quota_i$ deve estar entre 0 e 1 e a soma do produto entre Budget e Quota $(Budget * Quota_i)$ deve ser igual ao orçamento. Além disso, na Eq. 4.28, a soma dos custos de todos os serviços de microsserviço `i` deve ser menor ou

igual ao custo de execução do microsserviço `i`.

$$MS_i.c \leqslant Budget * Quota_i, \forall MS_i \mid MS_i \in AP \tag{4.26}$$

$$Quota_i \in]0,1], \sum_{i=1}^{t} Budget * Quota_i = Budget \tag{4.27}$$

$$\sum_{j=1}^{r} S_{ij}.c \leqslant Budget * Quota_i, \forall S_{ij} \mid S_{ij} \in MS_i, 1 \leqslant i \leqslant t \tag{4.28}$$

Um microsserviço necessita de vários tipos de serviços de nuvem, e para cada um deles existem vários candidatos em um mesmo provedor. Assim, para compor um microsserviço, é necessário um serviço candidato de cada tipo solicitado pelo microsserviço. Assim, os serviços candidatos devem ser combinados, de forma que se tenha um de cada tipo solicitado pelo microsserviço e deve ser verificado o custo de execução da combinação de serviços. Para isso, primeiro, combina-se os serviços candidatos de um microsserviço que são oferecidos por um mesmo provedor, que é representado por $comb_{ikj}$ na Eq. 4.29, a qual indica a combinação `j` do provedor `k` para o microsserviço `i`. Em seguida, o custo de execução da combinação é calculado, e então, é verificado se ele é menor ou igual a $(Budget * Quota_i)$ como mostrado na Eq. 4.30, que está de acordo com as Eqs. 4.26, 4.27 e 4.28. Cada elemento da combinação $S_{ij_x}^{m_x}$ representa o serviço candidato j_x da classe m_x para o microsserviço `i`. E o custo de cada elemento da combinação é representado por $S_{ij_x}^{m_x}.c$. Cada microsserviço tem no máximo `r` serviços e cada serviço tem no máximo de n_x serviços candidatos por provedor. Cada provedor possui no máximo `o` classes e o conjunto de provedores CP tem no máximo `q` provedores.

$$comb_{ikj} = (S_{ij_1}^{m_1}, ..., S_{ij_r}^{m_r}) \mid comb_{ikj}.c \leqslant Budget * Quota_i, 1 \leqslant j \leqslant o * r, 1 \leqslant m \leqslant q \tag{4.29}$$

$$comb_{jl}.c = (S_{ij_1}^{m_1}.c + ... + S_{ij_r}^{m_r}.c) \mid 1 \leqslant j \leqslant o * r, 1 \leqslant m \leqslant q \tag{4.30}$$

Todo processo é repetido em todos os provedores disponíveis para todos os microsserviços de uma aplicação. Desta forma, tem-se um conjunto de combinações candidatas para cada um dos microsserviços de uma aplicação.

Em seguida, deve-se escolher uma combinação dentre as combinações candidatas em cada provedor para cada microsserviço. Para isso, calcula-se o *score* médio de cada combinação de serviços, representada por $comb_{kj}.avSc$ na Eq. 4.31. $comb_{kj}$ é a combinação `j` do provedor `k` que pertence a um conjunto de provedores (CP), e CP possui no máximo `q` provedores na Eq. 4.31. Além disso, $Score(S_{ij_1}^{m_1}) + Score(S_{ij_2}^{m_2}) + ... + Score(S_{ij_r}^{m_r})$ representa a soma de todos *scores* de serviço de uma combinação, S_{ij_x} é um serviço candidato de uma combinação, e `r` é um máximo de serviços de microsserviço. A combinação com a maior média de *score* deve ser selecionada.

$$comb_{kj}.avSc = \frac{Score(S_{ij_1}^{m_1}) + Score(S_{ij_2}^{m_2}) + ... + Score(S_{ij_r}^{m_1})}{r} \tag{4.31}$$

Se várias combinações tiverem a maior média de *score*, uma delas deve ser selecionada com base em um dos três requisitos, de acordo com as prioridades definidas por um arquiteto de *software*.

Ao final do segundo nível, deve haver uma combinação candidata em cada provedor candidato, (CSP_i) representa o conjunto de provedores candidatos para o microsserviço `i` de uma aplicação, como mostrado na Eq. 4.32. SP_{ki} representa o provedor candidato `k` para o microsserviço `i`, e ele tem uma combinação de serviço candidata selecionada para o microsserviço `i`. Uma aplicação possui no máximo t microsserviços, e um microsserviço tem no máximo q_i provedores candidatos .

$$CSP_i = \{SP_{1i}, SP_{2i}, \ldots, SP_{q_i i}\} \mid 1 \leqslant i \leqslant t, t = sizeof(AP), q_i = sizeof(CSP_i) \tag{4.32}$$

4.2.2.3 Terceiro Nível - Determinação dos Provedores para os Microsserviços

Um provedor para cada microsserviço deve ser selecionado. Primeiro, verifica-se o custo de execução de um microsserviço em cada provedor candidato, que foi calculado no segundo nível, como mostrado na Eq. 4.33. Nesta equação, $SP_{ki}.c$ indica o custo de execução do microsserviço `i` para o provedor candidato `k`. $SP_{ki}.c$ é a soma dos custos dos serviços que compõem a combinação candidata do provedor `k`. Na Eq. 4.33, $S_{ikj}^{m_j}.c$ é o custo do serviço `j` no

provedor candidato k para o microsserviço i.

$$SP_{ki}.c = \sum_{j=1}^{r} (S_{ikj}^{m_j}.c) \mid S_{ikj}^{m_j} \in SP_{ki}, SP_{ki} \in CSP_i, 1 \leqslant k \leqslant q_i, 1 \leqslant i \leqslant t \tag{4.33}$$

Em seguida, verifica-se o CSP do segundo nível para cada microsserviço, a fim obter o SP que apresenta o custo médio de execução do microsserviço e usa-se Eqs. 4.34, 4.35 e 4.36. As Eqs. 4.34 e 4.35 definem $Max(MS_i.c)$ e $Min(MS_i.c)$, os quais retornam o maior e o menor custo de execução de um microsserviço entre os provedores candidatos para o microsserviço i, respectivamente. Além disso, na Eq. 4.36, $Average(MS_i.c)$ retorna o custo médio de execução de microsserviço entre os provedores candidatos para microsserviço i.

$$Max(MS_i.c) = \max_{1 \leqslant k \leqslant s_i} (SP_{ki}.c) \mid \forall SP_{ki} \in CSP_i, 1 \leqslant i \leqslant t, s_i = sizeof(CSP_i) \tag{4.34}$$

$$Min(MS_i.c) = \min_{1 \leqslant k \leqslant s_i} (SP_{ki}.c) \mid \forall SP_{ki} \in CSP_i 1 \leqslant i \leqslant t, s_i = sizeof(CSP_i) \tag{4.35}$$

$$Average(MS_i.c) = \frac{Max(MS_i.c) + Min(MS_i.c)}{2} \tag{4.36}$$

Caso haja mais de um provedor com o mesmo custo médio de execução, o provedor que apresenta a maior disponibilidade ou tempo de resposta deve ser selecionado, observando a prioridade definida por um arquiteto de *software*. Para isso, usa-se as Eqs. 4.37 e 4.38 para calcular a disponibilidade e o tempo de resposta de cada provedor candidato, respectivamente. A equação 4.37 define $SP_{ki}.a$ como a disponibilidade da nuvem do provedor k para o microsserviço i, que recebe a menor disponibilidade de serviço candidato para o microsserviço i. A equação 4.38 define $SP_{ki}.rt$ como o tempo de resposta do provedor k para o microsserviço i, que recebe o maior tempo de resposta de serviço candidato do microsserviço i. Em seguida, usa-se Eqs. 4.39 e 4.40 para calcular o valor mais alto para disponibilidade e o valor mais baixo para o tempo de resposta em cada provedor candidato com base nas Eqs. 4.37 e 4.38, respectivamente.

$$SP_{ki}.a = \min_{1 \leqslant j \leqslant r} (S_{ikj}^{m_j}.a) \mid S_{ikj}^{m_j} \in SP_{ki}, SP_{ki} \in CSP_i, 1 \leqslant k \leqslant s_i, 1 \leqslant i \leqslant t \tag{4.37}$$

$$SP_{ki}.rt = \max_{1 \leqslant j \leqslant r} (S_{ikj}^{m_j}.rt) \mid S_{ikj}^{m_j} \in SP_{ki}, SP_{ki} \in CSP_i, 1 \leqslant k \leqslant s_i, 1 \leqslant i \leqslant t \tag{4.38}$$

$$Max(MS_i.a) = \max_{1 \leqslant k \leqslant s_i} (SP_k.a) \mid \forall SP_k \in CSP_i \tag{4.39}$$

$$Max(MS_i.rt) = \min_{1 \leqslant k \leqslant s_i} (SP_k.rt) \mid \forall SP_k \in CSP_i \tag{4.40}$$

4.3 Modelo para Microsserviços Vinculados Mapeado para Mochila - LM^2K

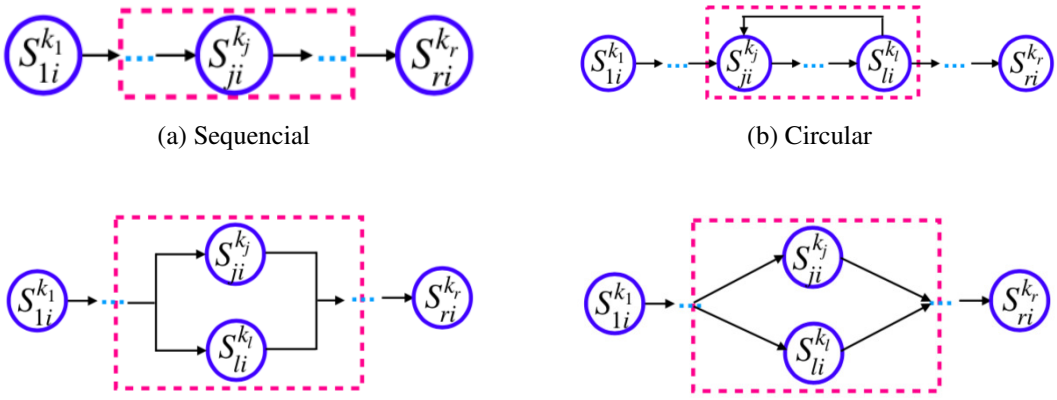
Nesta seção, o terceiro modelo é descrito para seleção de múltiplos provedores para hospedar uma aplicação baseada em microsserviços nomeado de modelo para microsserviços vinculados mapeado para mochila (*Model for Linked Microservices Mapped to the Knapsack* - LM^2K). Este modelo trata de aplicações baseadas em microsserviços vinculados, onde a comunicação entre os microsserviços é importante. Ele também mapeia o processo de seleção para o problema da mochila de múltiplas escolhas. Primeiro, descreve-se a formalização do modelo de serviço. Depois, o processo de seleção de provedores de nuvem.

4.3.1 Formalização do Modelo de Serviço

Como descrito na Subseção 4.1.1, um arquiteto de *software* pode solicitar diferentes serviços de nuvem com diferentes capacidades. Além disso, existem múltiplos provedores de nuvem, e cada um deles oferece diferentes tipos de serviços com diferentes recursos. Para cada serviço de nuvem, neste modelo, como nos modelos UM^2k e UM^2Q, considera-se o tempo de resposta de uma aplicação (tempo de execução + atraso), a disponibilidade dos serviços de nuvem e custo de execução de uma aplicação, mas outros requisitos podem ser incluídos com poucas alterações na implementação.

De acordo com o modelo de seleção de múltiplos provedores, descrito no início deste capítulo, seis definições são descritas para modelar os serviços referente aos provedores de nuvem e a uma aplicação baseada em microsserviços. Este modelo trata de aplicações às quais os microsserviços possuem vínculos uns com os outros, ou seja, existe comunicação considerável entre os microsserviços. Estas definições estão em consonância com as definições feitas para os modelos UM^2k e UM^2Q.

- Definição 1: **modelo de serviço de nuvem** - como S(S.rt, S.a, S.c), em que S.rt, S.a e S.c representam tempo de resposta, disponibilidade e custo, respectivamente.
- Definição 2: **classe de serviços de nuvem** - como SC_k ={S_{k1}, S_{k2}, ..., S_{ko}} em que S_{k1}, S_{k2}, ..., S_{ko} são serviços da classe `k` de um mesmo provedor com as mesmas funcionalidades, mas com diferentes capacidades.
- Definição 3: **modelo de provedor de serviços** - como $SP_j = \{SC_{j1}, SC_{j2}, \ldots, SC_{jp}\}$, em que $SC_{j1}, SC_{j2}, \ldots, SC_{jp}$ são classes de serviços para o provedor de nuvem `j`.
- Definição 4: **conjunto de provedores de nuvem** - como CP = {SP_1, SP_2,..., SP_q}, em que SP_1, SP_2,..., SP_q são provedores de nuvem disponíveis.
- Definição 5: **modelo de microsserviço** - como $MS_i = \{S_{i1}^{k1}, S_{i2}^{k2}, \ldots, S_{ir}^{kr}\}$, em que S_{i1}^{k1}, S_{i2}^{k2}, ..., S_{ir}^{kr} são serviços de nuvem indispensáveis para execução das tarefas do microsserviço `i`. Os serviços necessários para composição de um microsserviço pertence a diferentes classes de serviços de um provedor de nuvem. Desta forma, $1 \leqslant k_r \leqslant o$, onde o é o máximo de classes de um provedor de nuvem. Além disso, um microsserviço é composto por um fluxo de tarefas, as quais devem ser executadas por serviços de nuvem. A Figura 22 mostra quatro estruturas básicas para compor uma tarefa: sequencial, circular, paralela e de seleção, a qual está de acordo com Zhou *et al.* (2018).



(a) Sequencial (b) Circular

(c) Paralela (d) Seleção

FIGURA 22. ESTRUTURAS DE UMA TAREFA DE UM MICROSSERVIÇO.

Esta tese considera o fluxo de tarefas para um microsserviço como mostrado na Figura 23, para atender aos requisitos de um microsserviço. Um fluxo de tarefas é um grafo acíclico, no qual cada vértice representa um serviço e cada seta representa uma relação de precedência entre os serviços, o qual está de acordo com as estruturas básicas mostradas na

Figura 22. Todos os serviços de nuvem utilizados por um microsserviço devem pertencer ao mesmo provedor.

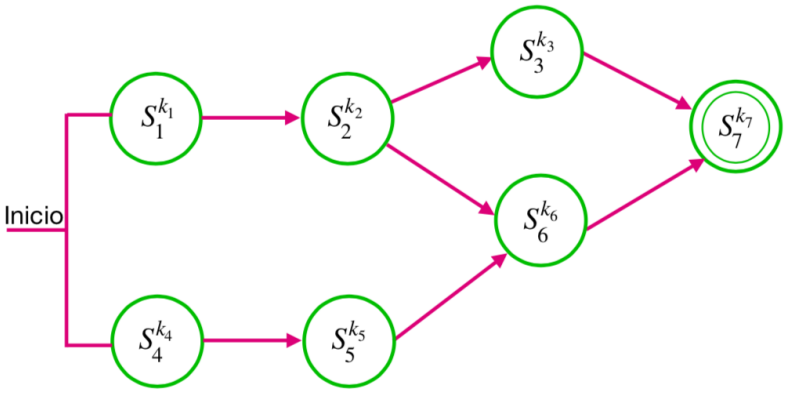
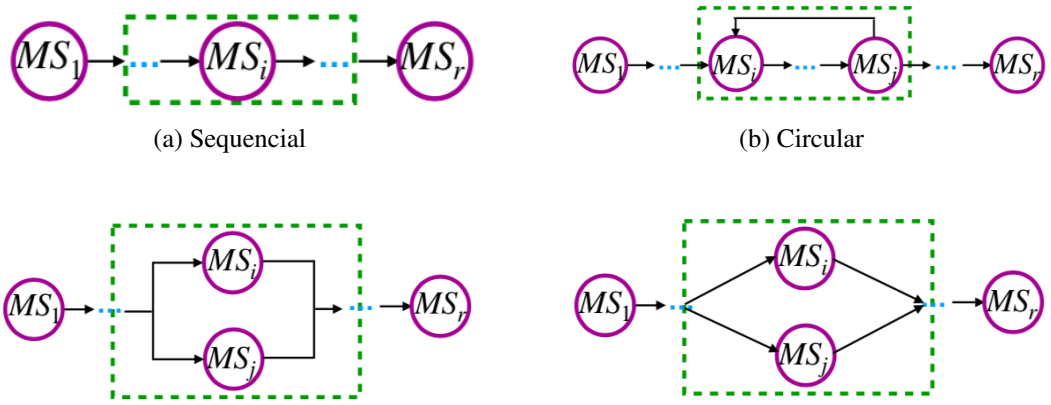


FIGURA 23. FLUXO DE TAREFAS DE NUVEM PARA UM MICROSSERVIÇO.

- Definição 6: **modelo de aplicação** - como AP = $\{MS_1, MS_2, \ldots, MS_t\}$, que $MS_1, MS_2, \ldots, MS_t$ são microsserviços de uma mesma aplicação. Para a composição de uma aplicação, os microsserviços devem usar as estruturas básicas mostradas na Figura 24.



(a) Sequencial (b) Circular

(c) Paralelo (d) Seleção

FIGURA 24. ESTRUTURAS DE COMPOSIÇÃO DE MICROSSERVIÇOS PARA UMA APLICAÇÃO.

O fluxo de microsserviços de uma aplicação considerado neste trabalho é mostrado no exemplo da Figura 25, se difere do fluxo de execução de um microsserviço porque os microsserviços de uma aplicação devem ser hospedados em provedores de nuvem diferentes, enquanto que os serviços de nuvem necessários para um microsserviço devem pertencer ao mesmo provedor.

Neste modelo, como nos modelos UM^2k e UM^2Q, um arquiteto de *software* para implantar uma aplicação em múltiplas nuvem através do PacificClouds deve especificar os

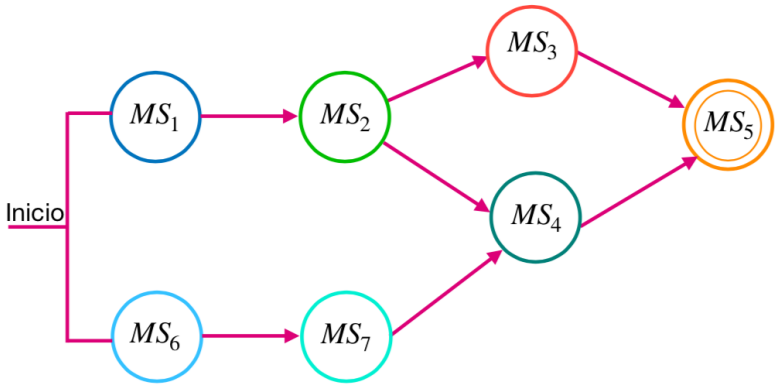


FIGURA 25. FLUXO DE MICROSSERVIÇOS DE UMA APLICAÇÃO CONSIDERADO PELO PACIFICCLOUDS.

limiares de requisitos para tempo de resposta, disponibilidade e custo. Além disso, ele deve definir a prioridade para cada um dos requisitos. O processo de seleção de provedores de nuvem tem como objetivo selecionar os provedores que melhor atendem aos requisitos de um arquiteto de *software* e otimizar os objetivos especificados por ele.

Para calcular os requisitos de um microsserviço, o fluxo de microsserviços de uma aplicação é dividido em dois níveis, o primeiro é chamado de termo; e o segundo, de sequência. As Figuras 26 e 27 mostram os termos e as sequências do exemplo de fluxo de microsserviços da Figura 25. Os termos 1 e 2, da Figura 26, são os fluxos de microsserviços da Figura 25 que devem ser realizados em paralelo. As sequências de um termo, mostradas na Figura 27, são os fluxos de microsserviços de um termo, onde apenas uma delas é ativada por cada vez que o fluxo de microsserviços for executado.

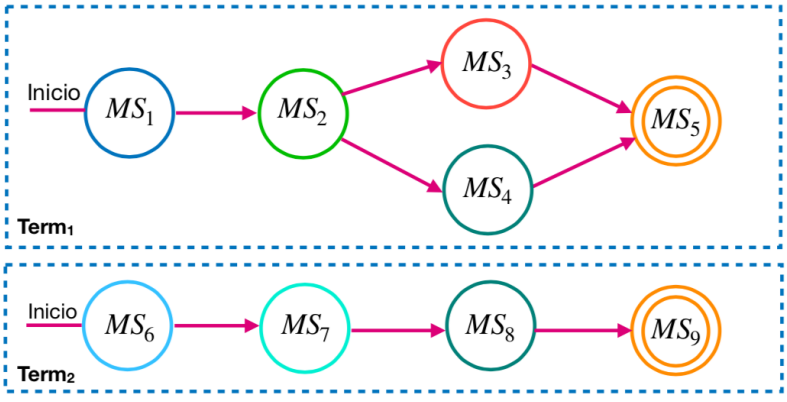


FIGURA 26. TERMOS DO FLUXO DE MICROSSERVIÇOS DE UMA APLICAÇÃO.

As próximas subseções descrevem as fórmulas para calcular os requisitos de uma aplicação. Para isso, considera-se o fluxo de microsserviços de uma aplicação, como mostra o exemplo das Figuras 26 e 27 e a comunicação entre os microsserviços, nomeados de *links* de

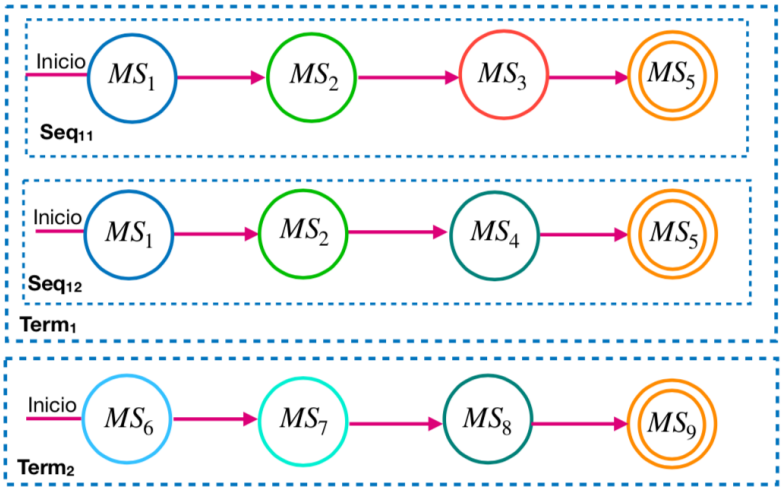


FIGURA 27. SEQUÊNCIAS DOS TERMOS DO FLUXO DE MICROSSERVIÇOS DE UMA APLICAÇÃO.

comunicação. As fórmulas para os requisitos estão em consonância com Hongzhen *et al.* (2016), Zeng *et al.* (2004), Alrifai e Risse (2009), Canfora *et al.* (2005) e Zhou *et al.* (2018).

4.3.1.1 Requisito de disponibilidade

A disponibilidade de um provedor de nuvem para uma aplicação deve atender pelo menos ao limiar definido por um arquiteto de *software*. MinAvbty é o limiar de disponibilidade mínima definido por um arquiteto de *software*. Na Eq. 4.41, define-se AP.a como a disponibilidade de uma aplicação, a qual é o produto entre todos os termos de uma aplicação. A Eq. 4.42 define $Term_j.a$ como a disponibilidade do termo `j` de uma aplicação. A disponibilidade do $Term_j.a$ é a soma das disponibilidades de todas as sequências que pertencem ao termo `j`. Na Eq. 4.43, define-se a disponibilidade da sequência `i` do termo `j` nomeado de $Seq_{ij}.a$, a qual é o produto entre as disponibilidades de todos microsserviços, as disponibilidades de todos os *links* que pertencem à sequência, e a probabilidade da sequência ser selecionada. A probabilidade de uma sequência ser selecionada é representada na Eq. 4.43 por α.

$$AP.a = \prod_{j=1}^{numOf(Term)} Term_j.a \geqslant MinAvbty, \forall Term_j \mid Term_j \in AP \tag{4.41}$$

$$Term_j.a = \sum_{i=1}^{numOf(Seq)} Seq_{ij}.a, \forall Seq_{ij} \mid Seq_{ij} \in Term_j \tag{4.42}$$

$$Seq_{ij}.a = \alpha * \prod_{k=1}^{numOf(MS)} MS_k.a * \prod_{k=1}^{numOf(MS-1)} Links_k.a, \forall MS_k \mid MS_k \in Seq_{ij}, Link_k.ainSeq_{ij} \tag{4.43}$$

A disponibilidade para um microsserviço é calculada de forma semelhante à disponibilidade de uma aplicação. A diferença é que a disponibilidade de uma aplicação é baseada no fluxo de microsserviços, enquanto que a disponibilidade de um microsserviço é baseada na disponibilidade dos serviços de nuvem que são necessários para compor um microsserviço.

4.3.1.2 Requisito de Tempo de Resposta

O tempo de resposta de uma aplicação deve atender ao limiar definido por um arquiteto de *software*. Um arquiteto de *software* define MaxRT como o limiar máximo de tempo de resposta, o qual é a soma entre o limiar máximo de tempo de execução (MaxExecTime) e o limiar máximo de atraso (MaxDelay) como mostra a Eq.4.44. Na Eq. 4.45, AP.rt é definido como o tempo de resposta de uma aplicação, que recebe o maior tempo de resposta dentre todos termos de uma aplicação e deve ser menor ou igual a MaxRT. A Eq. 4.46 define $Term_j.rt$ como o tempo de resposta do termo `j`, ao qual é atribuído a soma dos tempos de resposta de todas as sequências que pertencem ao termo `j`. O tempo de resposta para a sequência `i` do termo `j` é representado pela Eq. 4.47, a qual recebe a soma dos tempos de resposta de todos os microsserviços que pertencem à sequência e dos atrasos entre cada microsserviço multiplicado pela probabilidade de seleção da sequência, a qual é representada por α. O atraso entre dois microsserviços de uma sequência é representado por $Link_k.rt$.

$$MaxRT = MaxExecTime + MaxDelay \tag{4.44}$$

$$AP.rt = \max_{j=1}^{numOf(Term)} \{Term_j.rt\} \leqslant MaxRT, \forall Term_j \mid Term_j \in AP \tag{4.45}$$

$$Term_j.rt = \sum_{i=1}^{numOf(Seq)} Seq_{ij}.rt, \forall Seq_{ij} \mid Seq_{ij} \in Term_j \tag{4.46}$$

$$Seq_{ij}.rt = \alpha * (\sum_{k=1}^{numOf(MS)} MS_k.rt + \sum_{k=1}^{numOf(MS-1)} Link_k.rt), \forall MS_k \mid MS_k \in Seq_{jj}, Link_k \in Seq_{ij} \tag{4.47}$$

O tempo de resposta para um microsserviço é calculado de forma semelhante ao tempo de resposta de uma aplicação. A diferença é que o tempo de resposta de uma aplicação é baseado no fluxo de microsserviços, enquanto que o tempo de resposta de um microsserviço é baseado no tempo de resposta dos serviços de nuvem que são necessários para compor um microsserviço.

4.3.1.3 Requisito de Custo

O custo da execução de uma aplicação não deve ser superior ao Budget, que é o limiar de custo definido por um arquiteto de *software*. Na Eq. 4.48, AP.c é definido como o custo de execução de aplicação, que é atribuído ao custo total de todos os termos que pertencem ao fluxo de microsserviço de uma aplicação e deve ser menor ou igual ao Budget. A Eq. 4.49 define $Term_j.c$ como o custo do termo `j`, ao qual é atribuído a soma dos custos de todas as sequências que pertencem ao termo `j`. O custo para a sequência `i` do termo `j` é representado pela Eq. 4.50, o qual recebe a soma dos custos de todos os microsserviços que pertencem à sequência e dos custos dos links entre cada microsserviço multiplicado pela probabilidade de seleção da sequência, a qual é representada por α. O custo de um link entre dois microsserviços de uma sequência é representado por $Link_k.c$.

$$AP.c = \sum_{j=1}^{numOf(Term)} Term_j.c \leqslant Budget, \forall Term_j \mid Term_j \in AP \tag{4.48}$$

$$Term_j.c = \sum_{i=1}^{numOf(Seq)} Seq_{ij}.c, \forall Seq_{ij} \mid Seq_{ij} \in Term_j \tag{4.49}$$

$$Seq_{ij}.c = \alpha * (\sum_{k=1}^{numOf(MS)} MS_k.c + \sum_{k=1}^{numOf(MS-1)} Link_k.c), \forall MS_k \mid MS_k \in Seq_{ij}, Link_k \in Seq_{ij} \tag{4.50}$$

O custo para um microsserviço é calculado de forma semelhante ao custo de uma aplicação. A diferença entre eles é que o custo de uma aplicação é baseado no fluxo de

microsserviços, enquanto que o custo de um microsserviço é baseado no custo dos serviços de nuvem que são necessários para compor um microsserviço. Além disso, uma aplicação possui no máximo `t` microsserviços e um microsserviço necessita no máximo de `r` serviços de nuvem.

4.3.2 Seleção de Provedores de Nuvem

Nesta subseção, o processo de seleção de provedores de nuvem é detalhado para o modelo LM^2K, e tem como base o modelo de serviço descrito na Seção 4.3.1. O processo proposto tem três níveis: o primeiro nível descobre todos os serviços de nuvem candidatos para um microsserviço em todos os provedores de nuvem; o segundo determina os provedores candidatos de todos os microsserviços de uma aplicação; e o terceiro seleciona os provedores para implantar todos os microsserviços de uma aplicação, mapeando o processo para o problema da mochila de múltiplas escolhas.

4.3.2.1 Primeiro Nível - Determinação dos Serviços de Nuvem Candidatos

Os serviços que atendem a todos os requisitos de um arquiteto de *software* são selecionados em todos os provedores disponíveis, o que resulta em um conjunto de serviços candidatos em cada provedor. Em seguida, classifica-se todos os serviços candidatos em cada provedor. Para isso, usa-se a técnica SAW como em Seghir e Khababa (2016) e como descrito nas Subseções 4.1.2.1 e 4.2.2.1 para os modelos UM^2K e UM^2Q respectivamente. Ao final deste nível, tem-se um conjunto de todos os serviços candidatos para cada microsserviço em todos os provedores disponíveis

4.3.2.2 Segundo Nível - Determinação dos Provedores Candidatos

Todas as combinações de serviços de um provedor para compor um microsserviço devem ser realizadas. Para isso, deve-se combinar os serviços candidatos para um microsserviço que são oferecidos pelo mesmo provedor. A Eq. 4.51 define SSP_{ik} como o conjunto de combinações candidatas do provedor `k` para o microsserviço `i`. Além disso, na Eq. 4.51, $comb_{ikj}$ indica a combinação `j` do provedor `k` para o microsserviço `i`. Uma combinação $comb_{ikj}$ é um conjunto de serviços de nuvem de classes diferentes para atender às necessidades do microsserviço `i`, como mostrado na Eq. 4.52. Em que $S_{l_n i}^{m_{in}}$ indica o serviço candidato l_n da classe m_n necessário para função `n` do microsserviço `i`. O microsserviço possui no máximo `r` funções e,

consequentemente, necessita de no máximo r serviços de nuvem, um provedor candidato tem no máximo m combinações e um microsserviço tem no máximo q provedores candidatos.

$$SSP_{ik} = \{comb_{ik1}, ..., comb_{ikm}\} \tag{4.51}$$

$$comb_{ikj} = \{S_{l_1 i}^{m_1}, ..., S_{l_r i}^{m_r}\} \tag{4.52}$$

O *score* e o custo devem ser calculados para cada combinação em SSP_{ik}, como mostrado pelas Eqs. 4.53 e 4.54. A Eq. 4.53 define $comb_{ikj}.score$ como a soma dos *score* de cada serviço de uma combinação mais a soma dos scores dos links entre cada serviço que pertence ao fluxo de serviços de um microsserviço. A Eq. 4.54 define $comb_{ikj}.c$ como o custo total de todos os serviços combinados mais o custo de cada link entre os serviços de uma combinação de acordo com o fluxo de serviços para um microsserviço.

$$comb_{ikj}.score = \sum_{x=1}^{r} Score(S_{l_x i}^{m_x}) + \sum_{x=1}^{r-1} Score(Link_x) \tag{4.53}$$

$$comb_{ikj}.c = \sum_{x=1}^{r} S_{l_x i}^{m_x}).c + \sum_{x=1}^{r-1} Link_x.c \tag{4.54}$$

Ao final do segundo nível, tem-se um conjunto de todas as combinações candidatas em todos os provedores disponíveis para todos os microsserviços de uma aplicação (SAMS), e cada um delas tem um *score* e um custo. A Eq. 4.55 define SMS_i o qual é um conjunto de provedores candidatos para o microsserviço i, em que SSP_{ik} indica o conjunto de combinações candidatas do provedor k para o microsserviço i. Na Eq. 4.56, SAMS é definido como o conjunto de provedores candidatos para cada microsserviço de uma aplicação, e uma aplicação tem no máximo t microsserviços.

$$SMS_i = \{SSP_{ik} \mid k \in [1, q]\} \tag{4.55}$$

$$SAMS = \{SMS_1, ..., SMS_t\} \tag{4.56}$$

4.3.2.3 Terceiro Nível - Seleção de Provedores para implantar Microsserviços

A fim de hospedar cada microsserviço de uma aplicação devem-se selecionar provedores de nuvem do SAMS (resultado do primeiro nível). Para isso, deve-se construir as combinações candidatas de SAMS, de forma que cada combinação tenha uma combinação candidata para cada microsserviço, e o conjunto dessas combinações é chamado de SAPP (Eq. 4.57). Além disso, a Eq. 4.57, $comb_{ik_ij_{k_i}}$ indica a combinação `j` do provedor candidato `k` para o microsserviço `i`. Posteriormente, a Eq. 4.58 define $SAPP_l.c$ como o custo de execução de cada combinação em SAPP e verifica se é menor ou igual ao custo de execução da aplicação, e exclui de outra forma. Em seguida, calcula-se o *score* de cada item em SAPP. Um *score* de uma combinação candidata ($SAPP_l.score$) é a soma do *score* de cada combinação de itens, que é mostrada na Eq. 4.59.

$$\begin{aligned} SAPP = \{(comb_{1k_1j_{k_1}}, ..., comb_{tk_tj_{k_t}}) \mid comb_{ik_ij_{k_i}} \in SSP_{ik}, SSP_{ik} \in SMS_i, \\ SMS_i \in SAMS, 1 \leqslant i \leqslant t, 1 \leqslant k_i \leqslant q_i, 1 \leqslant j_{k_i} \leqslant w_{k_i}\} \end{aligned} \tag{4.57}$$

$$SAPP_l.c = \sum_{i=1}^{t} comb_{ik_ij_{k_i}}.c + \sum_{i=1}^{t-1} Link_{k_i}.c \tag{4.58}$$

$$SAPP_l.score = \sum_{i=1}^{t} comb_{ik_ij_{k_i}}.score + \sum_{i=1}^{t-1} Link_{k_i}.socre \tag{4.59}$$

O processo de seleção descrito para o modelo LM^2K é um problema de otimização combinatória, pois os provedores para hospedar os microsserviços são selecionados a partir de várias combinações realizadas. Para isso, primeiro obtém-se as combinações candidatas para cada microsserviço em todos os provedores de nuvem disponíveis. Depois, obtém-se as combinações para uma aplicação, onde cada combinação contém um conjunto de combinações candidatas, às quais podem pertencer provedores de nuvem distintos, sendo uma para cada microsserviço da aplicação. Ao final, seleciona-se uma das combinações de aplicação, a qual contém um provedor para hospedar cada um dos microsserviços. O processo de seleção é mapeado para o problema da mochila de múltipla escolha, como no modelo UM^2K na subseção 4.1.2.3.

4.4 Resumo do Capítulo

Neste capítulo três modelos para a seleção de múltiplos provedores de nuvem foram propostos para hospedar uma aplicação distribuída baseada em microsserviços. Os modelos propostos são UM^2K, UM^2Q e LM^2K, e para cada um deles um modelo de serviços e o processo de seleção de múltiplos provedores de nuvem foram descritos. Os modelos UM^2K, UM^2Q e LM^2K diferem um do outro em relação ao grau de comunicação entre os microsserviços de uma aplicação, e a forma de mapear o problema. Os dois primeiros modelos desconsideram a comunicação, pois tratam de aplicações onde a comunicação entre os microsserviços possam ser desconsideradas. O terceiro modelo trata a comunicação entre os microsserviços considerando o fluxo de serviços de uma aplicação. Em relação ao mapeamento do processo de seleçao de provedores de nuvem, o primeiro e o terceiro modelo mapeiam para o problema da mochila de múltiplas escolhas. O segundo modelo trata os requisitos de forma independente para cada microsserviço, assim ele aborda o problema através de cotas de orçamento.

5 SOLUÇÕES PARA SELEÇÃO DE MÚLTIPLAS NUVENS

Este capítulo apresenta e descreve soluções computacionais para os modelos UM^2k, UM^2Q, e LM^2K descritos no capítulo 4. Os modelos UM^2K e LM^2k mapeiam o processo de seleção para o problema da mochila de múltiplas escolhas e na literatura existem alguns algoritmos para o problema da mochila. Soluções são propostas baseadas nos algoritmos dinâmicos, guloso e no algoritmo bioinspirado em colônia de formigas. O algoritmo dinâmico foi escolhido por sempre obter uma solução ótima, o guloso por aceitar uma entrada de dados maior e o algoritmo bioinspirado por obter uma solução melhor do que a do guloso e uma entrada de dados melhor do que o dinâmico. Para o modelo UM^2Q, uma solução foi proposta para selecionar os provedores para hospedar cada microsserviço de forma independente. Os requisitos são individuais, ou seja, nenhum requisito de um microsserviço interfere no requisito de outro microsserviço. Desta forma, pode-se oferecer alternativas para o processo de seleção de acordo com a necessidade de um arquiteto de *software*. Um resumo das contribuições deste capítulo são dadas abaixo:

- duas soluções são propostas para o modelo UM^2K, a primeira usando um algoritmo dinâmico e a outra usando um algoritmo guloso.
- uma solução é proposta para o modelo UM^2Q, a qual é uma solução baseada nos requisitos individuais de cada microsserviço de uma aplicação.
- três soluções são propostas para o modelo LM^2K, de forma que a primeira é baseada em um algoritmo dinâmico, a segunda em um algoritmo guloso e a terceira em um algoritmo bioinspirado, colônia de formigas.

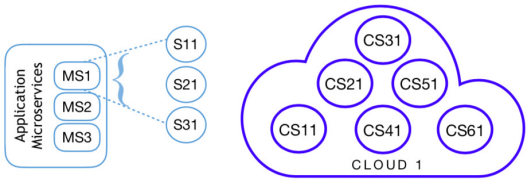
Este capítulo contempla as Seções 5.1, 5.2 e 5.3, as quais descrevem as soluções paras os modelos UM^2K, UM^2Q e LM^2K respectivamente. Para todas as soluções propostas, apresenta-se um exemplo, um diagrama e um algoritmo.

5.1 Soluções para o modelo UM^2K

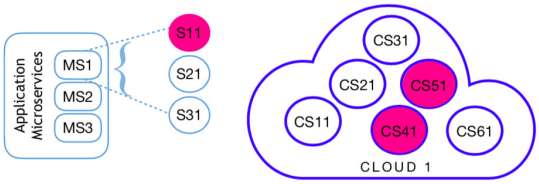
O modelo de microsserviços desvinculados e mapeados para a mochila (*Unlinked Microservice Mapped to the Knapsack* - UM^2k) tem como principal objetivo aplicações baseadas em microsserviços independentes. Neste modelo, o processo de seleção de múltiplos provedores é mapeado para o problema da mochila de múltiplas escolhas. De acordo com o mapeamento descrito nas subseções 4.1.2.3 e 4.3.2.3, o orçamento corresponde à capacidade máxima da

mochila, cada microsserviço representa uma classe de itens e cada combinação candidata de cada microsserviço representa um item da classe a que pertence. O objetivo é maximizar o *score*, o qual representa o lucro de cada item. Neste modelo, o *score* de uma aplicação será a soma dos *scores* de cada combinação selecionada para cada microsserviço. A combinação escolhida para cada microsserviço será a que possui maior *score* e que cabe na capacidade da mochila, ou seja, no orçamento da aplicação. Na subseção 5.1.1, a solução dinâmica é descrita para este modelo e na Subseção 5.1.2 a solução gulosa.

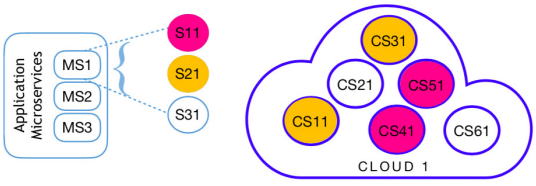
A fim de ilustrar o processo descrito para o modelo UM^2K, a Figura 28a apresenta um exemplo simplificado. O exemplo mostra uma aplicação baseada em microsserviços, a qual possui 3 microsserviços, representados por MS1, MS2 e MS3. O MS1 necessita de 3 serviços de nuvem, representados por S11, S21 e S31. A Figura 28a também ilustra os serviços do provedor *cloud* 1, o qual possui 6 serviços, representados por CS11, CS21, CS31, CS41, CS51 e CS61. Além disso, as Figuras 28b, 28c e 28d apresentam a seleção dos serviços candidatos do provedor 1 que atendem a cada um dos serviços de nuvem necessários para compor o microsserviço 1.



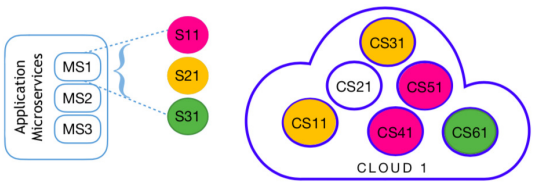
(a) Microserviços de uma Aplicação e um Provedor disponível



(b) Serviços Candidatos para o Serviço 1 do Microsserviço 1 da Aplicação.



(c) Serviços Candidatos para o Serviço 2 do Microsserviço 1 da Aplicação.



(d) Serviços Candidatos para o Serviço 3 do Microsserviço 1 da Aplicação.

FIGURA 28. EXEMPLO DE SERVIÇOS CANDIDATOS DE UM PROVEDOR PARA UM MICROSSERVIÇO.

A Figura 29, mostra as combinações candidatas para atender o microsserviço 1 no provedor *cloud* 1. O processo é repetido para todos os provedores disponíveis e em todos os microsserviços da aplicação.

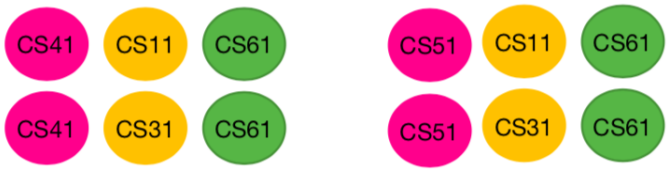


FIGURA 29. COMBINAÇÕES CANDIDATAS PARA O MICROSSERVIÇO DA FIGURA 28.

5.1.1 Solução Dinâmica para UM^2K

Para a solução dinâmica, primeiramente apresenta-se um exemplo. Em seguida, um diagrama, o qual mostra uma visão geral de cada nível descrito na Subseção 4.1.2. O algoritmo é descrito seção B do Apêndice B.

5.1.1.1 Exemplo Dinâmico para UM^2K

Nesta subseção, um exemplo é apresentado para o modelo UM^2K usando um algoritmo dinâmico, o qual é realizado usando as combinações candidatas geradas para cada microsserviços (obtido no segundo nível do modelo UM^2K na Subseção 4.1.2.2) e ilustrado pela Figura 30.

Microsserviço	Combinação	Score	Custo	Provedor	Orçamentos para uma aplicação												
					0	1	2	3	4	5	6	7	8	9	10	11	12
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0.6	2	P1	0	0	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	2	0.7	3	P2	0	0	0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	3	0.8	4	P3	0	0	0	0	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
2	1	0.5	2	P5	0	0	0.6	0.7	1.1	1.2	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	2	0.7	4	P4	0	0	0.6	0.7	0.8	0.8	1.3	1.4	1.5	1.5	1.5	1.5	1.5
	3	0.8	7	P3	0	0	0.6	0.7	0.8	0.8	0.8	0.8	0.8	1.4	1.5	1.6	1.6
3	1	0.6	1	P2	0	0.6	0.6	1.2	1.3	1.7	1.8	1.9	2.0	2.1	2.1	2.1	2.2
	2	0.7	3	P5	0	0	0.6	0.7	1.1	1.3	1.4	1.8	1.9	2.0	2.1	2.2	2.2
	3	0.8	8	P1	0	0	0.6	0.7	1.1	1.2	1.3	1.4	1.5	1.5	1.5	1.6	1.9

FIGURA 30. MATRIZ DE COMBINAÇÕES CANDIDATAS PARA O MODELO UM^2K.

As linhas da matriz da Figura 30 representam as combinações candidatas para cada microsserviço, neste exemplo cada microsserviço possui 3 combinações candidatas, e para cada uma delas são apresentados o *score* (coluna 3), o custo (coluna 4) e o provedor à qual ela pertence (coluna 5). Além disso, as demais colunas da matriz representam os valores possíveis para o orçamento de uma aplicação, cujo valor máximo é o limiar de orçamento definido por um arquiteto de *software* para a aplicação. Desta forma, cada coluna contém um valor inteiro,

iniciando em 0 até o valor do limiar de orçamento de uma aplicação. Neste exemplo simplificado, o valor do orçamento é 12, portanto a matriz possui 13 colunas correspondentes aos possíveis orçamentos de uma aplicação. Por fim, as demais células da matriz é preenchida observando-se o custo e o *score* de cada combinação candidata, bem como os orçamentos possíveis para uma aplicação.

Para preencher as colunas referentes aos orçamentos de uma aplicação da Figura 30, a primeira coluna deve ser preenchida com zero, bem como a primeira linha, a qual é uma linha extra que não contempla nenhuma combinação. Para as demais colunas existem duas situações que devem ser observadas para o preenchimento da matriz. A primeira está relacionada às combinações candidatas para primeiro microsserviço da matriz e a segunda às combinações dos outros microsserviços da matriz. Para preencher cada célula referente às combinações do primeiro microsserviço, verifica-se o custo da combinação, se for maior que o valor do orçamento da célula que deseja preencher, o valor permanece como da célula anterior na mesma coluna. Caso contrário, a célula é preenchida com o valor do *score* da combinação.

Preenchido as células da matriz relacionadas às combinações candidatas do primeiro microsserviço, o próximo passo é preencher as células correspondentes aos demais microsserviços. Para isto, se o custo da combinação for maior que o valor do orçamento da célula que deseja preencher, o valor da célula será o maior valor dentre as combinações candidatas do microsserviço anterior, com o orçamento correspondente ao da célula (na mesma coluna da célula). Caso contrário, será o maior valor entre V_1 e V_2. V_1 é o maior valor dentre as combinações candidatas do microsserviço anterior, como o orçamento correspondente ao da célula (na mesma coluna da célula). V_2 é a soma entre o *score* da combinação e o maior valor dentre as combinações do microsserviço anterior, com orçamento `k`, onde `k` corresponde à diferença entre orçamento correspondente à célula que deseja preencher e o custo da combinação.

De acordo com as regras de preenchimento da matriz, pode-se observar na matriz da Figura 30, que as linhas referentes às combinações candidatas para o microsserviço 1 foram preenchidas com os valores do *score* da respectiva combinação, quando o custo da combinação é menor que o orçamento correspondente à célula. As linhas referentes as combinações candidatas do microsserviço 2 foram preenchidas observando as regras para os demais microsserviços. Assim, como exemplo, a célula correspondente à combinação candidata 2 para o microsserviço 2 cujo o orçamento é 9 foi preenchida com o valor 1.5, o qual é a soma entre 0.7 e 0.8. O valor 0.7 corresponde ao *score* da combinação candidata 2 para o microsserviço 2 e o valor 0.8

corresponde ao maior *score* do microsserviço 1 cujo o orçamento é 5. O orçamento 5 foi obtido através da diferença entre o orçamento cujo valor é 9 e o valor do custo da combinação 2 para o microsserviço 2 cujo valor é 4. As linhas das combinações do microsserviço 3 são preenchidas como as do microsserviço 2, só que para o microsserviço 3 se observa as combinações do microsserviço 2 enquanto que as do microsserviço 2 se observa as combinações do microsserviço 1.

Depois do preenchimento da matriz, uma combinação deve ser selecionada para cada microsserviço. A matriz da Figura 31, mostra a seleção das combinações para o exemplo. O processo de seleção das combinações inicia-se observando o último microsserviço, no exemplo o microsserviço 3. A combinação selecionada será a que tiver maior valor preenchido na última coluna da matriz, ou seja, cujo orçamento é o maior valor possível. No exemplo existem dois valores iguais na última coluna para o último microsserviço, mas como a seleção inicia a partir da última célula da matriz a combinação selecionada será a primeira encontrada. Depois, na linha da combinação selecionada verifica se existe um valor maior para aquela combinação, caso exista o valor selecionado será alterado para a coluna que tiver o maior valor. No exemplo isso não acontece, portanto a combinação selecionada para o microsserviço 3 é a combinação 2 com o valor de *score* da última coluna.

Para selecionar uma combinação para o próximo microsserviço, que no exemplo é o microsserviço 2. Primeiro, obtém-se a coluna que deve ser analisada. Para isto, calcula-se a diferença entre o valor da coluna selecionada para o microsserviço 3, no caso a coluna cujo o orçamento é 12, e o valor do custo da combinação selecionada para o microsserviço 3, no caso o custo é 3. Assim, a coluna que deve ser analisada para o microsserviço 2 é a coluna cujo o orçamento é 9 (12 - 3). Desta forma, deve-se selecionar uma combinação candidata para o microsserviço 2 que tenha o maior lucro na coluna 9 da matriz, que no exemplo é a combinação 2. O processo realizado para selecionar a combinação para o microsserviço 2 deve ser realizado para os demais microsserviços, no exemplo é o microsserviço 1. Portanto, a coluna selecionada para o microsserviço 1 é a coluna cujo o orçamento é 5 (9 - 4), e o maior lucro na coluna onde o orçamento é 5 para o microsserviço 1 é o da combinação candidata 3.

5.1.1.2 Diagrama Dinâmico para UM^2K

O diagrama da Figura 32 possui como entrada um conjunto de provedores de nuvem disponíveis e um conjunto de microsserviços de uma aplicação. Além disso, o diagrama é

Microsserviço	Combinação	Score	Custo	Provedor	Orçamentos para uma aplicação												
					0	1	2	3	4	5	6	7	8	9	10	11	12
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0.6	2	P1	0	0	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	2	0.7	3	P2	0	0	0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	3	0.8	4	P3	0	0	0	0	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
2	1	0.5	2	P5	0	0	0.6	0.7	1.1	1.2	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	2	0.7	4	P4	0	0	0.6	0.7	0.8	0.8	1.3	1.4	1.5	1.5	1.5	1.5	1.5
	3	0.8	7	P3	0	0	0.6	0.7	0.8	0.8	0.8	0.8	0.8	1.4	1.5	1.6	1.6
3	1	0.6	1	P2	0	0.6	0.6	1.2	1.3	1.7	1.8	1.9	2.0	2.1	2.1	2.1	2.2
	2	0.7	3	P5	0	0	0.6	0.7	1.1	1.3	1.4	1.8	1.9	2.0	2.1	2.2	2.2
	3	0.8	8	P1	0	0	0.6	0.7	1.1	1.2	1.3	1.4	1.5	1.5	1.5	1.6	1.9

FIGURA 31. SELEÇÃO DE COMBINAÇÕES PARA CADA MICROSSERVIÇO.

composto por três partes, cada parte representa um nível da solução usando algoritmo dinâmico para o modelo UM^2K. O primeiro nível contém a descoberta dos serviços candidatos para cada microsserviço de uma aplicação, bem como a classificação dos serviços baseado no *score* calculado de acordo com os requisitos de um arquiteto de *software*. O segundo nível combina os serviços candidatos em cada provedor, onde cada combinação deve conter todos os serviços de nuvem necessários para um microsserviço. Além disso, para cada combinação construída é calculado o *score* e o custo. No terceiro nível, construiu-se uma matriz de solução, como descrito no exemplo da subseção anterior, e retorna um conjunto de combinações contendo uma combinação para cada microsserviço de uma aplicação.

5.1.2 Solução Gulosa para UM^2K

Nesta subseção, a solução gulosa é descrita para o modelo UM^2K. Primeiro, apresenta-se um exemplo. Depois, um diagrama que contém uma visão geral. O algoritmo guloso está descrito na seção B do Apêndice B.

5.1.2.1 Exemplo Guloso para UM^2K

Um exemplo é apresentado nesta subseção para o modelo UM^2K usando um algoritmo guloso, o qual é baseado no exemplo simplificado descrito no início da Seção 5.1. Assim, a Figura 33a mostra o conjunto de todas as combinações candidatas para todos os microsserviços de uma aplicação, o qual possui 3 combinações candidatas para cada um dos 3 microsserviços. Além disso, a Figura 33a apresenta para cada combinação candidata o *score*, o custo e o provedor ao qual pertence. As combinações candidatas de cada microsserviço devem ser ordenadas pelo

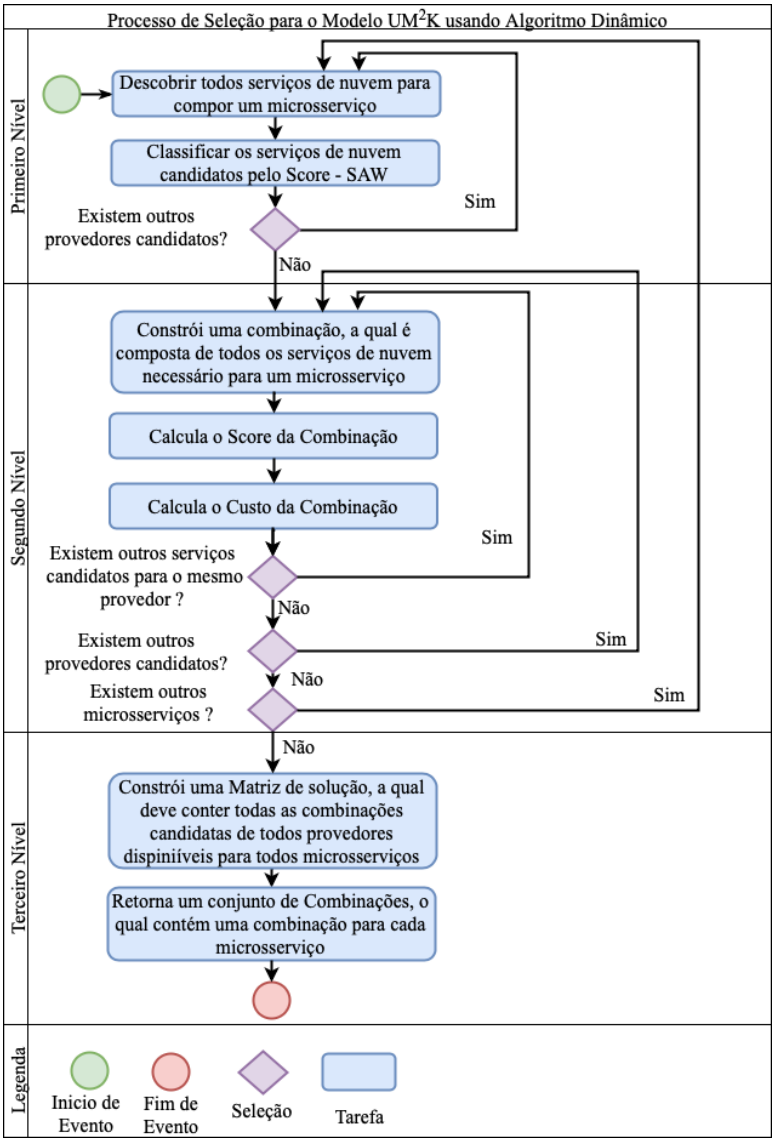


FIGURA 32. DIAGRAMA PARA O MODELO UM^2K USANDO ALGORITMO DINÂMICO.

custo da combinação. Para este exemplo, o limiar do orçamento definido por um arquiteto de *software* é 12.

Microsserviço	Combinação	Score	Custo	Provedor
1	1	0.6	2	P1
	2	0.5	3	P2
	3	0.8	4	P3
2	1	0.5	2	P5
	2	0.7	4	P4
	3	0.8	7	P3
3	1	0.6	1	P2
	2	0.7	3	P5
	3	0.8	8	P1

(a) Todas as combinações candidatas.

Microsserviço	Combinação	Score	Custo	Provedor
1	1	0.6	2	P1
	2	**0.5**	**3**	**P2**
	3	0.8	4	P3
2	1	0.5	2	P5
	2	0.7	4	P4
	3	0.8	7	P3
3	1	0.6	1	P2
	2	0.7	3	P5
	3	0.8	8	P1

(b) Exclusão de combinações.

Microsserviço	Combinação	Score	Custo	Provedor
1	1	0.6	2	P1
	3	0.8	4	P3
2	1	0.5	2	P5
	2	0.7	4	P4
	3	0.8	7	P3
3	1	0.6	1	P2
	2	0.7	3	P5
	3	0.8	8	P1

(c) Combinações candidatas resultantes.

FIGURA 33. COMBINAÇÕES CANDIDATAS USANDO O ALGORITMO GULOSO PARA O MODELO UM^2K.

A Figura 33b mostra a exclusão da combinações candidatas de cada microsserviço que são dominadas ou lp-dominadas. Para isso, observa-se as Eqs. 5.1 e 5.2, descritas em Kellerer *et al.* (2004). A Eq. 5.1 exclui as combinações candidatas dominadas, ou seja, exclui a combinação candidata `j`, cujo o custo ($cost_j$) é maior do que o custo da combinação candidata `i` ($cost_i$) e o *score* (sc_j) é menor que o *score* da combinação `i`(sc_i). A Eq. 5.2 exclui as combinações candidatas lp-dominadas, ou seja, exclui a combinação candidata `j`, cujo o custo está entre os custos de `k` e `i` ($cost_k > cost_j > cost_i$) e o *score* está entre os *scores* de `k` e `i` ($sc_k > sc_j > sc_i$), mas o custo benefício é menor. Neste exemplo, somente uma combinação será excluída, a combinação 2 do microsserviço 1, pois possui um custo maior que a combinação 1, mas possui menor *score* que a combinação 1. Desta forma, a Figura 33c, mostra as combinações candidatas que estão aptas para o processo de seleção.

$$cost_j \geqslant cost_i, sc_j \leqslant sc_i \mid 1 \leqslant j \leqslant numof(comb), i \leqslant j \tag{5.1}$$

$$\frac{sc_k - sc_j}{cost_k - cost\,j} \geqslant \frac{sc_j - sc_i}{cost_j - cost\,i} \mid 1 \leqslant j \leqslant numof(comb), i < j < k \tag{5.2}$$

Depois de excluir todas as combinações candidatas dominadas e lp-dominadas, quatro passos devem ser realizados para selecionar uma combinação para cada microsserviço. No primeiro passo, a eficiência incremental deve ser calculada para todas as combinações candidatas, exceto para as que possuem menor custo em cada um dos microsserviço, como mostra a Figura 34a. A Eq. 5.3 apresenta a fórmula para a eficiência incremental, a qual representa o custo benefício de uma combinação candidata. Na Eq. 5.3, `i` e `j` representam combinações candidatas para um microsserviço, e `j` deve estar imediatamente depois de `i`. Além disso, na Eq. 5.3, *sc* e *cost* são os *score* e o custo das combinações.

$$e = \frac{sc_j - sc_i}{cost_j - cost\,i} \mid 1 < j \leqslant numof(comb), i \leqslant j \tag{5.3}$$

No segundo passo, as combinações são colocadas em ordem decrescente pelo valor da eficiência incremental independentemente do microsserviço, como mostrado na Figura 34b. No terceiro passo, a combinação candidata com menor custo de cada microsserviço é colocada em um conjunto como uma solução inicial, e os seus respectivos custos são decrementados do orçamento da aplicação. No exemplo: 12 - (2 + 2 + 1) = 7, pois 12 é o orçamento e os outros

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
1	1	0.6	2	P1	
	3	0.8	4	P3	**0.2/2**
2	1	0.5	2	P5	
	2	0.7	4	P4	**0.2/2**
	3	0.8	7	P3	**0.1/3**
3	1	0.6	1	P2	
	2	0.7	3	P5	**0.1/2**
	3	0.8	8	P1	**0.1/5**

(a) Eficiência Incremental.

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
1	3	0.8	4	P3	**0.2/2**
2	2	0.7	4	P4	**0.2/2**
3	2	0.7	3	P5	**0.1/2**
2	3	0.8	7	P3	**0.1/3**
3	3	0.8	8	P1	**0.1/5**

(b) Combinações ordenadas por Eficiência Incremental.

FIGURA 34. SELEÇÃO DAS COMBINAÇÕES PARA O MODELO UM^2K USANDO O ALGORITMO GULOSO.

valores correspondem ao menor custo para o microsserviço, 1, 2 e 3, respectivamente. Assim, a solução inicial é composta pelas combinações 1 de cada microsserviço.

No último passo, verifica-se a possibilidade de substituir cada combinação na solução inicial por uma que está no conjunto ordenado pela eficiência incremental da Figura 34b, a fim de obter uma solução com maior custo benefício. De forma que, uma combinação de um microsserviço só possa ser substituída por outra do mesmo microsserviço. Além disso, o valor do custo da combinação não pode ultrapassar o orçamento. O processo de substituição termina quando o custo de uma combinação é maior que o orçamento.

Todo o processo de substituição para o exemplo é realizado observando a ordem das combinações na Figura 34b. Como a primeira combinação da Figura 34b pertence ao microsserviço 1, então, caso o custo da combinação 3 caiba no orçamento, deve-se substituir a combinação 1 do microsserviço 1 pela combinação 3. Para verificar se o custo da combinação 3 cabe no orçamento, obtém se o orçamento atual, 7, calculado no terceiro passo. Depois decrementa do orçamento atual a diferença entre o custo da combinação 3 e o da combinação 1 do microsserviço 1, a qual será 2 (4-2). Assim, o orçamento torna-se 5 (7 - 2).

O processo de substituição deve ser realizado até o custo de uma combinação não caiba no orçamento. Assim, para substituir a combinação 1 do microsserviço 2 pela combinação 2 (2ªcombinação da Figura 34b), verificar-se que o orçamento atual é 5, de acordo com a última substituição realizada. Depois decrementa do orçamento atual a diferença entre o custo da combinação 2 e o da combinação 1 do microsserviço 2, a qual será 2 (4-2). Assim, o orçamento torna-se 3 (5 - 2). Para substituir a combinação 1 do microsserviço 3 pela combinação 2 (3ªcombinação da Figura 34b), observa-se que o orçamento atual é 3. Depois decrementa do orçamento atual a diferença entre o custo da combinação 2 e o da combinação 1 do microsserviço 3, a qual será 2 (3-1). Assim, o orçamento torna-se 1 (3 -2). Com o orçamento atual igual a 1, não

é possível fazer outra substituição. Logo, o processo de substituição para o exemplo é concluído e o resultado final é mostrado na Figura 35, a qual destaca as combinações selecionadas para hospedar os microsserviços.

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
1	1	0.6	2	P1	
	3	0.8	4	P3	**0.2/2**
2	1	0.5	2	P5	
	2	0.7	4	P4	**0.2/2**
	3	0.8	7	P3	**0.1/3**
3	1	0.6	1	P2	
	2	0.7	3	P5	**0.1/2**
	3	0.8	8	P1	**0.1/5**

FIGURA 35. COMBINAÇÕES PARA OS MICROSSERVIÇOS USANDO ALGORITMO GULOSO EM UM^2K.

5.1.2.2 Diagrama Guloso para UM^2K

Nesta subseção, na Figura 36 é apresentado um diagrama para o modelo UM^2K usando um algoritmo guloso. O diagrama contém três níveis, como descrito na subseção 4.1.2. O primeiro nível descobre e classifica os serviços de nuvem que atendem aos requisitos dos microsserviços de uma aplicação. No segundo nível, constrói-se as combinações que contém todos os serviços de nuvem candidatos para um microsserviço em um mesmo provedor. O processo é repetido para todos os serviços candidatos dentro de um mesmo provedor, bem como em todos os provedores disponíveis e para todos os microsserviços de uma aplicação. O terceiro nível mapeia o processo para o problema da mochila de múltiplas escolhas, onde o orçamento da aplicação é a capacidade máxima da mochila e o *score* é o lucro de uma combinação candidata, afim de obter as combinações com maior *score* que contemplam o orçamento. Para isto, as combinações candidatas para cada microsserviço são ordenadas pelo custo. Depois, as combinações que estão de acordo com as Eqs, 5.1 e 5.2 são excluídas. Em seguida, a eficiência incremental é calculada para cada combinação, e as combinações são colocadas em ordem decrescente de acordo com a eficiência incremental. Por fim, as combinações são selecionadas observando o orçamento da aplicação e a eficiência incremental.

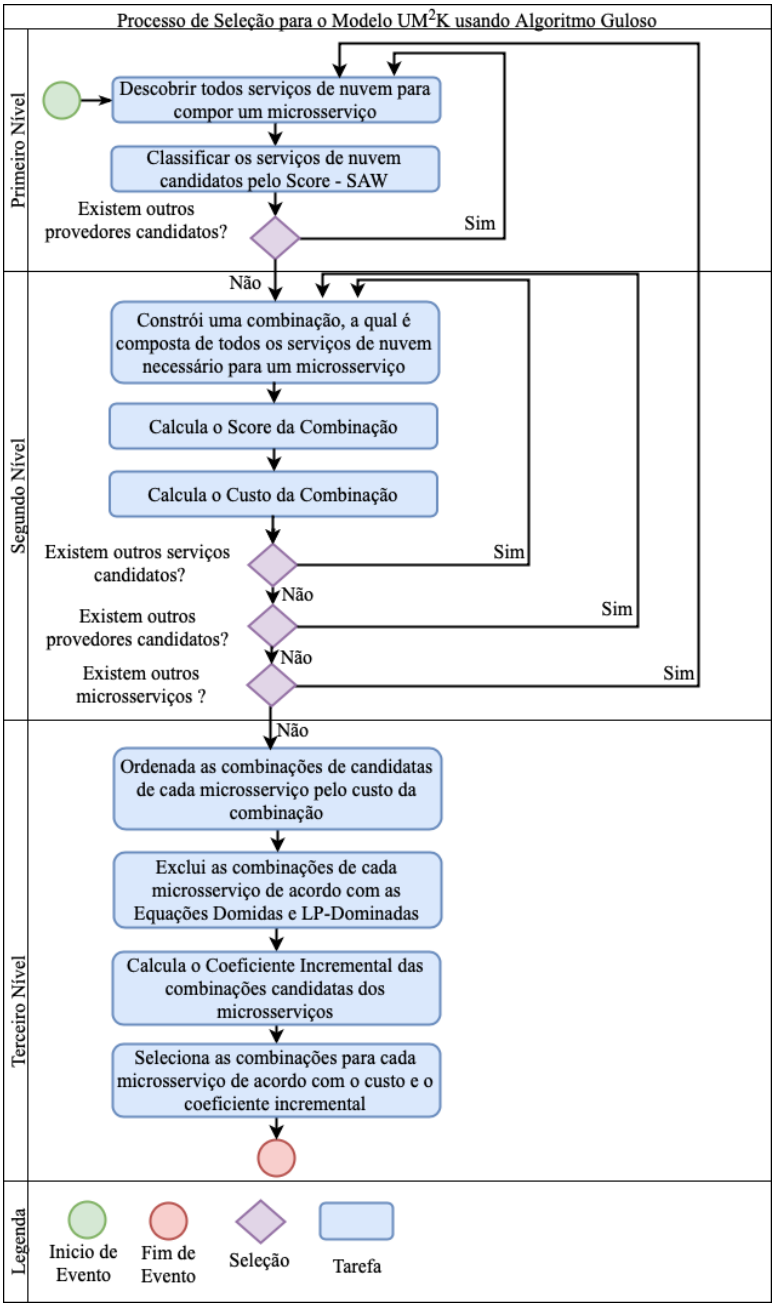


FIGURA 36. DIAGRAMA PARA O MODELO UM^2K USANDO ALGORITMO GULOSO.

5.2 Solução para o modelo UM^2Q

O modelo de microsserviços desvinculados e mapeados em cotas de orçamento (*Unlinked Microservice Mapped to the Quota* - UM^2Q), tem como objetivo principal selecionar múltiplos provedores de nuvem para hospedar aplicações baseadas em microsserviços independentes. O processo de seleção observa os requisitos de cada microsserviço bem como os requisitos de um arquiteto de *software* de forma individual, ou seja, os valores dos requisitos para um microsserviço não influenciam em outro microsserviço. Assim, o processo de seleção pode

paralelamente selecionar uma combinação para cada microsserviço. Nesta seção, um exemplo, um diagrama e no Apêndice B na Seção B um algoritmo é apresentado para a solução deste modelo.

5.2.1 *Exemplo para* UM^2Q

Nesta subseção, um exemplo é apresentado e descrito para o modelo UM^2Q, o qual possui como base o exemplo descrito no preâmbulo da Seção 5.1. Primeiro identifica-se os serviços que atendem aos requisitos de um microsserviço em um provedor, então faz-se todas as combinações possíveis entre eles para compor um microsserviço. Em seguida, o *score* da combinação é calculado e verificado se o custo da aplicação é menor ou igual à cota do orçamento para o microsserviço. Então, seleciona-se de um provedor a combinação com maior *score*. O processo é repetido em todos os provedores para todos os microsserviços.

A Figura 37a mostra a aplicação do exemplo, a qual possui 3 microsserviços, cada um deles com uma cota de orçamento. As combinações candidatas para cada microsserviço são as combinações com maior *score* em cada provedor. A Figura 37b mostra o resultado final. Através dos resultados é possível observar que para cada microsserviço foi selecionada a combinação com maior *score*, cujo custo seja menor ou igual à cota de orçamento para o microsserviço.

Microsserviço	Combinação	Cota	Score	Custo	Provedor
1	1	4	0.6	2	P1
	2		0.7	3	P2
	3		0.8	4	P3
2	1	4	0.5	2	P5
	2		0.7	4	P4
	3		0.8	7	P3
3	1	4	0.6	1	P2
	2		0.7	3	P5
	3		0.8	8	P1

(a) Todas as combinações candidatas.

Microsserviço	Combinação	Cota	Score	Custo	Provedor
1	1	4	0.6	2	P1
	2		0.7	3	P2
	3		0.8	4	P3
2	1	4	0.5	2	P5
	2		0.7	4	P4
	3		0.8	7	P3
3	1	4	0.6	1	P2
	2		0.7	3	P5
	3		0.8	8	P1

(b) Seleção de combinações.

FIGURA 37. SELEÇÃO DE COMBINAÇÕES POR MICROSSERVIÇO PARA O MODELO UM^2Q.

5.2.2 *Diagrama para* UM^2Q

Um diagrama é apresentado na Figura 38 para o modelo UM^2Q. A Figura 38 mostra uma visão geral de cada nível descrito em Subseção 4.2.2. O diagrama possui como entrada um conjunto das capacidades dos provedores de nuvem disponíveis e um conjunto dos requisitos dos

microsserviços de uma aplicação. Ele está dividido em três partes, cada uma delas representa um nível do modelo UM^2Q. Na primeira parte do diagrama existem duas atividades: a primeira descobre os serviços candidatos para um microsserviço em um provedor e, a segunda classifica todos os serviços candidatos.

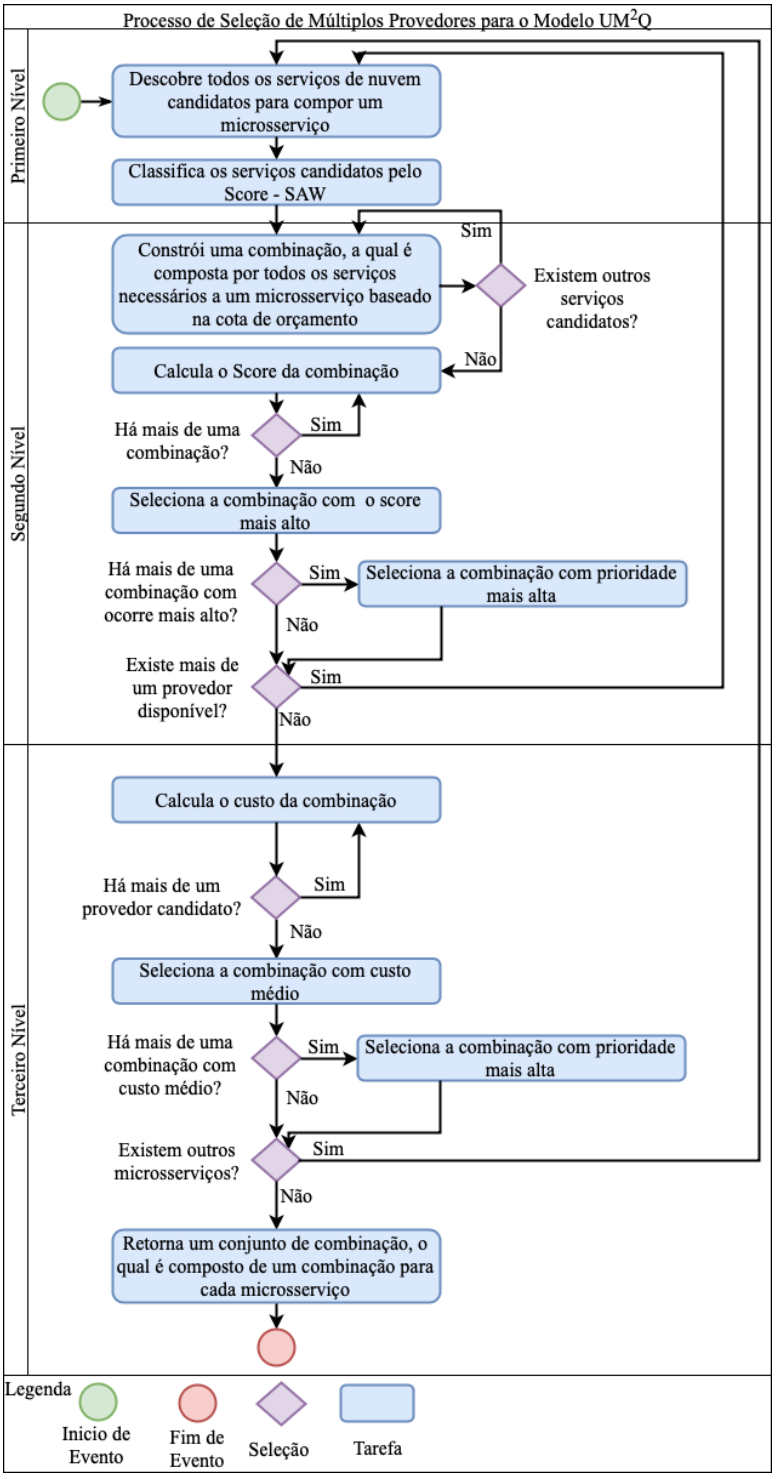


FIGURA 38. DIAGRAMA PARA O MODELO UM^2Q.

A segunda parte do diagrama mostra quatro tarefas, que visam criar todas as combinações candidatas para um microsserviço em todos os provedores de nuvem disponíveis. Para isso, obtém-se todas as combinações candidatas de um único provedor. Uma combinação candidata é composta por todos os serviços de nuvem necessários para um microsserviço e o custo da combinação deve ser menor ou igual à cota do orçamento. Em seguida, o *score* de uma combinação é calculado. Em seguida, seleciona-se a combinação com o *score* mais alto. Se houver mais de uma combinação com o *score* mais alto, a combinação escolhida é a que tem a prioridade mais alta. Todo o processo é repetido para todos os provedores disponíveis.

A terceira parte do diagrama da Figura 38 apresenta quatro tarefas, que visam selecionar uma combinação dentre as combinações de cada provedor de nuvem candidato para cada microsserviço. Na primeira e segunda tarefas, calcula-se o custo das combinações e seleciona-se a combinação com o custo médio. Se houver mais de uma combinação com o custo médio. Em seguida, seleciona-se a combinação com a prioridade mais alta. Todo o processo é repetido para todos os microsserviços de uma aplicação. Finalmente, um conjunto de combinações é retornado, em que cada combinação representa um microsserviço.

5.3 Soluções para o modelo LM^2K

Esta seção descreve o modelo de microsserviços vinculados e mapeados para a mochila (*Linked Microservice Mapped to the Knapsack - LM^2k*). Este modelo tem como objetivo selecionar múltiplos provedores de nuvem para aplicações baseadas em microsserviços que se comunicam entre si. Como no modelo UM^2K, neste modelo, o processo de seleção de múltiplos provedores é mapeado para o problema da mochila de múltiplas escolhas. De acordo com o mapeamento descrito nas subseções 4.1.2.3 e 4.3.2.3, o orçamento corresponde à capacidade máxima da mochila, cada microsserviço representa uma classe de itens, e cada combinação candidata de cada microsserviço representa um item da classe que pertence. O objetivo é maximizar o *score*, o qual representa o lucro de cada item. Neste modelo o *score* de uma aplicação será a soma dos *scores* de cada combinação selecionada para cada microsserviço, adicionado dos *scores* dos *links* de comunicação entre os provedores que hospedam os microsserviços. A combinação escolhida para cada microsserviço será a que possui maior *score*, cujo custo total não exceda a capacidade da mochila, ou seja, ao orçamento da aplicação. O custo total é a soma entre os custos de todas as combinações selecionadas para todos os microsserviços e os custos dos *links* de comunicação entre todos os provedores selecionados. A subseção 5.3.1 descreve a

solução dinâmica para este modelo, na Subseção 5.3.2 a solução gulosa. Na Subseção 5.3.3 a solução bioinspirada, usando o algoritmo colônia de formiga.

Antes de descrever cada uma das soluções, um exemplo simplificado é apresentado para o modelo LM^2K, como mostrado na Figura 39. O exemplo mostra uma aplicação baseada em microsserviços, a qual possui 3 microsserviços, representados por MS1, MS2 e MS3. O MS1 necessita de 3 serviços de nuvem, representados por S11, S21 e S31. O MS3 necessita de 2 serviços de nuvem representados por S13 e S23. A Figura 39 também ilustra os serviços dos provedores *Cloud* 1 e *Cloud* 2. O provedor *Cloud* 1 possui 6 serviços, representados por CS11, CS21, CS31, CS41, CS51 e CS61. O provedor *Cloud* 2 possui 5 serviços, representados por CS12, CS22, CS32, CS42 e C52.

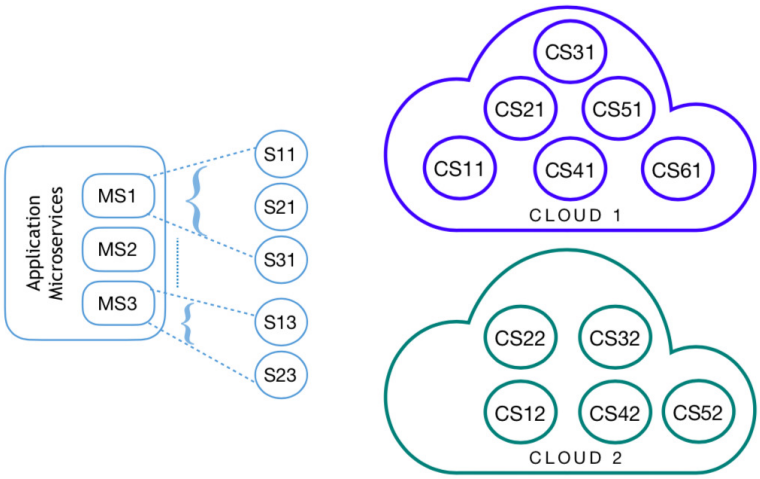


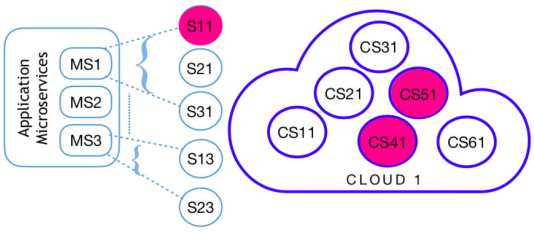
FIGURA 39. EXEMPLO SIMPLIFICADO PARA O MODELO LM^2K.

As Figuras 40a, 40b, 40c, 40d e 40e apresentam a seleção dos serviços candidatos do provedor *Cloud* 1 e *Cloud* 2 que atendem cada um dos serviços de nuvem necessários para compor o MS1 e MS3, respectivamente.

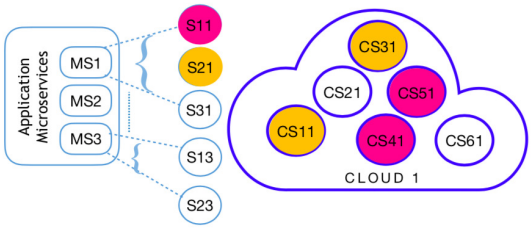
A Figura 41a, mostra as 4 combinações candidatas que atendem ao microsserviço MS1 no provedor *Cloud* 1, enquanto que a Figura 41b mostra as 4 combinações candidatas que atendem ao microsserviço MS3 no provedor *Cloud* 2.

Nas Figuras 41a e 41b as setas representam o *link* de comunicação entre os serviços de nuvem de um mesmo provedor, os *links* de comunicação estão de acordo com o fluxo de atividades do microsserviço.

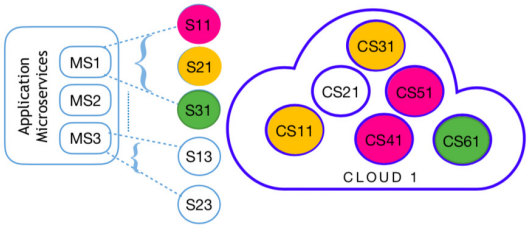
A Figura 42 ilustra o *link* de comunicação entre os provedores selecionados para hospedarem os microsserviços MS1 e MS3, respectivamente. O processo é repetido para todos os provedores disponíveis e em todos os microsserviços de uma aplicação.



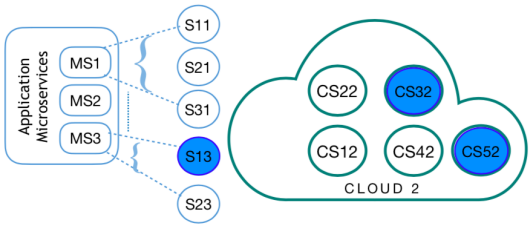
(a) Serviços Candidatos para o Serviço 1 do Microsserviço 1 da Aplicação.



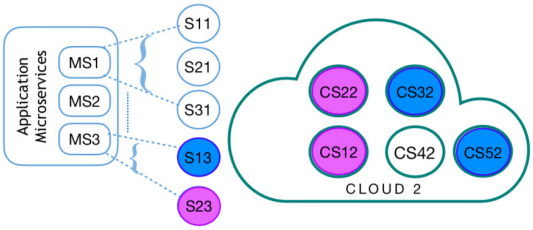
(b) Serviços Candidatos para o Serviço 2 do Microsserviço 1 da Aplicação.



(c) Serviços Candidatos para o Serviço 3 do Microsserviço 1 da Aplicação.



(d) Serviços Candidatos para o Serviço 1 do Microsserviço 3 da Aplicação.



(e) Serviços Candidatos para o Serviço 2 do Microsserviço 3 da Aplicação.

FIGURA 40. EXEMPLO DE SERVIÇOS CANDIDATOS PARA OS MICROSSERVIÇOS MS1 E MS3.

(a) Combinações para MS1 em Cloud1.

(b) Combinações para MS3 em Cloud2.

FIGURA 41. COMBINAÇÕES CANDIDATAS PARA UM MS1 E MS3 EM CLOUD1 E CLOUD2 RESPECTIVAMENTE.

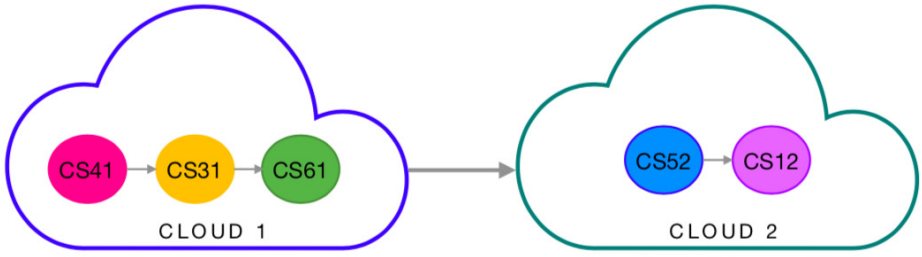


FIGURA 42. PROVEDORES SELECIONADOS PARA HOSPEDAR MS1 E MS3.

5.3.1 Solução Dinâmica para LM^2K

Um exemplo é apresentado, nesta subseção, para a solução dinâmica do modelo LM^2K. Depois, um diagrama é apresentado, o qual mostra uma visão geral de cada nível descrito em Subseção 4.3.2. Os Algoritmos 5 e 6 são descritos no Apêndice B na Seção B, através dos quais é possível observar todos os níveis do processo de seleção descrito na Subseção 4.3.2.

5.3.1.1 Exemplo Dinâmico para LM^2K

O exemplo apresentado, nesta subseção, usa as combinações candidatas geradas para cada microsserviços apresentadas no exemplo do preâmbulo deste modelo na Figura 41, e mostradas na matriz da Figura 43. Como no exemplo do modelo UM^2K na Figura 30, o exemplo simplificado para o modelo LM^2K na Figura 43 possui 3 microsserviços e para cada um deles existem 3 combinações candidatas, e a matriz apresenta o *score*, o custo e o provedor para cada uma das combinações candidatas. O limiar do orçamento definido pelo arquiteto de *software* para a aplicação é 12, assim, a matriz possui 13 orçamentos de 0 a 12. O preenchimento da matriz leva em consideração os *links* de comunicação entre os provedores das combinações candidata, diferentemente do modelo UM^2K.

Microsserviço	Combinação	Score	Custo	Provedor	Orçamentos para uma aplicação												
					0	1	2	3	4	5	6	7	8	9	10	11	12
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0.6	2	P1	0	0	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	2	0.7	3	P2	0	0	0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	3	0.8	4	P3	0	0	0	0	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
2	1	0.5	2	P5	0	0	0.6	0.7	1.1	0.8	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	2	0.7	4	P4	0	0	0.6	0.7	0.8	0.8	1.3	1.4	1.5	0.8	1.5	1.5	1.5
	3	0.8	7	P3	0	0	0.6	0.7	0.8	0.8	0.8	0.8	0.8	1.4	1.5	1.6	1.6
3	1	0.6	1	P2	0	0.6	0.6	1.2	1.3	1.7	1.4	1.9	2.0	2.1	2.1	2.1	2.2
	2	0.7	3	P5	0	0	0.6	0.7	1.1	1.3	1.4	1.8	1.5	2.0	2.1	2.2	2.1
	3	0.8	8	P1	0	0	0.6	0.7	1.1	0.8	1.3	1.4	1.5	1.5	1.5	1.6	1.9

FIGURA 43. MATRIZ DE COMBINAÇÕES CANDIDATAS PARA O MODELO LM^2K.

A Figura 44 destaca alguns valores de preenchimento da matriz, para que se possa observar a diferença no preenchimento da matriz em relação à solução dinâmica para o modelo UM^2K. Neste exemplo, supõe-se valores para os *links* de comunicação entre provedores. Nas células correspondentes às combinações candidatas 1 e 2 do microsserviço 2 observa-se que a combinação 1 pertence ao provedor P5 e a 2 ao provedor P4. Na coluna cujo orçamento é 5 para a combinação 1 observa-se que o valor de preenchimento deveria ser 1.2 (0.5 + 0.7),

por somar o valor do *score* da combinação com o maior *score* do microsserviço 1 na coluna onde o orçamento é 3 (5 - 2). Porém, de acordo com a suposição para os *links* de comunicação o valor dos requisitos entre os provedores P4 e P5 não atendem às necessidades da aplicação. Nesta situação, a solução dinâmica para este modelo difere da solução dinâmica para o modelo UM^2K. Portanto, a célula é preenchida com o maior *score* do microsserviço anterior, no caso o microsserviço 1, na coluna onde o orçamento é 5. O mesmo acontece com a combinação 2 para o microsserviço 2 na coluna onde o orçamento é 9. Os valores destas células refletem nos valores das combinações 3 e 2, das colunas 5 e 12, respectivamente, do microsserviço 3.

Microsserviço	Combinação	Score	Custo	Provedor	Orçamentos para uma aplicação												
					0	1	2	3	4	5	6	7	8	9	10	11	12
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0.6	2	P1	0	0	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	2	0.7	3	P2	0	0	0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	3	0.8	4	P3	0	0	0	0	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
2	1	0.5	2	P5	0	0	0.6	0.7	1.1	0.8	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	2	0.7	4	P4	0	0	0.6	0.7	0.8	0.8	1.3	1.4	1.5	0.8	1.5	1.5	1.5
	3	0.8	7	P3	0	0	0.6	0.7	0.8	0.8	0.8	0.8	0.8	1.4	1.5	1.6	1.6
3	1	0.6	1	P2	0	0.6	0.6	1.2	1.3	1.7	1.4	1.9	2.0	2.1	2.1	2.1	2.2
	2	0.7	3	P5	0	0	0.6	0.7	1.1	1.3	1.4	1.8	1.5	2.0	2.1	2.2	2.1
	3	0.8	8	P1	0	0	0.6	0.7	1.1	0.8	1.3	1.4	1.5	1.5	1.5	1.6	1.9

FIGURA 44. PREENCHIMENTO DA MATRIZ DE COMBINAÇÕES PARA O MODELO LM^2K.

O processo de seleção das combinações é similar para o modelo $UM2K$, mas como os valores finais não são os mesmos, as combinações selecionadas são diferentes, como mostra a Figura 45. Primeiro, verifica-se qual combinação do último microsserviço, o microsserviço 3 no exemplo, na última coluna tem o maior *score*, no exemplo a combinação 1. Depois seleciona-se a combinação para o microsserviço 2, para isto, verifica-se o custo da combinação 1 selecionada para o microsserviço 3, assim, a coluna analisada para o microsserviço 2 é a coluna com valor 11 (12 - 1). A combinação com maior *score* para o microsserviço 2 na coluna 11 é a combinação 3. Como o custo da combinação 3 do microsserviço 2 é 7, a coluna analisada para o microsserviço 1 é a 4 (11 - 7). A combinação com maior *score* na coluna 4 para o microsserviço 1 é a 3. Assim, o resultado é diferente do exemplo para a solução dinâmica para o modelo UM^2K mostrado na Figura 31. Desta forma, pode-se notar a influência do *link* de comunicação entre os provedores na escolha das combinações para hospedar os microsserviços de uma aplicação.

Microsserviço	Combinação	Score	Custo	Provedor	Orçamentos para uma aplicação												
					0	1	2	3	4	5	6	7	8	9	10	11	12
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0.6	2	P1	0	0	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	2	0.7	3	P2	0	0	0	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	3	0.8	4	P3	0	0	0	0	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
2	1	0.5	2	P5	0	0	0.6	0.7	1.1	0.8	1.3	1.3	1.3	1.3	1.3	1.3	1.3
	2	0.7	4	P4	0	0	0.6	0.7	0.8	0.8	1.3	1.4	1.5	0.8	1.5	1.5	1.5
	3	0.8	7	P3	0	0	0.6	0.7	0.8	0.8	0.8	0.8	0.8	1.4	1.5	1.6	1.6
3	1	0.6	1	P2	0	0.6	0.6	1.2	1.3	1.7	1.4	1.9	2.0	2.1	2.1	2.1	2.2
	2	0.7	3	P5	0	0	0.6	0.7	1.1	1.3	1.4	1.8	1.5	2.0	2.1	2.2	2.1
	3	0.8	8	P1	0	0	0.6	0.7	1.1	0.8	1.3	1.4	1.5	1.5	1.5	1.6	1.9

FIGURA 45. COMBINAÇÕES SELECIONADAS PARA O MODELO LM^2K USANDO ALGORITMO DINÂMICO.

5.3.1.2 Diagrama Dinâmico para LM^2K

Nesta subseção, um diagrama é apresentado na Figura 46 para a solução dinâmica do modelo LM^2K. O diagrama possui como entrada um conjunto das capacidades dos provedores de nuvem disponíveis, um conjunto dos requisitos dos microsserviços de uma aplicação, um conjunto de *links* de comunicação para os serviços de cada um dos provedores e para os provedores disponíveis, e um conjunto de fluxo de execução para as atividades de cada microsserviço e para os microsserviços de uma aplicação. Os *links* de comunicação contém informações como: atraso, custo e disponibilidade.

Nota-se que o diagrama é composto por três partes, cada parte representa um nível da solução usando algoritmo dinâmico para o modelo LM^2K. O primeiro nível contém a descoberta dos serviços candidatos para cada microsserviço de uma aplicação, bem como a classificação dos serviços baseado no *score* calculado de acordo com os requisitos de um arquiteto de *software*, como no modelo UM^2K. O segundo nível combina os serviços candidatos em cada provedor, onde cada combinação deve conter todos os serviços de nuvem necessários para um microsserviço. Além disso, o *score* e o custo são calculados para cada combinação construída. Diferentemente do modelo UM^2K, para este modelo, o *score* e o custo dos *links* entre os serviços de nuvem em um mesmo provedor devem ser adicionados ao *score* e ao custo de uma combinação respectivamente. No terceiro nível, uma matriz de solução é construída, como descrito no exemplo da subseção anterior, no qual o preenchimento da matriz deve considerar o *link* de comunicação entre os provedores envolvidos, ou seja, entre os provedores aos quais pertencem as combinações. Ao final, a solução proposta deve retornar um conjunto de combinações contendo uma combinação para cada microsserviço de uma aplicação.

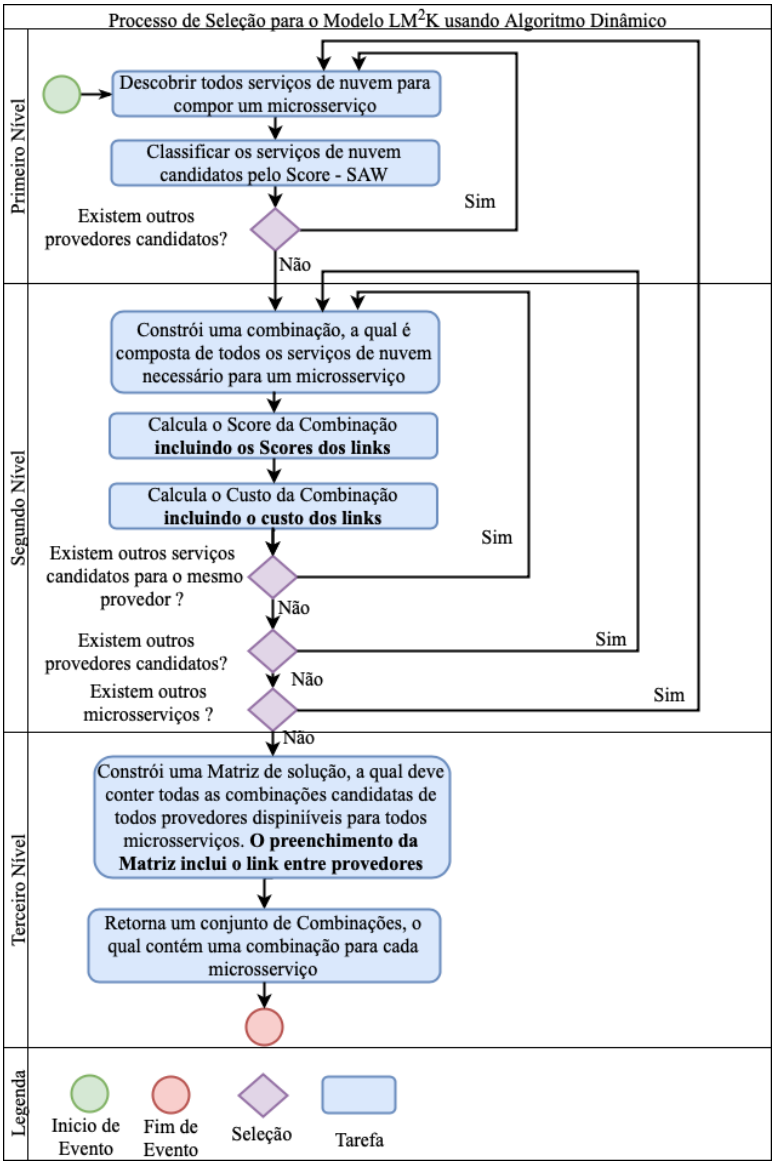


FIGURA 46. DIAGRAMA PARA O MODELO LM^2K USANDO ALGORITMO DINÂMICO.

5.3.2 Solução Gulosa para LM^2K

Nesta subseção, a solução gulosa é descrita para o modelo LM^2K. Primeiro, apresenta-se um exemplo, depois, um diagrama que contém uma visão geral. No apêndice B na Seção B descreve-se o algoritmo guloso.

5.3.2.1 Exemplo Guloso para LM^2K

O exemplo para o modelo LM^2K é baseado no exemplo simplificado descrito no início da Seção 5.3. Assim, a Figura 47a mostra o conjunto de todas as combinações candidatas para todos os microsserviços de uma aplicação, o qual possui 4 combinações candidatas para

cada um dos 3 microsserviços. Além disso, a Figura 47a apresenta para cada combinação candidata o *score*, o custo e o provedor ao qual pertence. As combinações candidatas de cada microsserviço devem ser ordenadas pelo custo da combinação.

Microsserviço	Combinação	Score	Custo	Provedor
1	1	0.4	1	P1
	2	0.3	3	P2
	3	0.9	4	P3
	4	1.0	6	P4
2	1	0.4	3	P5
	2	0.7	4	P4
	3	0.6	7	P2
	4	1.2	8	P3
3	1	0.3	1	P2
	2	0.7	3	P5
	3	0.9	7	P1
	4	0.6	8	P4

(a) Combinações Candidatas.

Microsserviço	Combinação	Score	Custo	Provedor
1	1	0.4	1	P1
	2	0.3	3	P2
	3	0.9	4	P3
	4	1.0	6	P4
2	1	0.4	3	P5
	2	0.7	4	P4
	3	0.6	7	P2
	4	1.2	8	P3
3	1	0.3	1	P2
	2	0.7	3	P5
	3	0.9	7	P1
	4	0.6	8	P4

(b) Exclusão Combinações Candidatas.

Microsserviço	Combinação	Score	Custo	Provedor
1	1	0.4	1	P1
	3	0.9	4	P3
	4	1.0	6	P4
2	1	0.4	3	P5
	2	0.7	4	P4
	4	1.2	8	P3
3	1	0.3	1	P2
	2	0.7	3	P5
	3	0.9	7	P1

(c) Combinações candidatas aptas para seleção.

FIGURA 47. EXEMPLO GULOSO PARA O MODELO LM^2K - PARTE1.

Como no exemplo da Figura 33b, a Figura 47b mostra a exclusão de combinações candidatas de cada microsserviço que são dominadas ou lp-dominadas. Para isso, observa-se as Eqs. 5.1 e 5.2, descritas em Kellerer *et al.* (2004). A Eq. 5.1 exclui as combinações candidatas dominadas, ou seja, exclui a combinação candidata `j`, cujo o custo ($cost_j$) é maior do que o custo da combinação candidata `i` ($cost_i$) e o *score* (sc_j) é menor que o *score* da combinação `i`(sc_i). A Eq. 5.2 exclui as combinações candidatas lp-dominadas, ou seja, exclui a combinação candidata `j`, cujo o custo está entre os custos de `k` e `i` ($cost_k > cost_j > cost_i$) e o *score* está entre os *scores* de `k` e `i` ($sc_k > sc_j > sc_i$), mas o custo benefício é menor. Neste exemplo, as combinações 2, 3 e 4 dos microsserviços 1, 2 e 3, respectivamente, possuem um custo maior do que a combinação anterior a ela em um mesmo microsserviço, mas possuem menor *score* do que a combinação anterior a ela em um mesmo microsserviço. A Figura 47c mostra as combinações candidatas que estão aptas para a seleção.

O próximo passo do processo de seleção é calcular a eficiência incremental para todas as combinações candidatas, exceto para as que possuem menor custo em cada um dos microsserviço, como mostra a Figura 48b. A eficiência incremental é o custo benefício de uma combinação candidata e ela é calculada usando a Eq. 5.3. Em seguida, as combinações candidatas devem ser colocadas em ordem decrescente observando a eficiência incremental, exceto para as combinações candidatas com menor custo para cada microsserviço, como mostra a Figura **??**.

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
1	1	0.4	1	P1	
	3	0.9	4	P3	0.4/3
	4	1.0	6	P4	0.3/2
2	1	0.4	3	P5	
	2	0.7	4	P4	0.3/1
	4	1.2	8	P3	0.5/4
3	1	0.3	1	P2	
	2	0.7	3	P5	0.4/2
	3	0.9	7	P1	0.2/4

(a) Calculo do Coeficiente Incremental.

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
2	2	0.7	4	P4	0.3/1
3	2	0.7	3	P5	0.4/2
1	3	0.9	4	P3	0.4/3
1	4	1.0	6	P4	0.3/2
2	4	1.2	8	P3	0.5/4
3	3	0.9	7	P1	0.2/4

(b) Ordenação pelo Coeficiente Incremental.

FIGURA 48. EXEMPLO GULOSO PARA O MODELO LM^2K - PARTE2.

As combinações com menor custo de cada microsserviço compõem a solução inicial e os seus respectivos custos são decrementados do orçamento da aplicação. No exemplo, as cominações 1 de cada microsserviços compõem a solução inicial e o limiar do orçamento para a aplicação é 12 e os custos das combinações de menor custo são 1, 3, e 1, respectivamente. Depois, verifica-se a possibilidade de substituir cada combinação na solução por uma que está no conjunto ordenado pela eficiência incremental da Figura **??**. Para realizar a substituição devem ser observados os valores dos requisitos para os *links* de comunicação entre provedores. Neste exemplo, os valores dos requisitos para os *links* de comunicação foram supostos. Assim, na Figura 49a substituiu-se a combinação 1 pela 2, tanto no microsserviço 2 quanto no microsserviço 3. No entanto, quando a combinação 1 é substituída pela 2 no microsserviço 1, os valores dos *links* de comunicação não atendem às necessidades da aplicação; então, a combinação 2 do microsserviço 1 deve ser removida. Depois, o processo de substituição deve ser continuado, mas não é possível, pois o próximo valor (5ªlinha da Figura **??**) excede o orçamento. Assim, a combinação 1 é selecionada para o microsserviço 1, como mostra a Figura 49b.

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
1	1	0.4	1	P1	
	3	0.9	4	P3	0.4/3
	4	1.0	6	P4	0.3/2
2	1	0.4	3	P5	
	2	0.7	4	P4	0.3/1
	4	1.2	8	P3	0.5/4
3	1	0.3	1	P2	
	2	0.7	3	P5	0.4/2
	3	0.9	7	P1	0.2/4

(a) Exclusão pelo *link* de comunicação.

Microsserviço	Combinação	Score	Custo	Provedor	Eficiência Incremental
1	1	0.4	1	P1	
	4	1.0	6	P4	0.3/2
2	1	0.4	3	P5	
	2	0.7	4	P4	0.3/1
	4	1.2	8	P3	0.5/4
3	1	0.3	1	P2	
	2	0.7	3	P5	0.4/2
	3	0.9	7	P1	0.2/4

(b) Seleção das Combinações.

FIGURA 49. EXEMPLO GULOSO PARA O MODELO LM^2K - PARTE3.

5.3.2.2 Diagrama Guloso para LM^2K

Nesta subseção, na Figura 50, um diagrama é apresentado para o modelo LM^2K usando um algoritmo guloso. O diagrama contém três níveis, como descrito na subseção 4.1.2. No primeiro nível, identifica-se e classifica-se os serviços de nuvem que atendem aos requisitos dos microsserviços de uma aplicação. No segundo, constrói-se combinações que contém todos os serviços de nuvem candidatos para um microsserviço em um mesmo provedor. O processo é repetido para todos os serviços candidatos dentro de um mesmo provedor, bem como em todos os provedores disponíveis e para todos os microsserviços de uma aplicação.

Depois de construir todas as combinações candidatas para todos os microsserviços, a seleção de uma combinação deve ser realizada para cada microsserviço. Para isto, no último nível, a seleção de múltiplas combinações candidatas é mapeada para o problema da mochila de múltiplas escolhas, onde o orçamento da aplicação é a capacidade máxima da mochila e o *score* é o lucro de uma combinação candidata, a fim de obter as combinações com maior *score* que contemplam o orçamento. Para isto, as combinações candidatas para cada microsserviço são ordenadas pelo custo. Depois, as combinações que estão de acordo com as Eqs, 5.1 e 5.2 são excluídas. Em seguida, a eficiência incremental é calculada para cada combinação e as combinações são colocadas em ordem decrescente de acordo com a eficiência incremental. Por fim, as combinações são selecionadas observando o orçamento da aplicação, a eficiência incremental e os *links* de comunicação.

5.3.3 Solução Bioinspirada usando Colônia de Formigas para LM^2K

Esta solução é baseada em um algoritmo bioinspirado em colônia de formigas, o qual pertence à classe de estratégias de resolução de problemas derivadas da natureza. De acordo com Goldbarg *et al.* (2016), o algoritmo em colônia de formigas é basicamente um sistema multiagente, onde existe um baixo nível de interações entre os agentes, ou seja, formigas artificiais, que resultam em um comportamento complexo de uma colônia de formigas. A ideia básica do algoritmo em colônia de formigas é modelar o problema em consideração como um problema de busca, onde um caminho de custo mínimo é pesquisado. As formigas artificiais são empregadas para procurar bons caminhos. O feromônio é um tipo de informação distribuída que é modificada pelas formigas para refletir sua experiência acumulada durante a resolução de problemas. Essa substância influencia as escolhas que eles fazem: quanto maior a quantidade de

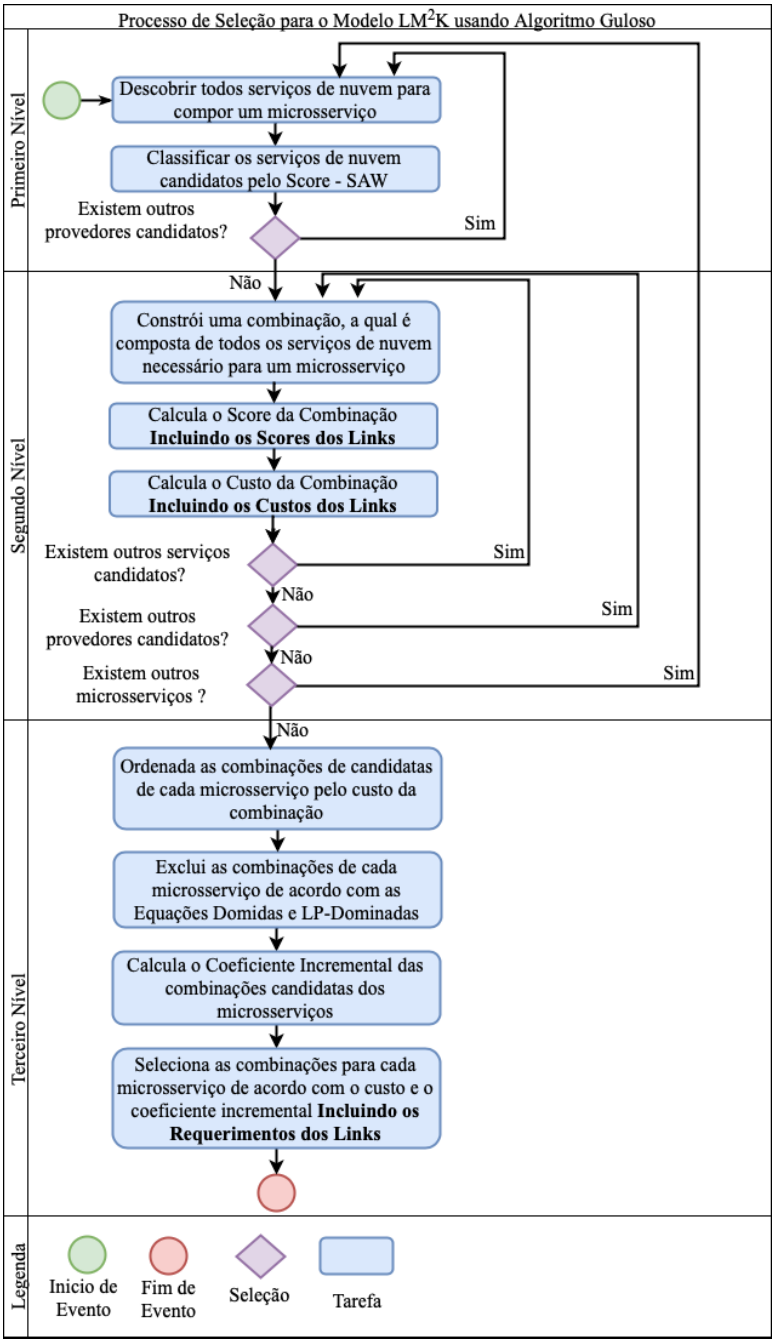


FIGURA 50. DIAGRAMA PARA O MODELO LM^2K USANDO ALGORITMO GULOSO.

feromônio em um determinado caminho, maior é a probabilidade de uma formiga selecionar o caminho. Intuitivamente, esse meio indireto de comunicação visa fornecer informações sobre a qualidade dos componentes do caminho a fim de atrair formigas. A seguir, apresenta-se um exemplo, um diagrama que refletem a solução proposta e um algoritmo na Seção B no Apêndice B.

5.3.3.1 *Exemplo Bioinspirado em Colônia de Formigas para LM^2K*

Para o modelo LM^2K, cada microsserviço representa um grupo de itens e cada combinação candidata ($Comb_{ij}$) é um item `i` do grupo `j`. A Figura 51 mostra o conjunto de todas as combinações candidatas para todos os microsserviços de uma aplicação, o qual possui i_j combinações candidatas para cada microsserviço `j`. Para cada combinação candidata da Figura 51 deve ser ter o *score*, o custo, o provedor ao qual pertence e o feromônio inicial. As combinações candidatas de cada microsserviço devem ser ordenadas pelo custo da combinação. O processo possui `n` formigas e cada uma delas deve construir uma solução individual. Em cada solução, considera-se o tempo de resposta, a disponibilidade e o custo para os *links* de comunicação entre os provedores de cada combinação candidata selecionada por uma formiga. O processo é repetido `m` ciclos, ou seja, cada formiga deve construir `m` soluções. De forma que cada formiga constrói uma solução em cada ciclo. A solução com maior lucro em cada ciclo é selecionada e ao final de todos os ciclos a melhor solução é selecionada.

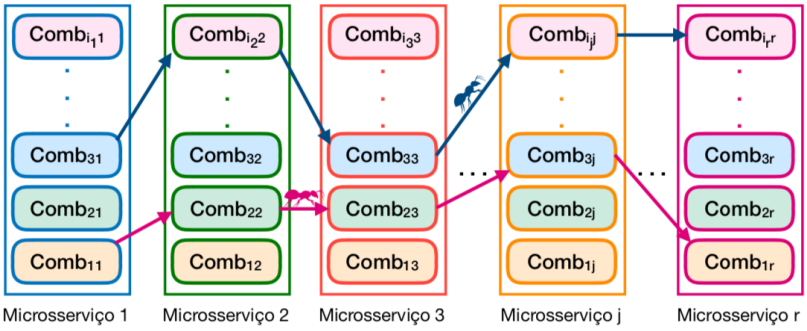


FIGURA 51. EXEMPLO PARA O MODELO LM^2K USANDO COLÔNIA DE FORMIGAS.

5.3.3.2 *Diagrama Bioinspirado em Colônia de Formigas para LM^2K*

Nesta subseção, na Figura 52 um diagrama usando um algoritmo bioinspirado em colônia de formigas é apresentado para o modelo LM^2K. O diagrama contém três níveis, como descrito para o modelo na subseção 4.1.2. Como para as outras soluções deste modelo, no primeiro nível, descobre-se e classifica-se os serviços de nuvem que atendem aos requisitos dos microsserviços de uma aplicação. No segundo nível, constrói-se combinações que contém todos os serviços de nuvem candidatos para um microsserviço em um mesmo provedor. O processo é repetido para todos os serviços candidatos dentro de um mesmo provedor, bem como em todos os provedores disponíveis e para todos os microsserviços de uma aplicação.

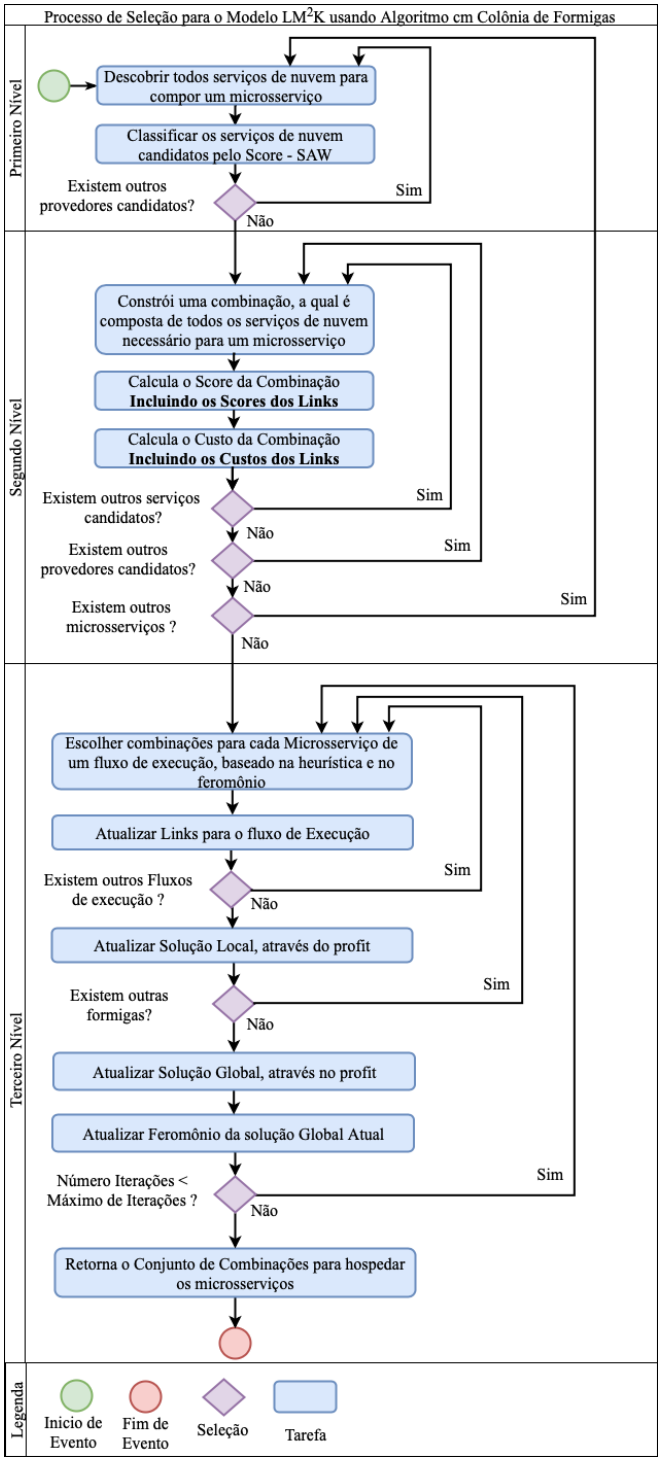


FIGURA 52. DIAGRAMA PARA O MODELO LM^2K USANDO COLÔNIA DE FORMIGAS.

No terceiro nível, o processo de seleção inicia com um grupo com n formigas e cada uma delas constrói individualmente uma solução. Uma formiga começa com uma mochila vazia. Em seguida, ela seleciona aleatoriamente uma combinação candidata do primeiro microsserviço pertencente ao fluxo de execução de uma aplicação. Depois, ela atualiza o feromônio local e depois seleciona outra combinação para o próximo microsserviço do fluxo de execução de

acordo a quantidade de feromônio depositada na combinação até o momento da seleção e de seu valor heurístico local. O processo é repetido até que seja selecionada uma combinação candidata para cada um dos microsserviços do fluxo de execução. Assim, uma solução local é construída por uma formiga, o processo é repetido para todas as formigas. Cada formiga, no processo de construção de uma solução coloca feromônio no caminho que ela percorre para que aquelas combinações possam ter uma maior atratividade para outras formigas.

Esta solução considera os requisitos para os *links* de comunicação entre provedores, ou seja, o caminho que a formiga percorre de uma combinação até a próxima. A solução que obtiver maior lucro é selecionada no primeiro ciclo. O processo é repetido para todas as formigas `m` vezes, e ao final de cada ciclo o feromônio é atualizado.

5.4 Resumo do Capítulo

Neste capítulo, exemplos, diagramas e algoritmos são apresentados e descritos para as soluções propostas para os três modelos propostos no Capítulo 4 para a seleção de múltiplos provedores de nuvem para hospedar microsserviços de uma aplicação com perspectiva de um arquiteto de *software*, sendo eles: UM^2K, UM^2Q e LM^2K. Para o modelo UM^2K, propôs-se duas soluções: uma usando algoritmo dinâmico; e a outra, algoritmo guloso. Uma solução foi proposta para o modelo UM^2Q, a qual é baseada em requisitos independentes para cada microsserviço de uma aplicação, ou seja, os requisitos de um microsserviço não interfere nos requisitos de outro microsserviço de uma mesma aplicação. Para o modelo LM^2K, três soluções foram propostas, a primeira delas baseada em algoritmo dinâmico; a segunda, em um algoritmo guloso e a última em um algoritmo bioinspirado em colônia de formigas.

6 EXPERIMENTOS E RESULTADOS

Nos Capítulos 4 e 5, os modelos e as soluções foram descritas para a seleção de múltiplos provedores de nuvem da perspectiva de um arquiteto de *software*, para hospedar uma aplicação baseada em microsserviços, os quais devem ser utilizados pelo PacificClouds. Cada microsserviço de uma aplicação deve ser hospedado em um único provedor, o que melhor atenda aos requisitos de um microsserviço. A fim de avaliar o desempenho das soluções propostas para os modelos (UM^2K, UM^2Q e LM^2K), este capítulo descreve as ferramentas implementadas, os cenários e os experimentos. Além disso, ele compara os resultados obtidos dos experimentos.

6.1 Descrição das Ferramentas

Uma ferramenta foi desenvolvida para cada uma das soluções propostas para os três modelos. O Python 3.7 foi usado para implementar as ferramentas e o formato JSON para os dados de entrada e saída. As ferramentas desenvolvidas para os modelos UM^2K, UM^2Q possuem como entrada: um arquivo contendo todos os requisitos de uma aplicação baseada em microsserviços e um arquivo contendo todas as capacidades de todos os provedores de nuvem disponíveis. Além destes arquivos, as ferramentas para o modelo LM^2K também possuem como entrada um arquivo com os fluxos de execução entre os microsserviços de uma aplicação e entre as atividades de cada microsserviço, e um arquivo com os dados de interações entre os provedores de nuvem disponíveis. Todas as ferramentas possuem como saída um arquivo JSON contendo informações sobre os provedores e serviços que foram selecionados para hospedar cada um dos microsserviços de uma aplicação.

De acordo com descrito na seção 4.1, um arquiteto de *software* deve definir os limiares para os requisitos de uma aplicação. Assim, o arquivo com os dados de uma aplicação deve conter: os limiares para o orçamento, disponibilidade, tempo de resposta e o peso para cada um deles. Além dessas informações, ele deve conter, para cada microsserviço, os requisitos para os serviços de nuvem necessários para executar suas atividades. A ferramenta para o modelo UM^2Q deve conter também uma cota de orçamento para cada microsserviço, além do orçamento total. Um exemplo de arquivo com os requisitos para uma aplicação é ilustrado pelas Figuras 53 e 54.

A Figura 53a ilustra os dados de uma aplicação a qual possui 6 microsserviços, e cada deles têm 3 informações. A Figura 53b mostra que as informações sobre os microsserviços

(a) (b)

FIGURA 53. EXEMPLO JSON PARA REQUISITOS DE APLICAÇÕES - PARTE 1.

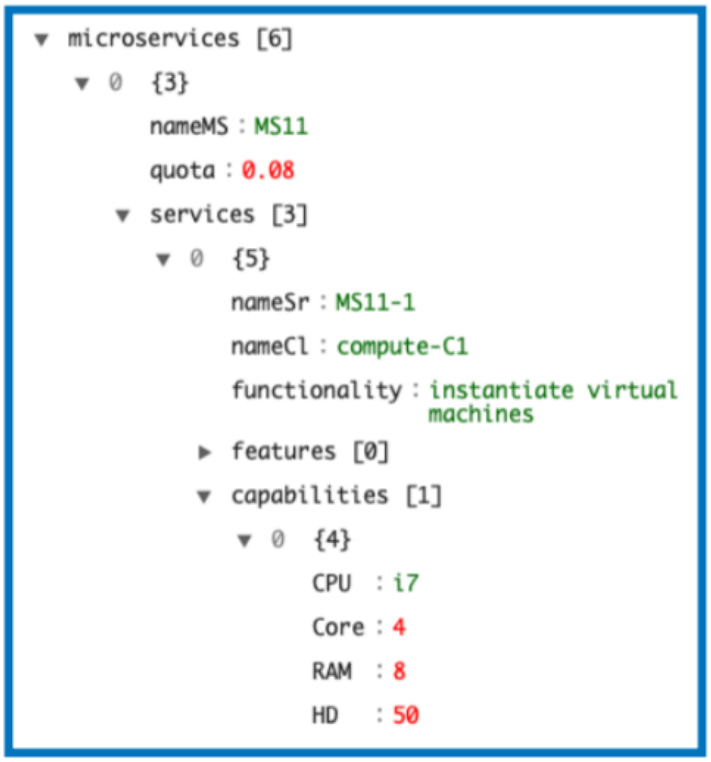


FIGURA 54. EXEMPLO JSON PARA REQUISITOS DE APLICAÇÕES - PARTE2.

são nome, quota e os requisitos necessários para os serviços de nuvem. A Figura 54 detalha informações dos serviços de nuvem, tais como: classe de serviço, funcionalidade, características e capacidades. Como o serviço de nuvem do exemplo é da classe computação, as capacidades apresentadas são em relação às capacidades de uma máquina virtual.

As informações referentes às capacidades dos provedores de nuvem disponíveis consideradas em nesta tese são: classe do serviço, custo, disponibilidade, tempo de execução,

além de informações específicas do serviço. Um exemplo de arquivo JSON com capacidades de um provedor é apresentado na Figura 55. Pode-se notar na Figura 55a que a classe de serviços nomeada de compute-C1 possui 6 serviços e cada um dos serviços possuem 5 informações as quais são exemplificadas na Figura 55a. Na Figura 55a é possível identificar as capacidades funcionais do serviço (`capabilities`) e as capacidade não funcionais (`userRequirements`).

(a) (b)

FIGURA 55. EXEMPLO JSON PARA CAPACIDADES DE PROVEDORES.

A Figura 56, mostra o exemplo das informações de interações entre provedores, através das quais é possível obter as capacidades dos *links* de comunicação. Neste exemplo, a interação 0 significa a interação entre os provedores P0 e P1. A capacidade do *link* de comunicação para a disponibilidade é de 100, para o atraso é 2 e para o custo é 5. O arquivo JSON deve conter todas as interações entre os provedores de nuvem disponíveis.

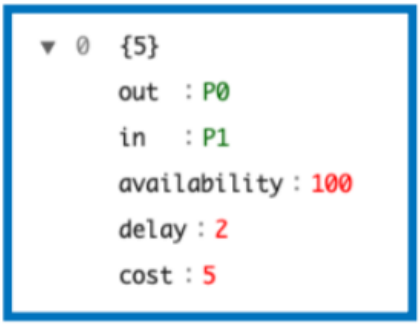


FIGURA 56. EXEMPLO JSON PARA INTERAÇÕES ENTRE PROVEDORES.

As Figuras 57a e 57b mostram interações entre os microsserviços, ou seja, o fluxo de execução da aplicação. E as Figuras 57c e 57d mostram interações entre os serviços de nuvem necessários a um microsserviço, o qual representa o fluxo de execução dos serviços para um microsserviço. Na Figura 57a, pode-se observar que a aplicação do exemplo possui dois termos (`iterationsAPP[2]`), cada um dos quais é composto por dois fluxos (`numOfFlow`) e cada fluxo tem duas sequência de microsserviços (`sequences[2]`). A Figura 57b detalha uma das sequências, para cada sequência tem-se o nome, a probabilidade de a sequência ser selecionada e os microsserviços que dela fazem parte, apresentado os *links* de entrada e saída para cada um deles. A Figura 57c apresenta os termos e os fluxos que devem existir entre os serviços de nuvem utilizados para executar um microsserviço. A Figura 57d detalha uma sequência de serviços para um fluxo de um microsserviço, a qual tem dois serviços e, para cada um deles, os *links* de entrada e saída.

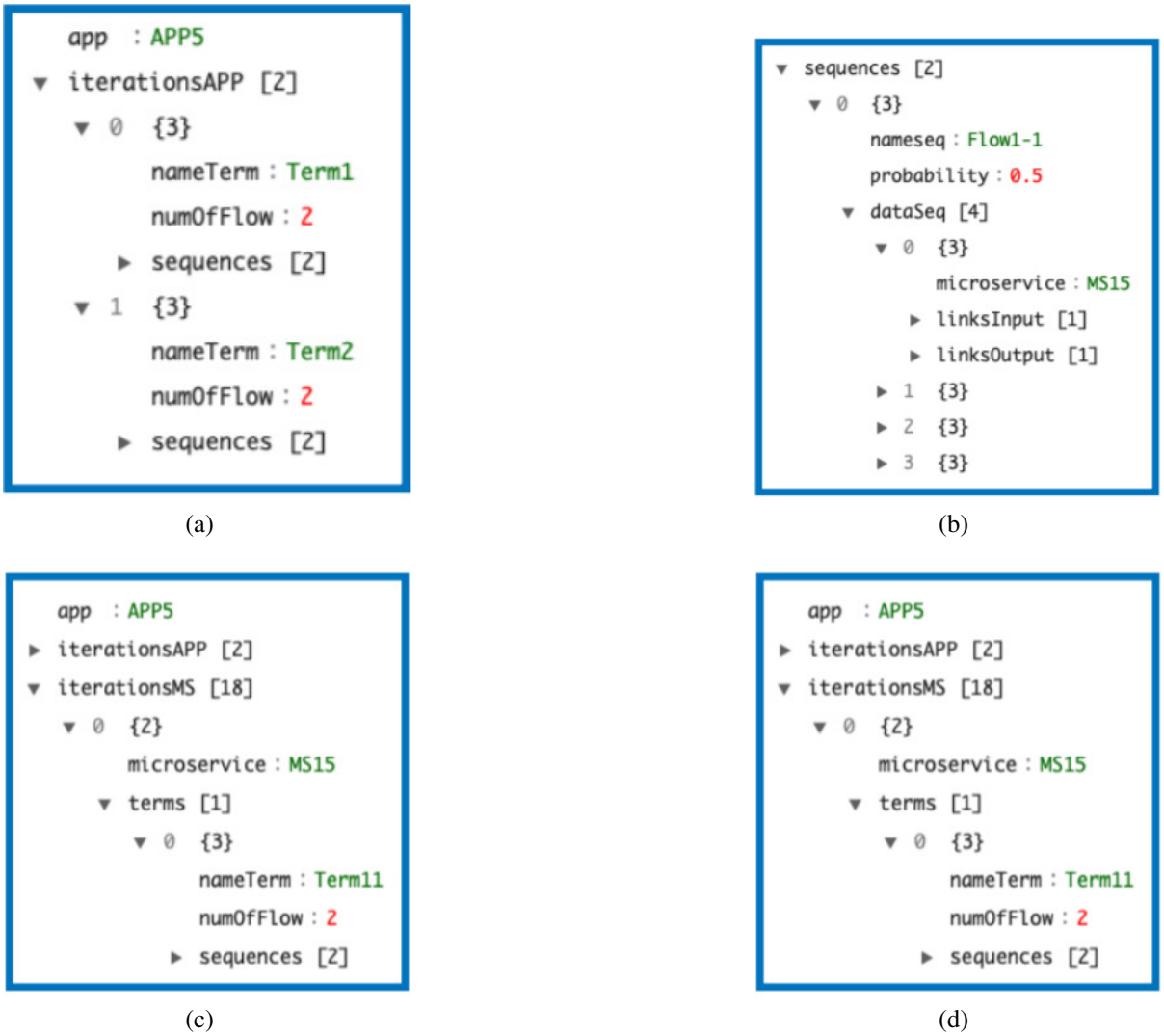


FIGURA 57. EXEMPLO JSON PARA DE ITERAÇÕES ENTRE ATIVIDADES E MICROSSERVIÇOS.

Além dos parâmetros descritos aqui, cada solução possui suas características, as quais foram descritas no Capítulo 5. Assim, as soluções dinâmicas são baseadas em tabelas, as quais crescem de acordo com o limiar do orçamento e com o número de combinações candidatas que atendem os microsserviços. As tabelas devem ser preenchidas, e então, percorridas para encontrar o melhor resultado. Enquanto que as soluções gulosas e a baseada em colônia de formigas baseiam-se em listas, as quais não necessariamente precisam ser totalmente analisadas. Assim, as soluções dinâmicas possuem uma limitação em relação aos parâmetros de entrada.

6.2 Descrição dos Experimentos

Esta seção descreve os cenários, os experimentos e os resultados da avaliação de desempenho de todas as soluções para os três modelos propostos. Cada experimento foi realizado 30 vezes, a fim de alcançar a distribuição normal, de acordo com Hogendijk e Whiteside (2011). Os cenários foram configurados sinteticamente em conformidade com a maioria das pesquisas na área, pois de acordo com Hayyolalam e Pourhaji Kazem (2018), há poucos conjuntos de dados disponíveis no domínio de pesquisa da computação em nuvem. Além disso, em todos os cenários os provedores foram organizados por conjunto, onde os conjuntos se diferem pela quantidade de provedores ou pela quantidade de serviços por provedores. As aplicações são organizadas por microsserviços e elas se diferem pela quantidade de microsserviços. Neste trabalho foi considerado que todos os microsserviços são do mesmo tamanho, ou seja, necessitam da mesma quantidade de serviços de nuvem, os quais se diferenciam pelos requisitos. Os experimentos foram realizados em um MacBook Air que é alimentado por um processador i5, 4 GB de RAM e SSD de 256 GB.

Observando que o orçamento de uma aplicação é o valor máximo disponível para contratar todos os serviços de nuvem necessários para uma aplicação. Além disso, que o valor do custo de uma aplicação implantada em múltiplos provedores de nuvem é a soma de todos os valores de custo dos serviços de nuvem usados pela aplicação. Desta forma, também foram configuradas classes de orçamento, onde cada classe tem um valor de orçamento distinto por aplicação, mas o valor do serviço é o mesmo em todas as aplicações. Além disso, os valores dos serviços são definidos com base nos valores de custo mínimo até o máximo de serviços de nuvem descritos para as capacidades dos provedores. Os requisitos de disponibilidade e tempo de resposta foram definidos com valores que podem ser atendidos pela maioria dos provedores. Por exemplo, se o custo de um serviço for estipulado como 10 e cada microsserviço necessitar de 3

serviços, o custo de um microsserviço será 30, e se a aplicação tiver 6 microsserviços o orçamento para esta aplicação será 180 e, se outra aplicação possuir 9 microsserviços o orçamento desta será 270, e elas estão na mesma classe de orçamento. Além da classe de orçamentos também foram configuradas as classes de requisitos, onde cada classe possui valores diferentes para cada um dos requisitos. As classes foram configuradas, de forma que, quando o requisito de disponibilidade aumenta, os requisitos de tempo de resposta e de custo diminuem. Portanto, quanto maior o valor de uma classe de requisitos maior é o grau de restrição.

Os resultados de todos os experimentos são apresentados em gráficos, nos quais o tempo médio de seleção de provedores em milissegundos (ms) é apresentado no eixo y, as linhas representam cada aplicação, e o eixo x mostra cada um dos experimentos.

6.2.1 Experimentos para a Solução Dinâmica de UM^2K

Esta seção tem como objetivo avaliar a solução baseada em programação dinâmica para o modelo UM^2K e, consequentemente, avaliar a ferramenta desenvolvida para esta solução. Para isto, um cenário foi configurado contendo os conjuntos de provedores e as aplicações, o qual é ilustrado nas Tabelas 6 e 7. Na Tabela 6, pose-se notar que foram configurados sinteticamente 5 conjuntos de provedores, que diferem entre si pelo número de provedores e pela capacidade de cada serviço do provedor. Para isso, primeiro, um conjunto com 5 provedores foi configurado. Em seguida, um outro conjunto de provedores foi configurado como descrito na Eq. 6.1. Na Eq. 6.1, $SPrvds_x$ é um conjunto de provedores, em que x é o número de provedores em SPrvds e $SPrvds_{x-5}$ é o maior conjunto de provedores configurado até aquele momento e x deve ser maior ou igual a 5. Se x for igual a 5, indica que é o primeiro conjunto do cenário. Se x for igual a 15, por exemplo, indica que será gerado um conjunto de 15 provedores, sendo que os 10 primeiros são iguais aos previamente configurados e os outros 5 são novos. Assim, a quantidade de provedores em cada conjunto é um valor múltiplo de 5, no qual o primeiro conjunto tem 10 provedores e o último tem 30. Cada provedor possui três classes de serviços: computação, armazenamento e banco de dados, e cada classe tem seis serviços. Logo, cada provedor possui 18 serviços.

$$SPrvds_x = SPrvds_{(x-5)} + SPrvds_5 \tag{6.1}$$

Na Tabela 7, pode-se observar que foram configuradas 4 aplicações: a primeira

(APP1) possui 4 microsserviços, a segunda (APP2) possui 5, a terceira (APP3) possui 6 e a última (APP4) possui 7 microsserviços. Os requisitos de disponibilidade, tempo de resposta e custo variam dependendo do tipo de avaliação.

TABELA 6. CONFIGURAÇÃO DOS CONJUNTOS DE PROVEDORES PARA A SOLUÇÃO DINÂMICA DO UM^2K

Solução	Características dos Conjuntos				
	Número de Conjuntos	Número de Provedores por Conjunto	Número de Serviços por Classe	Número de Serviços por Provedor	Número de Serviços por Conjunto
Dinâmica UM^2K	5	10	6	18	180
		15			270
		20			360
		25			450
		30			540

TABELA 7. CONFIGURAÇÃO DAS APLICAÇÕES PARA A SOLUÇÃO DINÂMICA DO UM^2K

Características	Aplicações			
	APP1	APP2	APP3	APP4
Número de Microsserviços	4	5	6	7
Número de Serviços na Aplicação	12	15	18	21

Com o conjunto de provedores e de aplicações configurados, o desempenho da seleção dinâmica para o modelo UM^2K é avaliado através de cinco experimentos. Os quatro primeiros experimentos realizados usaram um conjunto de 10 provedores descrito na primeira linha da Tabela 6. A Figura 58 mostra o primeiro experimento foi realizado variando o número de provedores por conjunto. No gráfico da Figura 58 o eixo x representa o número de provedores por conjunto e o eixo y o tempo médio de seleção e as linhas representam cada uma das aplicações. Para isto, foram utilizados os 5 conjuntos de provedores apresentados na Tabela 6. Os requisitos de disponibilidade, tempo de resposta e custo não foram modificados durante o experimento, mantendo o mesmo valor em todos os conjuntos. Esses requisitos foram configurados usando os valores intermediários, ou seja, eles não possuem a restrição mais baixa nem a mais alta. Na Figura 58, é possível ver que o aumento no número de provedores influencia o tempo médio de seleção, uma vez que aumenta o número de serviços que atendem aos requisitos das aplicações.

Os outros 4 experimentos são ilustrados na Figura 59, a qual possui um gráfico para cada um dos experimentos. No experimento da Figura 59a 5 valores diferentes para disponibilidade, sendo o primeiro valor 90% e o último 98%, pois através destes valores é possível identificar a variação no tempo médio de seleção. Figura 59a nota-se que o tempo médio de seleção diminui com o aumento do valor do requisito de disponibilidade em cada aplicação. Para experimento da Figura 59b cada aplicação foi configurada com 5 valores de

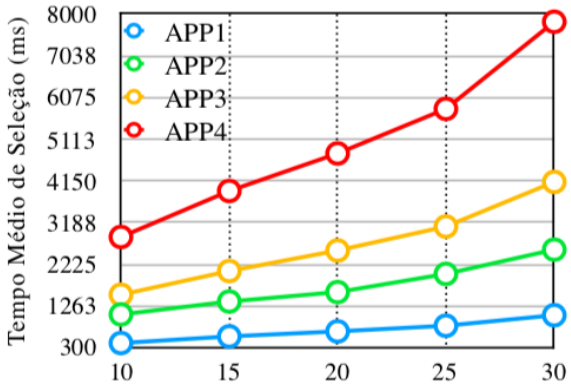
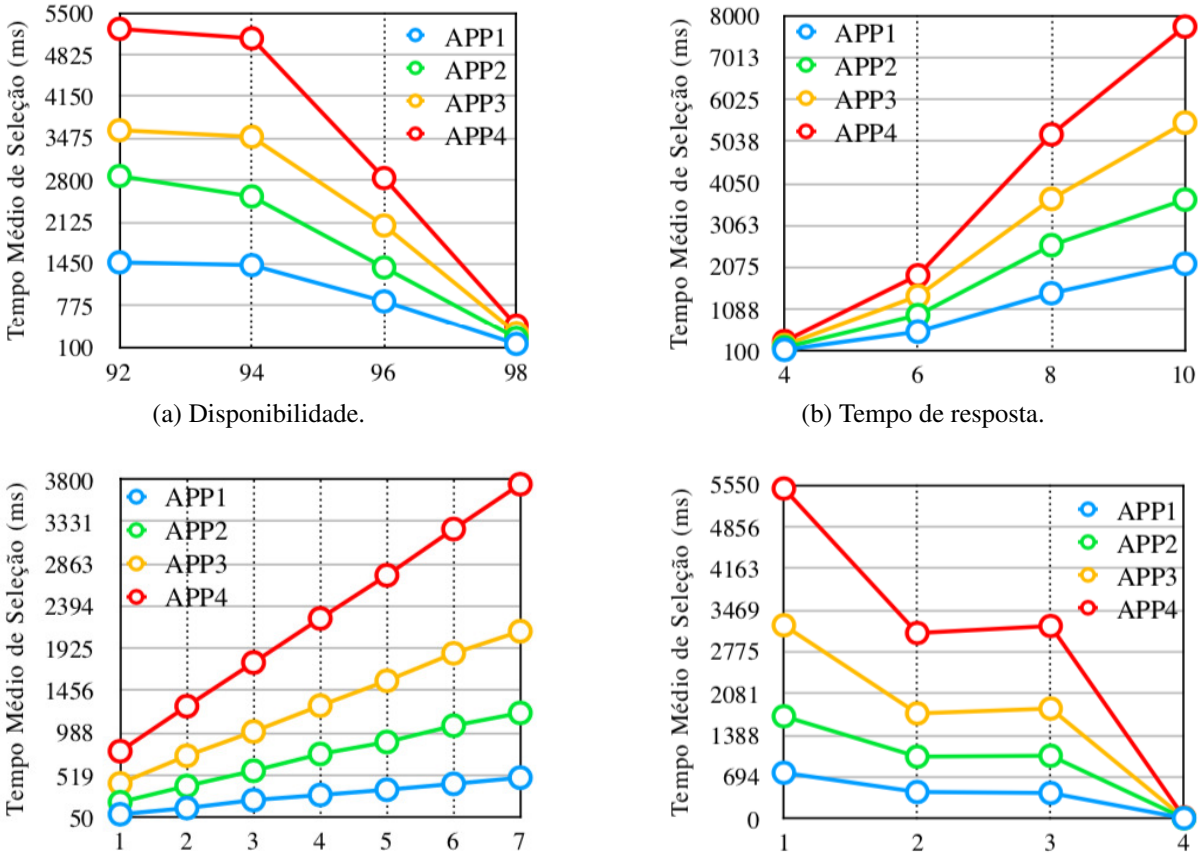


Figura 58. Experimento DUM^2K – Conjunto de Provedores.

tempo de resposta distintos. Esses valores indicam que apenas os serviços de provedores com valores menores ou iguais aos solicitados atendem ao requisito de tempo de resposta de uma aplicação. Os requisitos de disponibilidade e orçamento foram definidos com valores que podem ser atendidos pela maioria dos provedores. Os resultados da Figura 59b representam o esperado, porque o gráfico mostra que o tempo médio de seleção aumenta com o aumento do requisito de tempo de resposta em cada aplicação. A Figura 59c apresenta o resultado do experimento apresenta o resultado do experimento, no qual variou-se as classes de orçamentos como descrito no preâmbulo da Subseção 6.2. A Figura 59c mostra que o tempo médio de seleção aumenta de acordo com a classe de orçamento, mas depois de um certo valor, o tempo médio de seleção é estabilizado. O aumento do tempo médio acontece porque quanto maior o orçamento, maior o número de provedores que atendem ao requisito. E a estabilidade acontece porque, depois de um certo valor, o número de provedores que atendem ao requisito tende a se estabilizar, pois o valor máximo foi atingido. Por fim, a Figura 59d mostra o resultado do quando os requisitos de disponibilidade, de tempo de resposta e de custo são modificados ao mesmo tempo. De forma que, classes de requisitos foram criadas em conformidade com o descrito no início da Subseção 6.2. Pode-se observar que o tempo médio de seleção da Figura 59d diminui a medida que os valores do eixo x aumentam. Esse resultado mostra que o aumento dos requisitos reduz o número de provedores que atendem a todos os requisitos e que o tempo médio de seleção converge para um mesmo valor.

6.2.2 Experimentos para a Solução Gulosa de UM^2K

Esta seção visa avaliar o desempenho da solução gulosa para o modelo UM^2K. Para isto, cenários contendo informações dos conjuntos de provedores e das aplicações foram



(a) Disponibilidade.

(b) Tempo de resposta.

(c) Classe de Orçamentos.

(d) Classe de Requisitos.

Figura 59. Experimentos para a solução DUM^2K.

configurados e apresentados nas Tabelas 8 e 9. Observa-se que 10 conjuntos de provedores foram configurados, pois através destes é possível avaliar o desempenho da solução. Os primeiros 5 conjuntos de provedores se diferenciam pela quantidade de serviços em cada provedor, de maneira que cada conjunto têm 20 provedores, e cada provedor tem um valor múltiplo de 10 serviços por classe, sendo que o primeiro tem 10 e o último 50. Os outros 5 conjuntos de provedores diferem entre si pelo número de provedores. Cada conjunto tem um valor múltiplo de 10 provedores, o primeiro conjunto tem 20 e o último 60. Além disso, cada provedor tem 40 serviços por classe de serviço.

Na Tabela 9 são apresentadas 5 aplicações baseadas em microsserviço: a primeira (APP1) possui 6 microsserviços, a segunda (APP2) possui 9, em que seis são iguais à APP1; a terceira (APP3) tem 12, nos quais nove são iguais à APP2; a quarta (APP4) possui 15, nos quais 12 deles são iguais à APP3; e a última (APP5) possui 18 microsserviços, nos quais 15 são iguais à APP4.

Com os cenários configurados, é possível avaliar o desempenho da solução, bem

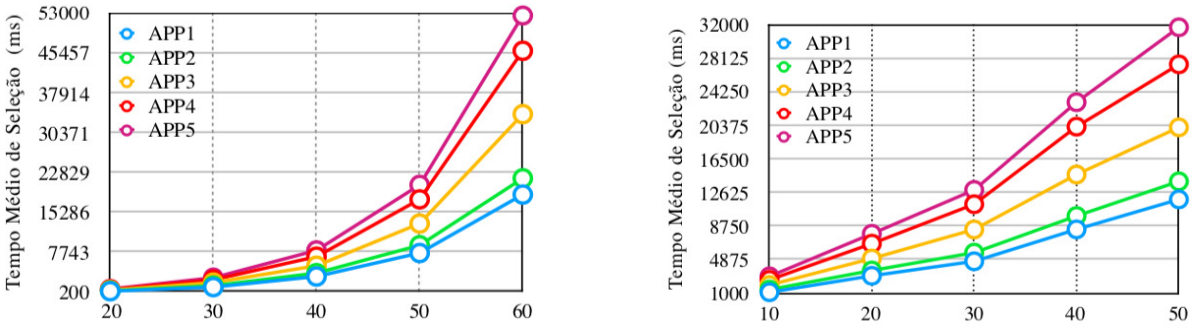
TABELA 8. CONFIGURAÇÃO DOS CONJUNTOS DE PROVEDORES PARA A SOLUÇÃO GULOSA DO UM^2K

Solução	Características dos Conjuntos				
	Número de Conjuntos	Número de Provedores por Conjunto	Número de Serviços por Classe	Número de Serviços por Provedor	Número de Serviços por Conjunto
Gulosa UM^2K	5	20	10	30	150
			20	60	300
			30	90	450
			40	120	600
			50	150	750
	5	20	40	120	2400
		30			3600
		40			4800
		50			6000
		60			7200

TABELA 9. CONFIGURAÇÃO DAS APLICAÇÕES PARA A SOLUÇÃO GULOSA DO UM^2K

Características	Aplicações				
	APP1	APP2	APP3	APP4	APP5
Número de Microsserviços	6	9	12	15	18
Número de Serviços na Aplicação	18	27	36	45	54

como o comportamento da ferramenta. Para isso, 6 experimentos foram realizados, nos quais foram usados os cenários apresentados nas Tabelas 8 e 9. Nos dois primeiros experimentos, o desempenho foi verificado, tanto pelo número de provedores em cada conjunto quanto pela quantidade de serviço por provedor, como mostrado na Figura 60. A Figura 60a mostra os conjuntos de provedores no eixo x, e a Figura 60b mostra o número de serviços do provedor no eixo x. Além disso, nos dois experimentos, os requisitos de disponibilidade, tempo de resposta e custo não foram modificados durante o experimento, mantendo o mesmo valor em todos os conjuntos. Para o experimento da Figura 60a, usou-se os últimos 5 conjuntos apresentados na Tabela 8, ou seja, os conjuntos de provedores que si diferenciam pelo número de provedores. Para o experimento da Figura 60b, foram usados os 5 primeiros conjuntos apresentados na Tabela 8, os quais se diferem pela quantidade de serviços por provedor.



(a) Quantidade de Provedores por Conjunto.

(b) Número de Serviços por Provedor.

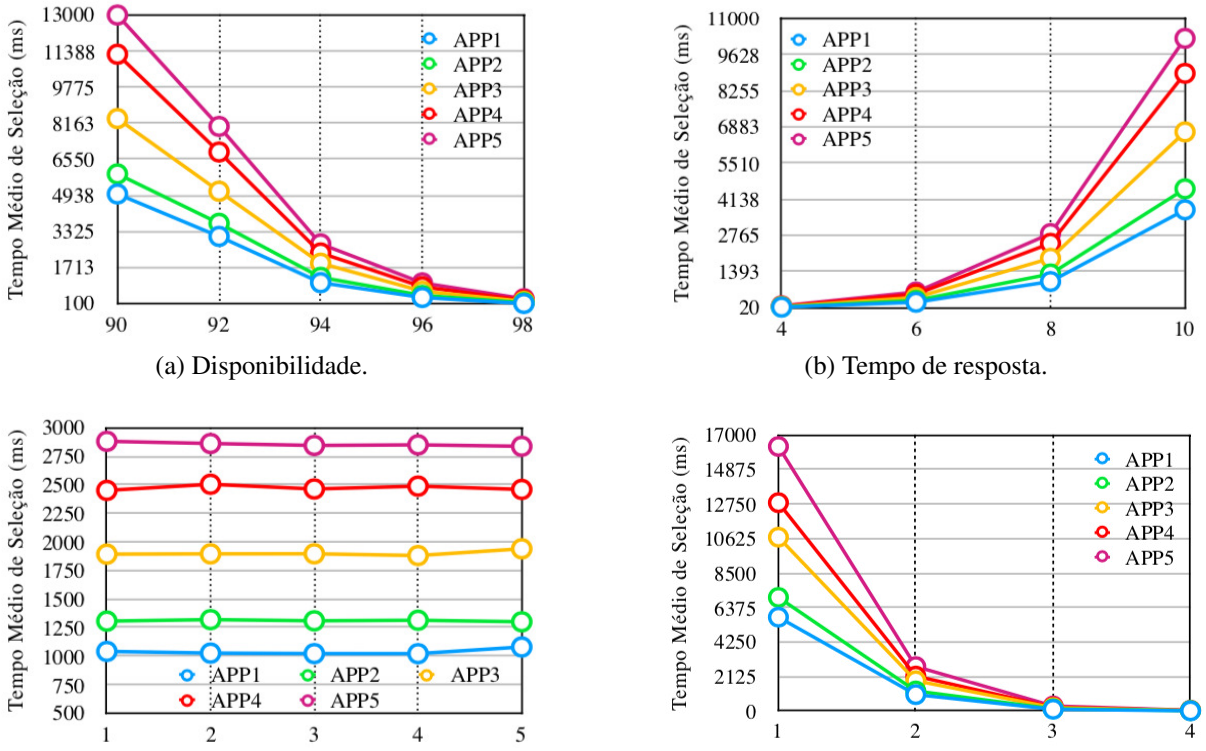
Figura 60. Experimentos relacionados a quantidade de provedores e serviços para GUM^2K.

Observar-se que o tempo médio de seleção aumenta nos dois experimentos de acordo com o aumento do número de provedores e com o aumento do número de serviços por provedores, respectivamente. Além disso, nota-se que o aumento no número de serviços por provedor tem maior influência no tempo médio de seleção que o aumento no número de provedores por conjunto.

A partir destes dois experimentos foi utilizado o sexto conjunto de provedores da Tabela 8 para os demais experimentos. O conjunto usado possui 20 provedores e cada provedor possui 120 serviços. A Figura 61 mostra os demais experimentos, a qual possui um gráfico para cada um dos experimentos. Na Figura 61a cada aplicação foi configurada com cinco valores diferentes para disponibilidade. Além disso, os requisitos de tempo de resposta e de custo foram definidos com valores que podem ser atendidos pela maioria dos provedores. Na Figura 61a é possível observar que o tempo médio de seleção diminui com o aumento do valor de disponibilidade em todas as aplicações. No experimento da Figura 61b cada aplicação foi configurada com cinco valores de tempo de resposta diferentes. De forma que, quanto menor o valor configurado, menor o número de serviços que atendem ao requisito de tempo de resposta. Os resultados mostram que o tempo de seleção aumenta com o aumento do valor do tempo de resposta em todas as aplicações. O experimento da Figura 61c mostra que classes de orçamentos foram configuradas como descritas no início da Subseção 6.2. O eixo y da Figura 61c mostra os resultados, os quais são constante para todas as aplicações. Isso indica que o tempo médio de seleção é independente do valor do orçamento. O tempo médio de seleção é influenciado apenas pelo número de microsserviços, pois a diferença entre as aplicações é a quantidade de microsserviços. A Figura 61d mostra os resultados do experimento onde os valores dos requisitos de disponibilidade, tempo de resposta e custo são modificados ao mesmo tempo. Para isto, foram criadas classes de requisitos como descrito no início da Subseção 6.2. Os resultados mostram que o tempo médio de seleção para a primeira classe é alto, mas que o tempo médio de seleção para a segunda classe é muito menor que o primeiro. Observar-se também que o tempo médio de seleção para os outros valores é muito menor. Isso acontece porque a maioria dos provedores atende a todas as classes com valores menores.

6.2.3 Experimentos para a Solução de UM^2Q

Esta seção tem como objetivo avaliar o desempenho da solução e da ferramenta proposta para o modelo UM^2Q, o qual seleciona múltiplos provedores de nuvem baseado em



(a) Disponibilidade.

(b) Tempo de resposta.

(c) Classe de Orçamentos.

(d) Classe de Requisitos.

Figura 61. Experimentos para a solução GUM^2K.

requisitos individuais. Assim, além de definir um orçamento para uma aplicação, um arquiteto de *software* deve definir também cotas de orçamento para cada um dos microsserviços de uma aplicação. A fim de alcançar este objetivo, primeiro, cenários são configurados para verificar o comportamento da solução proposta.

Como a maioria das pesquisas usam conjunto de dados configurado sinteticamente, para estes experimentos foram configurado 10 conjuntos de provedores sintéticos apresentados na Tabela 10, que diferem um do outro pelo número de provedores e capacidades dos serviços. Cada provedor possui sete serviços por classe. A quantidade de provedores em cada conjunto é um valor múltiplo de 100, no qual o primeiro conjunto tem 100 provedores e o último tem 1000. As aplicações são configuradas como na solução gulosa do modelo UM^2K e apresentadas na Tabela 9.

O tempo de seleção da solução proposta para o modelo UM^2Q foi avaliado usando a ferramenta desenvolvida para a solução. Para isso, 5 experimentos foram realizados e foram utilizados os cenários descritos nesta seção. O primeiro experimento é ilustrado pela Figura 62, para o qual foram usados os 10 conjuntos de provedores apresentados na Tabela 10. Os requisitos de disponibilidade, tempo de resposta e custo não foram modificados durante o experimento,

TABELA 10. CONFIGURAÇÃO DOS CONJUNTOS DE PROVEDORES PARA A SOLUÇÃO DO UM^2Q

Solução	Características dos Conjuntos				
	Número de Conjuntos	Número de Provedores por Conjunto	Número de Serviços por Classe	Número de Serviços por Provedor	Número de Serviços por Conjunto
UM^2Q	10	100	7	21	2100
		200			4200
		300			6200
		400			8400
		500			10500
		600			12600
		700			14700
		800			16800
		900			18900
		1000			21000

mantendo o mesmo valor em todos os conjuntos. A Figura 62 mostra que o aumento no número de provedores influencia o tempo médio de seleção, uma vez que aumenta o número de serviços que atendem aos requisitos das aplicações. Além disso, pode-se observar que o número de microsserviços também influencia o tempo médio de seleção dos provedores.

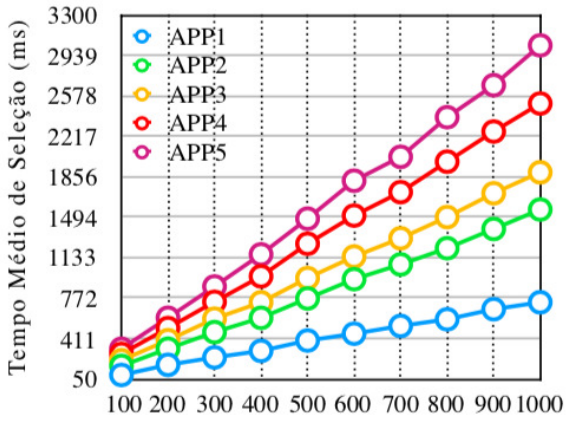
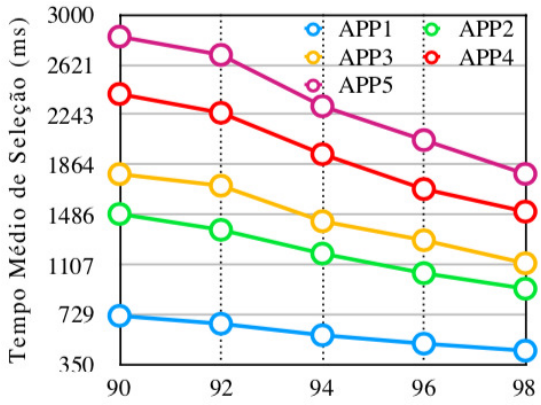


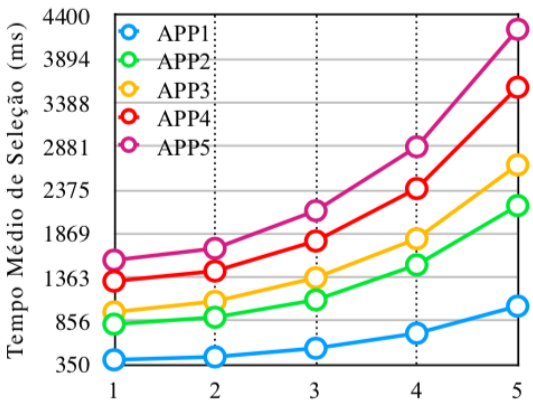
Figura 62. Experimento UM^2Q – Conjunto de Provedores.

Os outros 4 experimentos foram realizados usando um conjunto de 1000 provedores e são ilustrados pela Figura 63. No experimento da Figura 63a cada aplicação foi configurada com 5 valores diferentes, como configurado para os experimentos de disponibilidade das soluções do modelo UM^2K. Os requisitos de tempo de resposta e de custo foram definidos com valores que podem ser atendidos pela maioria dos provedores. Os resultados mostram que o tempo médio diminui com o aumento do valor do requisito de disponibilidade em cada aplicação. A Figura 63b mostra o experimento onde cada aplicação foi configurada com 5 valores de tempo de resposta. De forma que, quanto menor o valor do tempo de resposta, menos serviços de nuvem atendem ao requisito. Os requisitos de disponibilidade e de custo foram definidos com valores que podem ser atendidos pela maioria dos provedores. O gráfico da Figura 63b mostra que o

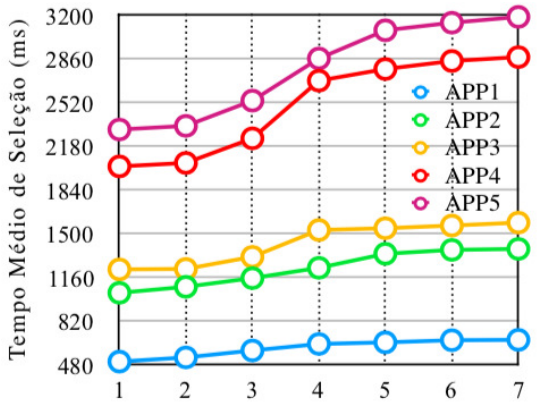
tempo de seleção aumenta com a diminuição do nível de exigência para tempo de resposta em cada aplicação. Para o experimento da Figura 63c classes de orçamentos foram criadas, como descrito no preâmbulo da Subseção 6.2. Os resultados mostram que o tempo médio de seleção aumenta conforme aumenta a classe de orçamentos, mas depois de uma certa quantidade, o tempo médio de seleção é estabilizado. Isso acontece porque o limite máximo foi atingido e, mesmo que o orçamento aumente, o número de serviço a ser selecionado não aumentará. A Figura 63d mostra o experimento quando a disponibilidade, o tempo de resposta e o custo são variados ao mesmo tempo. Para isto, classes de requisitos são criadas com descrito no preâmbulo da Subseção 6.2. Os resultados mostram que o tempo médio de seleção diminui com o aumento da restrição dos requisitos. Isto acontece porque o número de provedores que atende a todos os requisitos reduz e que o tempo médio de seleção tende a se estabilizar, porque o valor do orçamento é maior do que o valor de custo da nuvem.



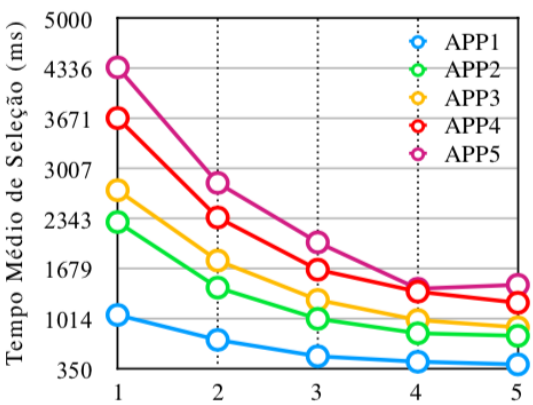
(a) Disponibilidade.



(b) Tempo de resposta.



(c) Classe de Orçamentos.



(d) Classe de Requisitos.

Figura 63. Experimentos para a solução UM^2Q.

6.2.4 Experimentos para a Solução Dinâmica de LM^2K

Esta seção avalia o desempenho da solução usando algoritmo dinâmico para o modelo LM^2K. Para alcançar este objetivo, configurou-se cenários contendo conjuntos de provedores (Tabela 11), informações sobre a comunicação entre os provedores disponíveis, aplicações (Tabela 12) e informações sobre o fluxo de execução entre os microsserviços de cada uma das aplicações, e entre as tarefas de cada microsserviço, como apresentado na Seção 6.1. Assim, como mostrado na Tabela 11, foram configurados 6 conjuntos de provedores, em que 4 deles se diferenciam pelo número de provedores por conjunto, e 2 deles se diferem pelo número de serviços por provedor. Assim, os conjuntos de provedores são organizados da seguinte forma: em 4 deles, cada provedor possui 12 serviços, e eles se diferenciam pela quantidade de provedores por conjunto: o primeiro possui 4, o segundo 5 e o terceiro possui 6 provedores; os outros 2 provedores se diferenciam pela quantidade de serviços por classe de serviços: o primeiro possui 5 e o segundo possui 6 serviços por classe, mas eles possuem 4 provedores por conjunto. Além disso, as informações sobre a comunicação entre os provedores disponíveis foram configuradas, sendo elas: disponibilidade, atraso e custo.

A Tabela 12 apresenta 5 aplicações: a primeira (APP1) possui 6 microsserviços, a segunda (APP2) possui 7, a terceira (APP3) possui 8, a quarta (APP4) possui 9 e a última (APP5) possui 10 microsserviços. Para cada aplicação um arquiteto de *software* deve definir o limiar para orçamento, disponibilidade e tempo de resposta e os pesos para cada um dos requisitos. Além disso, ele deve configurar os fluxos de execução entre os microsserviços de uma aplicação e de cada microsserviço. Assim, ele deve definir cada uma das sequências entre microsserviços, e para cada uma delas definir quais microsserviços pertencem à sequência, à ordem e à frequência. As mesmas informações devem ser definidas para as sequências de tarefas de cada microsserviço.

TABELA 11. CONFIGURAÇÃO DOS CONJUNTOS DE PROVEDORES PARA A SOLUÇÃO DINÂMICA DO LM^2K

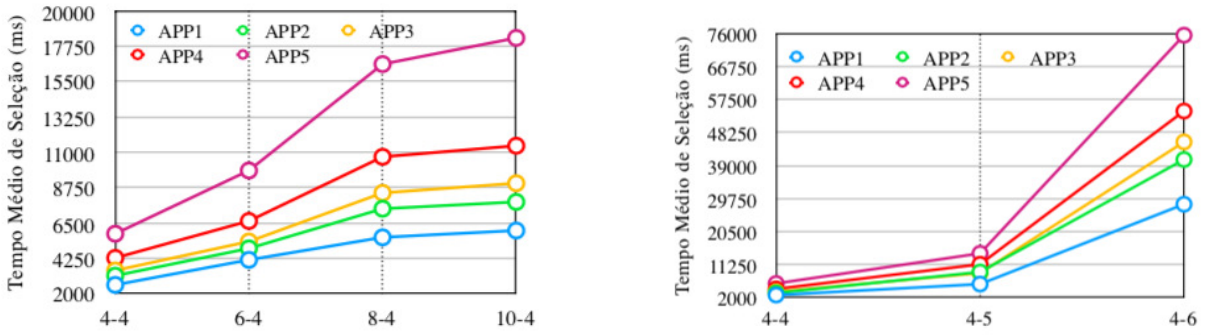
Solução	Características dos Conjuntos				
	Número de Conjuntos	Número de Provedores por Conjunto	Número de Serviços por Classe	Número de Serviços por Provedor	Número de Serviços por Conjunto
Dinâmica LM^2K	4	4			48
		5			60
		6			72
		7			84
	2	4	5	15	60
			6	18	72
			4	12	

Com os cenários configurados, experimentos foram definidos para avaliar o desem-

TABELA 12. CONFIGURAÇÃO DAS APLICAÇÕES PARA A SOLUÇÃO DINÂMICA DO LM^2K

Características	Aplicações				
	APP1	APP2	APP3	APP4	APP5
Número de Microsserviços	6	7	8	9	10
Número de Serviços na Aplicação	18	21	24	27	30

penho do processo de seleção. Primeiro, realiza-se os experimentos relacionados à quantidade de provedores por conjunto e à quantidade de serviços por provedor. Nestes experimentos, os requisitos de custo, disponibilidade e tempo de resposta não variaram, e foram configurados com valores que devem ser atendidos pela maioria dos provedores de nuvem disponíveis. Os resultados do primeiro e segundo experimento são apresentados na Figura 64. A Figura 64a ilustra o resultado do experimento, no qual varia-se a quantidade de provedores por conjunto. Pode-se observar no gráfico que o tempo de seleção aumenta significativamente com o aumento de número de provedores, mas que, a partir de um certo valor, a taxa de crescimento diminui. A Figura 64b mostra o resultado do experimento, no qual varia-se a quantidade de serviços por provedor. Nota-se que o aumento no tempo de médio de seleção é muito grande quando se altera o número de serviços por provedores.



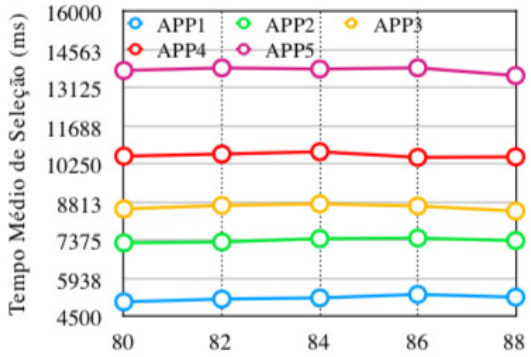
(a) Quantidade de Provedores por Conjunto.

(b) Número de Serviços por Provedor.

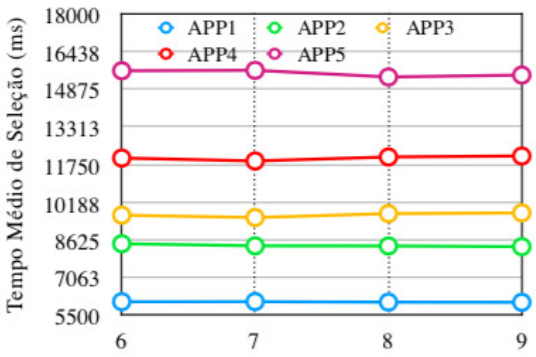
Figura 64. Experimentos relacionados a quantidade de provedores e serviços para DLM^2K.

Observando os resultados obtidos nestes dois experimentos, outros 4 experimentos foram realizados usando o conjunto da quinta linha da Tabela 11, o qual possui 4 provedores e cada provedor com 5 serviços por classe. A Figura 65 mostra em cada gráfico os outros 4 experimentos. No experimento da Figura 65a o requisito de disponibilidade das 5 aplicações foram configurados com 5 valores distintos. Os requisitos de custo e tempo de resposta foram configurados com valores que devem ser atendidos pela maioria dos provedores disponíveis. Observa-se que os valores de disponibilidade não influenciaram o tempo médio de seleção. Este fato acontece porque para, obter o valor de disponibilidade definido, os serviços de nuvem

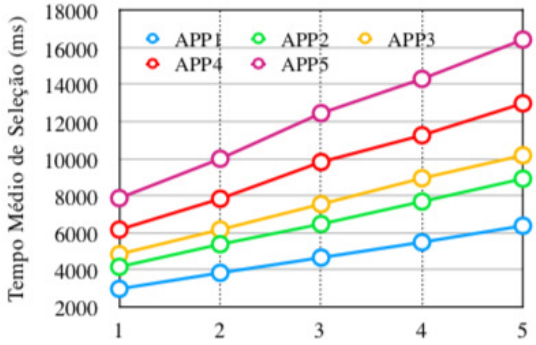
precisam ter valores altos de disponibilidade, desta forma a quantidade de serviços que possuem estes valores não altera muito de um valor de disponibilidade para outro. A Figura 65b verifica o desempenho da solução em relação à variação do requisito de tempo de resposta, onde as aplicações foram configuradas com 4 valores diferentes. Os requisitos de disponibilidade e custo foram definidos com valores que devem ser atendidos pela maioria dos provedores disponíveis. Os resultados mostram que os valores de tempo de resposta também não influenciam o tempo médio de seleção, como no experimento de disponibilidade. No experimento da Figura 65c classes de orçamentos foram definidas como descrito no preâmbulo da Subseção 6.2. Os resultados mostram que com o aumento da classe de orçamento, aumenta o tempo médio de seleção, diferentemente dos dois experimentos anteriores. Assim, o requisito de orçamento possui uma grande influência no tempo médio de seleção. No experimento da Figura 65d, no qual os três requisitos variam ao mesmo tempo, classes de requisitos foram criadas como descrito no início da Subseção 6.2. Os resultados mostram que o tempo médio de seleção aumenta com o aumento das classes de requisitos, influenciados pelo requisito de custo.



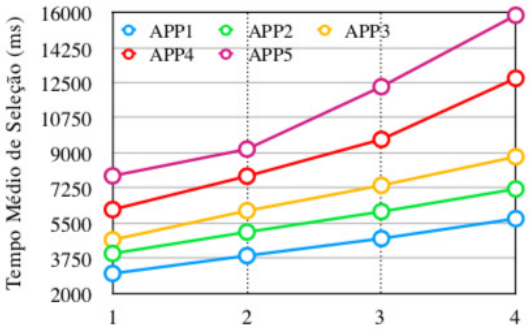
(a) Disponibilidade.



(b) Tempo de resposta.



(c) Classe de Orçamentos.



(d) Classe de Requisitos.

Figura 65. Experimentos para a solução DLM^2K.

6.2.5 Experimentos para a Solução Gulosa de LM^2K

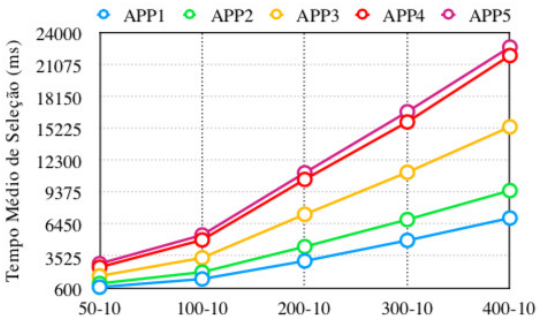
O objetivo desta seção é verificar o desempenho da solução gulosa para o modelo LM^2K. A fim de atingir este objetivo, foram configurados 9 conjuntos de provedores como mostrado na Tabela 13. De forma que, 5 conjuntos de provedores se diferenciam pela quantidade de provedores por conjunto e 4 que se diferenciam pelo número de serviços por provedor. Nos conjuntos que se diferenciam pela quantidade de provedores por conjunto, cada um eles possui 10 serviços por classe. Os conjuntos que se diferenciam pelo número de serviços por provedor possuem 50 provedores em cada conjunto, e a quantidade de serviços por provedor é múltipla de 5, o primeiro possui 10 e o último 25 serviços por classe. As informações definidas para a comunicação entre os provedores são: atraso, custo e disponibilidade. As aplicações são configuradas como na solução gulosa do modelo UM^2K e apresentadas na Tabela 9.

TABELA 13. CONFIGURAÇÃO DOS CONJUNTOS DE PROVEDORES PARA A SOLUÇÃO GULOSA DO LM^2K

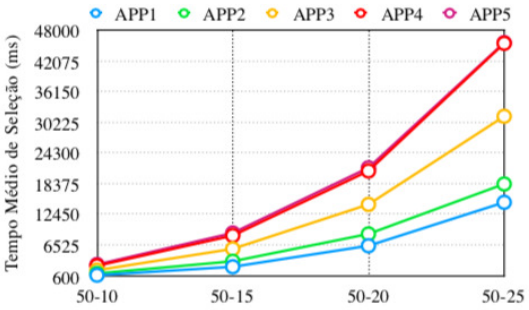
Solução	Características dos Conjuntos				
	Número de Conjuntos	Número de Provedores por Conjunto	Número de Serviços por Classe	Número de Serviços por Provedor	Número de Serviços por Conjunto
Gulosa LM^2K	5	50	10	30	1500
		100			3000
		200			6000
		300			9000
		400			1200
	4	50	10	30	1500
			15	45	2250
			20	60	3000
			25	75	3750

Depois de configurar todos os cenários, primeiramente dois experimentos foram realizados. O primeiro experimento verifica o desempenho da solução variando a quantidade de provedores por conjunto de provedores e o segundo variando o número de serviços por provedores. As Figuras 66a e 66b apresentam o resultado dos dois experimentos, respectivamente. Nos gráficos, o eixo x apresenta os valores relacionados aos experimentos, onde cada valor do eixo x é composto por dois números, o primeiro significa a quantidade de provedores por conjunto e o segundo o número de serviços por classe em cada provedor. Nos gráficos das Figuras 66a e 66b, pode-se notar que o tempo médio de seleção aumenta com o aumento da quantidade de provedores por conjunto e com o número de serviços por provedor, respectivamente. Ainda pode-se observar que a quantidade de serviços por provedor possui uma maior influência no aumento do tempo médio de seleção.

A partir dos 2 experimentos, 4 outros experimentos foram realizados, todos eles



(a) Quantidade de Provedores por Conjunto.



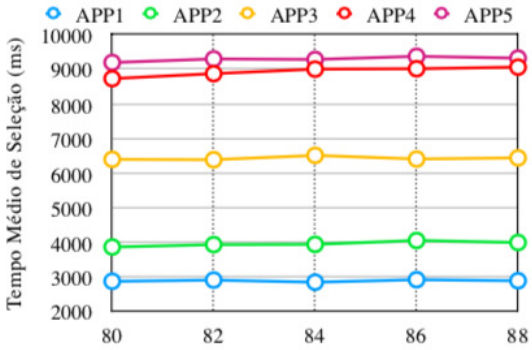
(b) Número de Serviços por Provedor.

Figura 66. Experimentos relacionados a quantidade de provedores e serviços para GLM^2K.

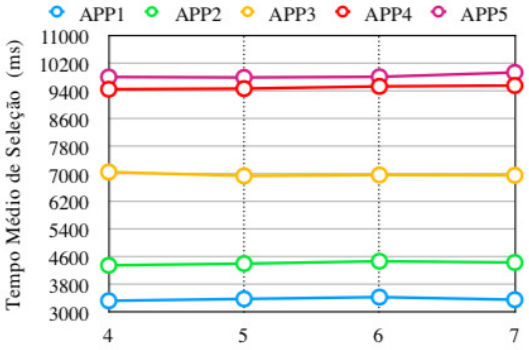
usaram um conjunto com 200 provedores, e cada provedor possui 10 serviços por classe, como descrito na terceira linha da Tabela 13. Os experimentos são ilustrados na Figura 67, a qual possui um gráfico para cada experimento. No experimento da Figura 67a, 5 valores diferentes foram configurados para o requisito de disponibilidade, e os requisitos de tempo de resposta e de custo foram configurados com valores fixos que podem ser atendidos pela maioria dos provedores. Para o experimento da Figura 67b, o requisito de tempo de resposta foi configurado com 4 valores distintos. Os requisitos de disponibilidade e custo se mantiveram fixos com valores que podem ser atendidos pela maioria dos provedores. A Figura 67c apresenta o resultado do experimento, no qual variou-se as classes de orçamentos como descrito no preâmbulo da Subseção 6.2. A Figura 67d mostra o resultado do experimento, onde os três requisitos foram modificados ao mesmo tempo, de forma que quando o requisito de disponibilidade aumenta, os requisitos de tempo de resposta e custo diminuem. Assim, a menor classe é a menos restrita e a maior é a mais restrita. Na Figura 67d, é possível observar que nenhum dos experimentos teve influência no tempo médio de seleção. Os valores se diferenciaram somente por aplicação, pois cada uma delas possui uma quantidade distinta de microsserviços.

6.2.6 Experimentos para a Solução usando Colônia de Formigas de LM^2K

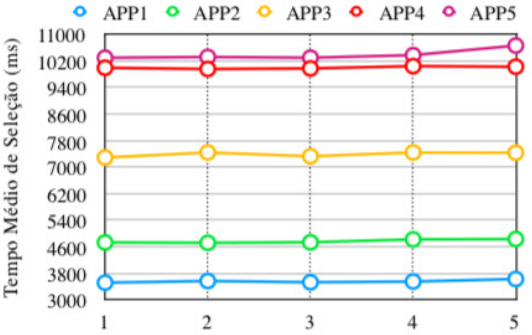
Esta seção visa investigar o desempenho da solução bioinspirada em colônia de formigas para o modelo LM^2K. Para alcançar este objetivo, cenários foram configurados como os da solução gulosa deste modelo. De acordo com a Tabela 14, foram configurados 4 conjunto de provedores, os quais se diferenciam pela quantidade de provedores por conjunto e outros 4 que se diferenciam pelo número de serviços por provedor. Nos conjuntos que se diferenciam pela quantidade de provedores por conjunto, cada um eles possui 10 serviços por classe. Cada conjunto possui como quantidade de provedores um valor múltiplo de 5, sendo que o primeiro



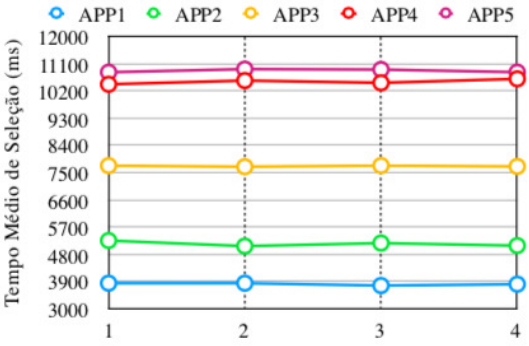
(a) Disponibilidade.



(b) Tempo de resposta.



(c) Classe de Orçamentos.



(d) Classe de Requisitos.

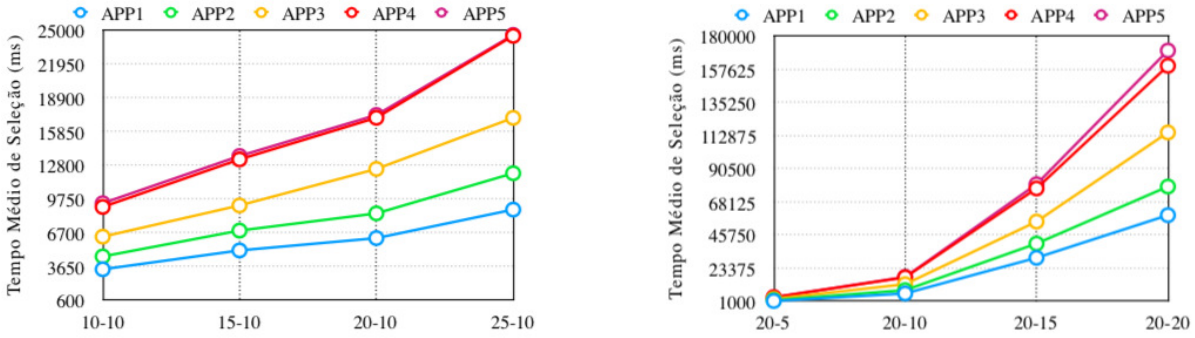
Figura 67. Experimentos para a solução GLM^2K.

conjunto possui 10 e o último possui 25 provedores. Os conjuntos que se diferenciam pelo número de serviços por provedor possuem 20 provedores em cada conjunto, e a quantidade de serviços por provedor é múltipla de 5, o primeiro possui 10 e o último 25 serviços por classe. As informações definidas para a comunicação entre os provedores são: atraso, custo e disponibilidade. As aplicações são configuradas como as das soluções gulosas dos modelos propostos e apresentadas na Tabela 9. Além disso, os fluxos de execução são definidos como na solução gulosa do modelo LM^2K, apresentada na Subseção 6.2.5.

TABELA 14. CONFIGURAÇÃO DOS CONJUNTOS DE PROVEDORES PARA A SOLUÇÃO BASEADA EM COLÔNIA DE FORMIGAS DO LM^2K

Solução	Características dos Conjuntos				
	Número de Conjuntos	Número de Provedores por Conjunto	Número de Serviços por Classe	Número de Serviços por Provedor	Número de Serviços por Conjunto
Colônia de Formigas LM^2K	4	10	10	30	300
		15			450
		20			600
		25			750
	4	20	10	30	600
			15	45	900
			20	60	1200
			25	75	1500

Com os cenários configurados, dois experimentos foram realizado com o objetivo de verificar o desempenho da solução variando a quantidade de provedores por conjunto e o número de serviços por provedores. As Figuras 68a e 68b apresentam o resultado dos dois experimentos, respectivamente. Observa-se que o tempo médio de seleção aumenta com o crescimento da quantidade de provedores por conjunto e com o aumento no número de serviços por provedor, respectivamente. E ainda pode-se observar que a quantidade de serviços por provedor possui uma maior influência no aumento do tempo médio de seleção.



(a) Quantidade de Provedores por Conjunto.

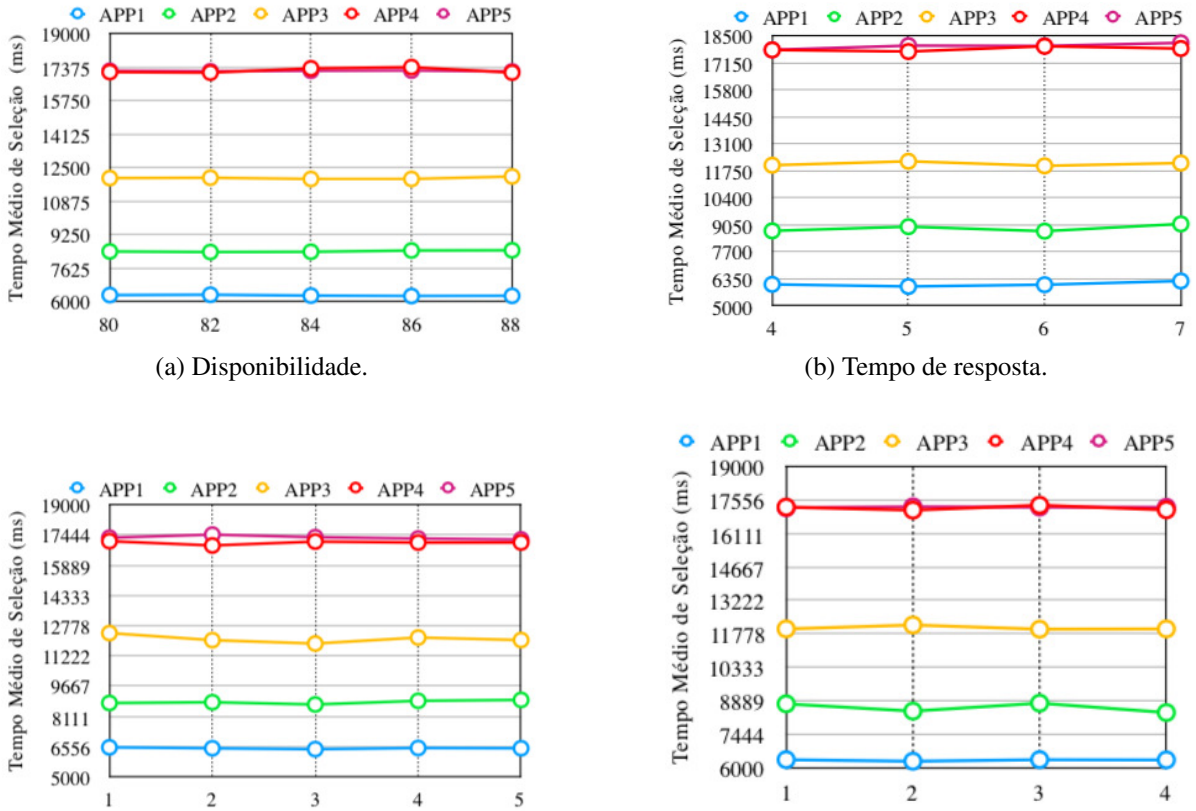
(b) Número de Serviços por Provedor.

Figura 68. Experimentos relacionados a quantidade de provedores e serviços para $ACOLM^2K$.

Outros 4 experimentos foram realizados, os quais usaram um conjunto com 20 provedores, e cada provedor possui 10 serviços por classe. Os experimentos são ilustrados pela Figura 69, a qual possui um gráfico para cada experimento. Todos os experimentos possuem a mesma configuração dos experimentos realizados para a solução gulosa do modelo LM^2K apresentados na Subseção 6.2.5, exceto pelos conjuntos de provedores. Como nos experimentos para a solução gulosa, observa-se que nenhum dos experimentos teve influência no tempo médio de seleção. Os valores se diferenciaram somente por aplicação, pois cada uma delas possui uma quantidade distinta de microsserviços.

6.3 Comparação de Resultados

O modelo UM^2K aborda aplicações baseadas em microsserviços, nas quais a dependência entre microsserviços pode ser desconsiderada. Assim, o único requisito global é o de custo, para o qual um orçamento é definido para uma aplicação. Além disso, a disponibilidade de uma aplicação é a disponibilidade do microsserviço que possui menor valor para este requisito. O requisito de tempo de resposta é o valor do tempo de resposta do microsserviço que possui



(a) Disponibilidade.

(b) Tempo de resposta.

(c) Classe de Orçamentos.

(d) Classe de Requisitos.

Figura 69. Experimentos para a solução $ACOLM^2K$.

maior valor para este requisito. Por fim, o custo é a soma dos custos de todos os microsserviços. Para este modelo duas soluções foram propostas: a primeira baseada em algoritmo dinâmico e a segunda em uma algoritmo guloso. Logo, diante do contexto e de acordo com os resultados dos experimentos apresentados nas Seções 6.2.1 e 6.2.2, pode-se observar que o tempo médio de seleção se modifica com a variação dos valores dos requisitos. Isto acontece porque existem valores menos restritos e valores mais restritos, ou seja, valores de requisitos que são atendidos por muitos provedores e valores que poucos deles atendem. As soluções se diferem em relação à influência do orçamento sobre o tempo médio de seleção, a solução gulosa não sofre influência deste requisito. Elas se diferenciam também em relação às entradas de dados, a solução gulosa aborda um maior número de provedores e aplicações maiores que a solução dinâmica para este modelo. Isto acontece principalmente pela influência do requisito de custo na solução dinâmica e do número de combinações candidatas dos microsserviços. Além disso, pode-se concluir que a solução dinâmica é eficiente para conjuntos de provedores pequenos, mas que são suficientes na maioria das situações.

O modelo UM^2Q trata todos os requisitos individualmente, inclusive o de custo, para

o qual são definidas cotas de orçamento para os microsserviços. Nesta solução, os resultados dos experimentos mostram que o tempo médio de seleção altera em relação à variação dos requisitos. Isto acontece pelo mesmo motivo do modelo anterior. Além disso, é possível notar, nos resultados dos experimentos em relação ao custo, que o tempo médio tende a se estabilizar depois de um certo valor. Isto significa que um limite foi atingido e que valores de orçamento maiores não obtém outras possibilidades de seleção. A solução para este modelo se diferencia das duas anteriores por aceitar como entrada um conjunto de provedores bem maior que os das anteriores. Além disso, ela possui valores médios de seleção inferiores ao da programação dinâmica. A limitação desta solução é definir inicialmente valores de cotas de orçamento para cada microsserviço.

O modelo LM^2K se difere dos dois anteriores, pois aborda aplicações baseadas em microsserviços que dependem uns dos outros, ou seja, existe uma comunicação entre os microsserviços que não pode ser desconsiderada. Assim, o cálculo para os requisitos de uma aplicação distribuída em múltiplas nuvens precisa considerar o fluxo de execução dos microsserviços e, consequentemente, a disponibilidade, o atraso e o custo de comunicação entre provedores. Portanto, as soluções para este modelo precisam tratar aspectos que não são tratados nos outros modelos. De acordo com este contexto e com os resultados obtidos dos experimentos, observa-se que, na maioria dos resultados, o tempo médio de seleção não se altera com a variação dos requisitos. Isto acontece porque, para atender aos limiares definidos por um arquiteto de *software*, é necessário que os provedores ofereçam serviços com valores altos. Assim, a variação na quantidade de oferta tem um valor insignificante no tempo médio de seleção. A exceção está na solução dinâmica para os experimentos que possuem variação de orçamento, pois estes influenciam no tempo médio de seleção, limitando, assim, o tamanho da entrada para essa solução. A solução que usa algoritmo guloso aceita uma entrada maior que a solução que usa colônia de formigas, possuindo tempo médio inferior ao desta. A solução que usa colônias de formiga aceita entrada maior do que a solução que usa programação dinâmica, bem como possui tempos médios de seleção menor.

Percebe-se que as soluções para o modelo LM^2k possuem tempo médio de seleção menor que os demais modelos. Isto acontece porque, para este modelo, a quantidade de serviços de nuvem que atendem aos limiares de um arquiteto de *software* é menor que para os outros modelos, como descrito acima. Desta forma, pode-se notar que cada um dos modelos podem ser utilizados, dependendo do cenário, devendo ser observado o número de provedores disponíveis,

o tamanho da aplicação e o grau de dependência entre microsserviços.

Todos os experimentos realizados variam pelo menos um dos requisitos ou variam a quantidade de provedores por conjunto ou a quantidade de serviços por provedor. Os experimentos mostram o tempo médio que cada solução precisa para selecionar os provedores de nuvem, para hospedar os microsserviços de uma aplicação. Nota-se que existem uma variação no tempo médio de seleção entre as aplicações, porque cada uma delas possui uma quantidade diferente de microsserviços. Uma exceção acontece no modelo LM^2K, no qual a diferença entre os tempos médios das aplicações APP4 e APP5 nas três soluções é quase imperceptível, pois neste modelo os valores dos requisitos entre provedores foram calculados e, assim, a diferença entre o número de microsserviços das aplicações teve pouca influência no tempo médio final.

De acordo com os experimentos realizados, observa-se também que ocorre uma variação nos provedores selecionados, conforme acontece uma variação nos valores dos requisitos da aplicação, ou no número de provedores por conjunto ou no número de serviços por provedor, ou ainda mudança nas capacidades de provedores, desta forma aparecem melhores ofertas. Isto mostra a sensibilidade das soluções propostas. Assim, um arquiteto de *software* deve decidir mover ou não um microsserviço de uma aplicação de um provedor para outro, observando as novas ofertas e a periodicidade de mudança.

6.4 Resumo do Capítulo

Este capítulo descreveu as ferramentas implementadas para cada uma das soluções propostas, detalhando seus arquivos de entrada. Experimentos relacionados ao desempenho foram descritos para as soluções, e para isso, configurou-se cenários adequados a cada uma das soluções propostas. Os resultados obtidos em todos os experimentos foram apresentados e comparados. Ao final, pode-se concluir que todas as soluções podem ser utilizadas de acordo com o cenário.

7 TRABALHOS RELACIONADOS

Este capítulo tem como objetivo apresentar, descrever e discutir trabalhos na literatura que sejam relacionados ao processo de seleção de múltiplos provedores de nuvem da perspectiva de um arquiteto de *software* para hospedar um aplicação distribuída. Para isso, pesquisas foram realizadas nas bases Springer[1], ScienceDirect[2] e IEEE Explorer[3], além da plataforma Researchgate[4] e Google Scholar[5]. Como refinamento foram utilizadas as *strings* de busca apresentadas na Tabela 15 e o período de publicação entre os anos de 2016 e 2019. Ao final, 23 trabalhos foram obtidos, depois de um processo de exclusão realizado por título, por trabalhos repetidos, por introdução e conclusão.

Tabela 15. *Strings* de busca para seleção de múltiplos provedores

ID	Search String
1	“selection” AND “multiple cloud” AND “microservice” AND (“QoS criteria” OR “multi-requirements“)
2	“selection” AND “multiple cloud” AND “service composition” AND (“QoS criteria” OR “multi-requirements“)
3	“selection” AND “multi-cloud” AND “service compistion” AND (“QoS criteria” OR “multi-requirements“)
4	“Raking” AND “multiple” AND “cloud” AND “service composition” AND (“QoS criteria” OR “multi-requirements“)
5	“scheduling” AND “multiple cloud” AND “service composition” AND (“QoS criteria” OR “multi-requirements”)

Neste trabalho de pesquisa, três modelos foram propostos para tratar o processo de seleção de múltiplos provedores de nuvem para implantar microsserviços da perspectiva de um arquiteto de *software*, sendo eles: UM^2K, UM^2Q e LM^2K. Para o modelo UM^2K, duas soluções foram propostas: uma baseada em um algoritmo dinâmico e outra um algoritmo guloso. Para o segundo modelo, propõe-se uma solução baseada em requisitos individuais para cada microsserviço. Por fim, para o terceiro modelo, três soluções foram propostas: uma utilizando algoritmo dinâmico, outra um algoritmo guloso e a outra um algoritmo baseado em colônia de formigas. Na literatura existem outros trabalhos que também tratam do problema de seleção de provedores de nuvem, mas muitos deles se concentram em outras questões. Alguns deles

[1] https://link.springer.com
[2] https://www.sciencedirect.com/
[3] https://ieeexplore.ieee.org/
[4] https://www.researchgate.net
[5] https://scholar.google.com/intl/en-US/scholar/about.html

selecionam serviços em uma única *cloud*, como por exemplo Chen *et al.* (2016) e Hongzhen *et al.* (2016). Já outros trabalhos tratam do problema usando *cloud federation*, como por exemplo Thomas e Chandrasekaran (2017) e Panda *et al.* (2018). Alguns trabalhos classificam serviços de nuvem Ding *et al.* (2017) Jatoth *et al.* (2018), enquanto outros abordam a seleção de provedores de nuvem para compor serviços, Bharath Bhushan e Pradeep Reddy (2016) Mezni e Sellami (2017). Certos trabalhos focam na avaliação da confiança, como as propostas de Tang *et al.* (2017) e Somu *et al.* (2018). Outros trabalhos se concentram na *cloud manufacturing*, como Zheng *et al.* (2016) e Zhou *et al.* (2018). Assim, vários trabalhos abordam o problema de seleção de provedores de nuvem, mas cada um deles se preocupa com diferentes aspectos. Isso de certa forma é normal, pois existem várias questões em aberto. Este capítulo descreve e compara alguns trabalhos de pesquisa que tratam do problema de seleção de provedores de nuvem a partir da perspectiva do usuário. Os trabalhos foram divididos em três categorias. A primeira delas descreve os trabalhos que selecionam serviços em um único provedor de nuvem, a segunda aborda os trabalhos que selecionam apenas um serviço dentre os serviços ofertados por múltiplos provedores e por último, os trabalhos que lidam com a seleção de vários serviços em múltiplos provedores.

7.1 Um provedor de nuvem na seleção de serviços

Esta seção faz uma breve descrição de trabalhos que utilizam apenas um provedor de nuvem para tratar a seleção de serviços. Os modelos propostos neste trabalho de doutorado também selecionam vários serviços em um único provedor de nuvem para compor um microsserviço, mas uma busca é realizada em todos os provedores disponíveis a fim de identificar a melhor oferta. Esse processo é repetido para todos os microsserviços de uma aplicação. Assim, os modelos propostos nesta tese tornam-se mais complexos. Além desta grande diferença, em alguns trabalhos descritos a seguir, os usuários não definem os limiares, as soluções propostas maximizam ou minimizam alguns dos requisitos para beneficiar os usuários. Diferenças mais específicas entre a presente tese e os trabalhos relacionados são tratadas a seguir.

Chen *et al.* (2016) propuseram um algoritmo de composição de serviços sensíveis à dependência (DASC) usando técnicas de *Vetor Ordinal Optimization* (VOO) para procurar soluções ideais de Pareto, considerando múltiplos requisitos e dependências de QoS para maximizar o lucro. Assim, os usuários não definem os limiares dos requisitos. As dependências significam melhores ofertas quando se escolhe serviços combinados. Além disso, os autores

consideram apenas estrutura sequencial para a composição de serviços. Os modelos propostos neste trabalho consideram os microsserviços independentes e não tratam o fluxo de trabalho entre eles ou consideram três estruturas: sequencial, paralela e seleção. Por fim, os autores focam em maximizar o lucro. Os modelos propostos nesta tese focam nas preferências de um arquiteto de *software* (usuário do modelos propostos) em relação ao custo, tempo de resposta e disponibilidade, podendo acrescentar outros requisitos com poucas alterações no código.

Hongzhen *et al.* (2016) propuseram uma estratégia de evolução da composição do serviço em nuvem, baseada em *hybrid particle swarm* combinado com algoritmo de estimação de distribuição de copula (CEDA) considerando múltiplos QoS. Para isso,eles exploram e descrevem um padrão de evolução e as expressões de cálculo da qualidade do serviço de nuvem. Depois, eles transformam o problema em um problema de programação multiobjetivo. Os valores de QoS são calculados com base no modelo de estrutura sequencial.

Liu *et al.* (2016) propuseram uma abordagem para a composição de serviços em nuvem com sensibilidade à QoS, que permite ao usuário definir as restrições de QoS e suas preferências. Para isso, os autores propõem um algoritmo de otimização de aprendizagem social (SLO), que simula o processo de evolução da inteligência humana. O SLO possui três espaços de co-evolução, onde a cooperação mútua destes espaços pode melhorar a capacidade de busca e a velocidade de convergência. A solução proposta pelos autores permite ao usuário definir os limiares dos requisitos. A proposta usa apenas a estrutura sequencial para composição de serviços, embora permita que outras estruturas sejam convertidas em sequenciais.

Seghir e Khababa (2016) também tratam o problema da composição de serviços em nuvem com sensibilidade de QoS e propõem um algoritmo genético híbrido (HGA) para resolvê-lo, incorporando o algoritmo de otimização da mosca das frutas (FOA). Os autores combinam os dois algoritmos com o objetivo de reduzir o tempo de computação e manter um equilíbrio entre as capacidades de exploração do espaço de busca global e local da HGA. A fase de otimização da mosca da fruta é realizada após cada evolução genética e um operador de elitismo é aplicado após o término dessa fase para evitar a perda das melhores soluções durante o processo evolutivo. Os autores usam a estrutura sequencial para a composição de serviços e convertem ou simplificam as outras estruturas. Como no trabalho aqui proposto, eles usam SAW para classificação dos serviços de nuvem.

Gavvala *et al.* (2019) propuseram uma solução para a composição de serviços usando a estratégia da águia com o algoritmo de otimização da baleia (*Eagle Strategy with*

Whale Optimization Algorithm) (ESWOA). A estratégia proposta pelos autores busca assegurar o equilíbrio adequado entre o espaço de busca global e o espaço de busca local. Como nesta tese, os autores usam SAW para classificação dos serviços candidatos e os usuários definem os limiares para os requisitos. A solução proposta por eles usa somente estrutura de composição de serviço sequencial, mas transformam os outros tipos de estruturas em estruturas sequenciais.

Jatoth *et al.* (2019) apresentaram o *Optimal Fitness Aware Cloud Service Composition* (OFASC) usando um algoritmo genético adaptativo, baseado em evolução de genótipos (AGEGA) que lida com vários parâmetros de QoS. Ele fornece as soluções que satisfazem os parâmetros de QoS e as restrições de conectividade da composição de serviço. A abordagem proposta pelos autores usa a estratégia de evolução progressiva do genótipo adaptativo para restringir as iterações de evolução. Essa estratégia considera os filhos da nova geração que têm melhor aptidão que qualquer um de seus pais. A fim de alcançar este objetivo, os autores propõem duas metodologias para avaliar a aptidão do serviço em nuvem e a adequação da composição, denominadas de distribuição de classificação uniforme discreta (DURD) e distribuição de classificação de serviço uniforme discreta (DUSRD). O estudo empírico da abordagem mostrou melhor desempenho em comparação com outras abordagens que são variações do algoritmo genético. No entanto, se houver um número muito grande de serviços candidatos na composição, o método proposto poderá resultar em um aumento do espaço de pesquisa dentro de ótimos locais.

Sun *et al.* (2019) propõem uma estrutura de seleção de serviços em nuvem com interações de critérios (CSSCI) que aplica uma medida difusa e uma integral de Choquet para medir e agregar relações não-lineares entre critérios. Os autores empregam um modelo de otimização de restrição não linear para estimar os índices de interação de importância e critérios de Shapley. Além disso, eles projetaram o PCSSCI, o qual é o CSSCI baseado em prioridade para resolver problemas de seleção de serviço em situações onde há falta de informações históricas para determinar relações e pesos de critérios, os quais se baseiam na ordem de prioridade definidas pelos usuários.

Nawaz *et al.* (2018) desenvolveram uma arquitetura de um *broker* para a seleção de serviços de nuvem, encontrando um padrão das prioridades mutáveis das preferências de um usuário (UPs). Para encontrar o padrão, uma cadeia de Markov foi empregada. O padrão é conectado à QoS para os serviços disponíveis. Um método MCDM, denominado método melhor pior (BWM) é usado para classificar os serviços. Assim, os autores focam na classificação e

recomendação de serviços baseando-se na mudanças de preferências do usuário de acordo com suas necessidades e com as experiências obtidas através do uso de serviços de nuvem.

Wu *et al.* (2016) propõem o conceito de serviços de componentes generalizados (GCSs), que é definido de maneira semântica, para expandir o escopo de seleção de forma a considerar instâncias de serviço candidatos que implementam vários serviços abstratos. Um modelo de composição de serviços de multi-granularidade sensível a QoS é formulado. Os autores também elaboraram como identificar todas as instâncias GCSs para composição de serviço. Por fim, um algoritmo baseado em *backtracking* e um algoritmo genético estendido são propostos para otimizar a composição de serviço de multi-granularidade sensível a QoS. Os limiares para os requisitos são definidos pelos usuários.

Khanam *et al.* (2018) introduzem uma abordagem chamada de otimização de enxame de partículas (PSO) modificada com base em QoS que reduz o espaço de pesquisa para a composição de serviços em nuvem. Além disso, os autores propõem um algoritmo de composição de serviço de nuvem baseado em PSO modificado (MPSO-CSC), que primeiro remove os serviços de nuvem dominados e, em seguida, emprega o PSO para encontrar o conjunto de serviços de nuvem ideais.

Kumar *et al.* (2019) tratam da seleção e composição dinâmica para composição de serviços de vários inquilinos. Os autores propuseram um *middleware multi-tenant* para composição dinâmica de serviços na nuvem SaaS. Em particular, os autores apresentaram uma representação de codificação e funções de adequação que modelam a seleção e a composição do serviço como uma busca evolucionária.

Os trabalhos descritos nesta seção, apesar de tratarem da seleção de serviços em computação em nuvem, abordam problemas diferentes em cada seção, e do problema tratado neste trabalho de doutorado. Apesar de serem diferentes, todos os trabalhos entendem a importância da seleção de serviços de nuvem por causa da proliferação de ofertas de serviços com as mesmas funcionalidades, mas com diferentes capacidades. Todos os trabalhos abordam questões em aberto, tal como este trabalho, mas aqui, trata-se de aspectos mais complexos, em razão da quantidade de serviços que devem ser selecionados em vários provedores de nuvem.

7.2 Múltiplos provedores de nuvem para selecionar um serviço

Os trabalhos relacionados nesta seção utilizam múltiplos provedores de nuvem no processo de seleção de serviços, mas selecionam apenas um serviço de um provedor de nuvem.

Assim, apesar de usarem múltiplos provedores de nuvem no processo de seleção, eles não selecionam vários serviços em múltiplos provedores de nuvem, bem como não inter-relacionam os serviços e os provedores como nesta tese.

Ding *et al.* (2017) apresentaram um método de predição orientado à classificação que ajuda no processo de descoberta de serviços de nuvem candidatos que gerem uma maior satisfação do usuário. A abordagem proposta pelos autores engloba duas funções básicas: classificação de serviços por similaridade e predição de classificação de serviço na nuvem, levando em consideração a preferência de um usuário. Para isso, os autores usaram um método de predição de classificação de serviços em nuvem denominado CSRP, que abrange a identificação de vizinhos semelhantes, as preferências de um cliente, a estimativa de satisfação do cliente e a previsão de classificação.

Jatoth *et al.* (2018) abordaram um modelo híbrido de tomada de decisão multicritério para seleção de serviços dentre os provedores disponíveis. A metodologia proposta atribui várias classificações para serviços de nuvem com base nos parâmetros QoS, usando uma técnica estendida de *Grey Technique for Order of Preference by Similarity to Ideal Solution* (TOPSIS), integrada ao processo hierárquico analítico (AHP).

Os trabalhos descritos nesta seção focam na classificação de serviços. Eles não se preocupam com a integração entre serviços e nem entre provedores de nuvem. Eles são relacionados a este trabalho por considerar necessário e difícil o processo de seleção de serviços de nuvem, principalmente quando se trata de múltiplos provedores.

7.3 Múltiplos provedores de nuvem para selecionar vários serviços

Esta seção descreve os trabalhos que usam múltiplos provedores de nuvem no processo de seleção e que selecionam múltiplos serviços em múltiplos provedores. A grande diferença desses trabalhos para os modelos propostos nesta tese é que eles selecionam apenas um serviço em cada provedor, e os modelos propostos aqui selecionam um microsserviço, o qual necessita de vários serviços de nuvem.

Thomas e Chandrasekaran (2017) propuseram o projeto e a implementação de um mecanismo de seleção de parceiros em uma *cloud federation*, usando o método AHP e a técnica TOPSIS e, para isso, eles consideram os valores de confiança de vários provedores de nuvem (CSPs) na federação. O método AHP é usado para calcular os pesos dos parâmetros de QoS usados no método TOPSIS, e este é usado para classificar os vários CSPs na *cloud federation* de

acordo com os requisitos do usuário. A solução proposta por Thomas e Chandrasekaran (2017) diferencia desta tese por usar o modelo de entrega *cloud federation*, o qual existe um acordo de colaboração entre os provedores. Além disso, o trabalho usa mais de um provedor somente quando o provedor requisitado para executar uma tarefa não puder cumprir o acordo de SLA. Neste contexto, as soluções propostas nesta tese possuem maior complexidade, pois utilizam o modelo de entrega *multi-cloud*, o qual não possui acordo de colaboração entre os provedores, normalmente um para cada microsserviço de uma aplicação.

Panda *et al.* (2018) propuseram dois algoritmos de escalonamento de tarefas em ambiente de *cloud federation* chamados contrato de nível de serviço - tempo de conclusão mínimo (SLA-MCT) e acordo de nível de serviço-min-min (SLA-Min-Min), ambos baseados em um SLA comum. O algoritmo recebe um conjunto de tarefas independentes e um conjunto de `m` provedores de nuvem, juntamente com tempo de execução, ganho e penalidade de custo de tarefas em provedores de nuvem diferentes. O problema é programar todas as tarefas para as nuvens com relação ao SLA, de modo que haja um equilíbrio entre o tempo de processamento geral e o ganho/penalidade de custo. O algoritmo SLA-MCT proposto é um escalonamento monofásico, enquanto o algoritmo SLA-Min-Min é um escalonamento bifásico. Ambos os algoritmos fornecem aos clientes dois parâmetros de seleção: o tempo e o custo de execução, conforme sua necessidade. Eles também mantêm um equilíbrio entre o processamento geral e o custo de execução dos serviços, a fim de atrair mais clientes.

Sousa *et al.* (2016) propuseram uma abordagem automatizada para a seleção e configuração de provedores de nuvem para aplicações baseadas em microsserviços em um ambiente *multi-cloud*. A abordagem proposta pelos autores usa uma linguagem específica de domínio para descrever os requisitos de um ambiente *multi-cloud* de uma aplicação e fornece um método sistemático para obter configurações adequadas que atendam aos requisitos de uma aplicação e às restrições dos provedores de nuvem. A solução proposta por Sousa *et al.* (2016) difere das propostas desta tese por tratar da seleção de serviços para compor um microsserviço, no qual os serviços que compõem os microsserviços estejam em provedores de nuvem distintos. Isto acarreta em atrasos, pois cada microsserviço executa várias tarefas que necessitam de vários serviços de nuvem. Neste contexto, os modelos propostos nesta tese, selecionam os serviços para executar as atividades de um microsserviço em um único provedor de nuvem. Como esta tese considera que uma aplicação é composta por vários microsserviços, os modelos propostos selecionam um provedor para cada microsserviço de uma aplicação, enquanto que a solução

proposta por Sousa *et al.* (2016) seleciona apenas um microsserviço.

Bharath Bhushan e Pradeep Reddy (2016) propuseram um algoritmo para composição de serviços baseado em múltiplos QoS com um número mínimo de combinações de provedores de nuvem. O algoritmo proposto seleciona um provedor de nuvem com maior número de arquivos de serviço e aplica *Preference Ranking Organization METHod for Enrichment Evaluation)* PROMETHEE, um método de tomada de decisão com vários critérios que seleciona o serviço mais adequado com base nos critérios de QoS (tempo de resposta, taxa de transferência, disponibilidade, sucesso, preço) dentre as melhores combinações dos provedores de nuvem. O trabalho proposto por Bharath Bhushan e Pradeep Reddy (2016) difere desta tese por selecionar um serviço por provedor de nuvem. Além disso, eles selecionam os serviços com o mínimo de provedores de nuvem.

Mezni e Sellami (2017) propuseram uma abordagem de composição de serviços para um ambiente *multi-cloud* (MCSC) baseada em análise de conceito formal (FCA). Os autores usaram o FCA para representar e combinar informações de múltiplos provedores de nuvem. O FCA baseia-se no conceito de rede, que é uma forma para classificar as informações sobre provedores de nuvem e serviços. Eles usam a teoria dos reticulados para reagrupar objetos com diversas nuvens em pequenos *clusters* baseados em propriedades comuns. Primeiro, eles modelam o ambiente de múltiplos provedores de nuvem como um conjunto de contextos formais. Em seguida, extraem e combinam os provedores de nuvem candidatos a partir de conceitos formais. Finalmente, a melhor combinação é selecionada e o MCSC é transformado em um problema clássico de composição de serviço. A solução proposta por Mezni e Sellami (2017) se diferencia desta tese pois seleciona um serviço por provedor para fazer uma composição de serviços. As soluções propostas nesta tese selecionam vários serviços por provedor. Os serviços selecionados em um provedor atendem às necessidades de um microsserviços de uma aplicação. Assim, cada provedor selecionado hospeda um microsserviço distinto de uma aplicação.

Yu *et al.* (2015) apresentaram um algoritmo guloso chamado Greedy-WSC e um algoritmo baseado em otimização de colônia de formigas chamado ACO-WSC, que seleciona uma combinação de serviços em múltiplos provedores de nuvem e com número mínimo de provedores de nuvem. As soluções propostas por Yu *et al.* (2015) se diferenciam das soluções propostas nesta tese pelos mesmos motivos da solução de Mezni e Sellami (2017).

Wang *et al.* (2015) apresentam uma estratégia de seleção de serviço de nuvem dinâmica chamada de DCS com base em *brokers* de serviços de nuvem para ajudar usuários a

selecionar seus serviços de nuvem preferidos. No processo de seleção de serviços, cada *broker* de serviços de nuvem gerencia alguns serviços em *cluster* e executa a estratégia DCS, cujo núcleo é um mecanismo de aprendizado adaptativo que compreende as funções de incentivo, esquecimento e degeneração. O mecanismo é desenvolvido para otimizar dinamicamente a seleção de serviços em nuvem e para retornar o melhor resultado de serviço ao usuário. Um conjunto de algoritmos de seleção dinâmica de serviços em nuvem é apresentado neste documento para implementar o mecanismo proposto. O trabalho de Wang *et al.* (2015) trata somente da seleção de serviços em múltiplos provedores de nuvem, não preocupando-se com a composição de serviços.

Moghaddam e Davis (2019) propuseram uma abordagem para seleção de serviço composto (CSS) para abordar a configuração que envolve várias solicitações simultâneas do serviço composto. Os autores propuseram um mecanismo de seleção de serviços baseado em leilões combinatórios para resolver esse problema, combinando simultaneamente as ofertas e solicitações de serviços da Web e realizando avaliações extensas em quatro setores de mercado, com tamanhos econômicos variados e complexidade de solicitações. Eles projetaram duas variações do mecanismo de leilão simultâneo, *full-matching* e *partial-matching*, e desenvolveram fórmulas de *integer linear programming* (ILP) para elas. No caso de , *full-matching* uma correspondência é buscada para todas as solicitações, enquanto a *partial-matching* relaxa essa suposição e visa encontrar correspondências para o maior subconjunto possível de solicitações. O trabalho realizado por Moghaddam e Davis (2019) seleciona várias combinações de serviços em um ambiente *multi-cloud*. Em relação à proposta desta tese, a grande diferença está no fato que os serviços de uma mesma combinação podem estar em provedores de nuvem distintos, enquanto que nos modelos propostos nesta tese cada combinação de serviço é selecionada em um único provedor de nuvem. Isto é feito para diminuir o atraso na execução de um microsserviço. Além disso, eles tratam em cada combinação somente a estrutura sequencial e não tratam da interação entre as combinações selecionadas.

Wang *et al.* (2017) propuseram uma estratégia de QoS para composição de serviços Web com base no modelo de nuvem considerando os ambientes em constante mudança, o qual pode refletir amplitude das flutuações do valor de desempenho de QoS e medir a estabilidade. Além disso, os autores criam um algoritmo de composição de serviços Web com vários objetivos, baseado em algoritmo genético quântico que consideram as origens físicas dos solicitantes de serviços e dos provedores de nuvem. Os autores tratam da composição de serviços em um ambiente de múltiplos provedores, mas em cada provedor é selecionado apenas um serviço.

Liu *et al.* (2018) propuseram um algoritmo evolutivo híbrido multi-objetivo chamado ADE-NSGA-II para compor serviços em um ambiente *inter-cloud* atendendo aos requisitos de QoS dos usuários. No algoritmo proposto utiliza-se uma mutação adaptativa e operador *crossover* para substituir estratégias do algoritmo genético de ordenação não-dominada-II (NSGA-II), de modo a buscar a solução mais eficiente dentro do espaço de soluções possíveis. Além disso, as principais estratégias do NSGA-II são mantidas para garantir a diversidade. O ambiente *inter-cloud* possui um *broker* para intermediar a negociação entre os provedores de nuvem. O trabalho de Liu *et al.* (2018) se diferencia desta tese, pois eles tratam da composição de serviços selecionando um serviço em cada provedor.

Para uma melhor visualização das principais características de cada trabalho relacionado aos propósitos desta tese, um resumo para cada um deles foi descrito na Tabela 16, a qual contém sete colunas. A primeira coluna enumera e a segunda apresenta a referência do trabalho. A terceira, a quarta e a quinta coluna referem-se a uma taxonomia para identificar o foco do trabalho que trata da seleção de provedores de nuvem em relação ao número de provedores e serviços usados por eles em cada fase, que denomina-se granularidade de provedores e serviços (*Provider and Service Granularity* - PSG). A terceira coluna mostra o número de provedores no processo de seleção, a quarta mostra o número de provedores selecionados e a quinta indica a granularidade de serviços por provedor selecionado. Além disso, a Tabela 16 apresenta, na sexta coluna, os métodos utilizados para resolver o problema de seleção, na sétima se o usuário define ou não o limiar de requisitos. Além disso, na Tabela 16, n indica o número de provedores no processo de seleção, que deve ser maior que 1; enquanto, m indica o número de provedores selecionados pelo processo de seleção e deve ser maior que 1 e menor ou igual a n.

7.4 Resumo do Capítulo

Este capítulo apresentou e descreveu os trabalhos relacionados com o processo de seleção de múltiplos provedores para hospedar microsserviços de uma aplicação. Apesar de descrever 23 trabalhos, nenhum deles trata do problema de seleção da mesma forma que os modelos propostos nesta tese. Mesmo diante deste contexto, os trabalhos reforçam a importância, a necessidade e a complexidade do processo de seleção de serviços. É possível perceber que o tema ainda está em aberto, pois não existe uma solução amplamente adotada, notadamente pela diversidade de aspectos a serem tratados.

Tabela 16. Resumo das características dos Trabalhos Relacionados.

Itens	Trabalhos Relacionados	Características				
		PSG			Método	Usuário define Limiar
		(1)	(2)	(3)		
1	Chen *et al.* (2016)	1	1	Composição de Serviço	Técnicas de Pareto e VOO	Não
2	Hongzhen *et al.* (2016)	1	1	Composição de Serviço	*Hybrid particle swarm* e CEDA	Não
3	Liu *et al.* (2016)	1	1	Composição de Serviço	SLO	Sim
4	Seghir e Khababa (2016)	1	1	Composição de Serviço	HGA e FOA	Sim
5	Gavvala *et al.* (2019)	1	1	Composição de Serviços	ESWOA	Sim
6	Jatoth *et al.* (2019)	1	1	Composição de Serviço	OFASC e AGEGA	Não
7	Sun *et al.* (2019)	1	1	Serviço	Definido pelos Autores	Não
8	Nawaz *et al.* (2018)	1	1	Classificação e recomendação de Serviços	AHP e BWM	Sim
9	Wu *et al.* (2016)	1	1	Composição de Serviço	Definido pelos Autores	Sim
10	Khanam *et al.* (2018)	1	1	Composição de Serviços	MPSO-CSC	Sim
11	Kumar *et al.* (2019)	1	1	Composição de Serviços	Definido pelos Autores	Sim
12	Ding *et al.* (2017)	n	1	Serviço	CSRP	Não
13	Jatoth *et al.* (2018)	n	1	Serviço	Técnica Grey, TOPSIS, AHP	Não
14	Thomas e Chandrasekaran (2017)	n	m	Recursos	TOPSIS, AHP	Não
15	Panda *et al.* (2018)	n	m	Tarefas	Definida pelos Autores	Não
16	Sousa *et al.* (2016)	n	m	Serviço	Definida pelos Autores	Não
17	Bharath Bhushan e Pradeep Reddy (2016)	n	m	Serviço	PROMETHEE	Não
18	Mezni e Sellami (2017)	n	m	Serviço	FCA	Não
19	Yu *et al.* (2015)	n	m	Serviço	Greedy-WSC e ACO-WSC	Sim
20	Wang *et al.* (2015)	n	m	Serviço	Definido pelos Autores	Sim
21	Moghaddam e Davis (2019)	n	m	Serviço	Definido pelos Autores	Não
22	Wang *et al.* (2017)	n	m	Serviço	Definido pelos Autores	Não
23	Liu *et al.* (2018)	n	m	Serviço	Definido pelos Autores	Sim
24	UM^2K Carvalho *et al.* (2018) e Carvalho *et al.* (2019)	n	m	Microsserviço	SAW, Algoritmo Dinâmico e Guloso	Sim
25	UM^2Q	n	m	Microsserviço	SAW, Definido pelos Autores	Sim
26	LM^2K	n	m	Microsserviço	SAW, Algoritmo Dinâmico, Guloso e Colônia de formigas	Sim

PSG-Granularidade de Provedor e Serviço, (1)-Granularidade do Provedor no Processo de Seleção, (2)- Granularidade do Provedor no Resultado da Seleção, (3)-Granularidade de Serviço por Provedor

8 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O crescimento da computação em nuvem e, consequentemente, do número de provedores e da quantidade de serviços ofertados por eles trazem muitas facilidade ao processo desenvolvimento de *software*. Além disso, os provedores de nuvem ofertam serviços especializados, de maneira que aumenta a possibilidade de um usuário se tornar refém de um provedor, ou seja, não conseguir migrar para outro provedor ou distribuir uma aplicação em múltiplos provedores de nuvem. Para que um arquiteto de *software* possa melhor obter as vantagens da computação em nuvem é necessário dar suporte ao uso de múltiplos provedores de nuvem, onde cada microsserviço possa ser hospedado em um provedor que melhor atenda às necessidades de uma aplicação. Embora, o uso de múltiplos provedores de nuvem traga muitos benefícios, há muitos desafios, principalmente quando não existe um acordo de cooperação entre os provedores de nuvem envolvidos, ou seja, em um ambiente *multi-cloud*.

Para uma aplicação tradicional, uma das primeiras decisões que um arquiteto de *software* deve tomar é escolher os provedores de nuvem para hospedar cada componente, pois a implementação depende do provedor selecionado. Entretanto, se as aplicações são desenvolvidas para a nuvem, elas devem usar um estilo arquitetônico que traga maior flexibilidade. Neste cenário, o estilo arquitetural mais utilizado é o baseado em microsserviços. Assim, se um arquiteto de *software* desenvolve uma aplicação baseada em microsserviços facilita a distribuição da mesma em múltiplos provedores de nuvem, mas mesmo assim existem os provedores de nuvem que ofertam muitos serviços com as mesmas funcionalidades, porém com diferentes capacidades. Além disso, cada microsserviço de uma aplicação pode precisar de vários serviços de nuvem. Logo, um arquiteto de *software* necessita selecionar um combinação de serviços para cada um dos microsserviços, observando todos os provedores de nuvem disponíveis, o que é uma tarefa complexa.

Neste sentido, o presente trabalho de doutorado propôs modelos e soluções para a seleção de múltiplos provedores de nuvem para hospedar cada microsserviço de uma aplicação da perspectiva de um arquiteto de *software*. O objetivo da tese é permitir que seja selecionada uma combinação de serviços em um único provedor para executar as tarefas de cada microsserviço da aplicação, observando seus requisitos e as demandas do arquiteto de *software*.

8.1 Contribuições da Pesquisa

De acordo com os objetivos aqui delimitados, as principais contribuições foram descritas nos Capítulos 4 e 5 e avaliados no Capítulo 6. As contribuições focam na proposta de modelos de seleção de múltiplos provedores de nuvem e nas soluções propostas para os modelos. Os três modelos propostos para seleção de múltiplos provedores focam em aplicações baseadas em microsserviços e na perspectiva de um arquiteto de *software*. Além disso, cada microsserviço é hospedado em um único provedor, de forma que é necessário selecionar os serviços para atender aos requisitos de um microsserviço e de um arquiteto de *software*. O modelo UM^2K foca em aplicações onde a comunicação entre os microsserviços pode ser desconsiderada e mapeia para o problema da mochila de múltiplas escolhas. UM^2Q é o modelo que concentra-se em aplicações onde a comunicação entre os microsserviços pode ser desconsiderada e individualiza quanto aos requisitos para os microsserviços, incluindo o orçamento destinado para hospedar uma aplicação em provedores de nuvem, colocando para cada microsserviço uma cota de orçamento. Por fim, LM^2K é o modelo que foca em aplicações onde a comunicação entre os microsserviços não pode ser desconsiderada. Assim, os requisitos devem ser considerados na comunicação entre provedores de acordo com o fluxo de execução dos microsserviços. Todas as soluções propostas para os modelos de seleção de múltiplos provedores usam SAW para classificar os serviços candidatos em cada um dos provedores. Duas soluções foram propostas para o modelo UM^2K, uma baseada em um algoritmo dinâmico e a outra em um algoritmo guloso. A segunda solução se mostrou mais eficiente em relação ao tamanho da entrada, pois seu desempenho é viável mesmo para aplicações e conjunto de provedores maiores. Para o modelo UM^2Q somente uma solução foi proposta, na qual todos os requisitos são individualizados por microsserviços, desta forma a seleção de um provedor para um microsserviço independe de outro microsserviço. Por isso, esta solução possui melhor desempenho em relação às demais soluções propostas. Para o modelo LM^2K três soluções foram propostas, a primeira é baseada em um algoritmo dinâmico, a segunda em um algoritmo guloso e a terceira em um algoritmo bioinspirada em colônias de formigas. Os resultados apontam que em relação ao desempenho, a solução usando colônias de formigas está entre a solução dinâmica e a gulosa. A gulosa possui o melhor desempenho em relação ao tamanho da entrada de dados. De acordo com os resultados, todas as soluções propostas são viáveis, dependendo do número de provedores e do tamanho da aplicação.

Além das contribuições descritas acima, este trabalho possui outras contribuições descritas nos Capítulos 2 e 3. Elas ajudaram a construir as soluções para alcançar os objetivos

desta pesquisa. A primeira delas está relacionada ao gerenciamento de recursos em múltiplos provedores, no qual uma definição e uma classificação para recursos em múltiplas nuvens (MCR) foram propostos. Além disso, três taxonomias foram propostas, as quais definem como e quando gerenciar MCR e como configurar recursos MCR da perspectiva de um arquiteto *software*. A outra é o PacificClouds, o qual é uma abordagem para o gerenciamento da implantação e execução de uma aplicação baseada em microsserviços da perspectiva de um arquiteto de *software*. Além de apresentar uma definição para microsserviços no contexto de *multi-cloud*.

Em relação às contribuições descritas nesta seção, obteve-se quatro publicações em congressos e periódicos, cujas referências são apresentadas na Tabela 17. Nos dois primeiros artigos, Carvalho *et al.* (2019) e Carvalho *et al.* (2018a) apresentam as soluções gulosa e dinâmica para o modelo de seleção UM^2K, respectivamente. No terceiro trabalho, Carvalho *et al.* (2018) apresentam um *survey* sobre o gerenciamento de recursos em múltiplos provedores de nuvem, fazendo uma revisão da literatura sobre o gerenciamento de MCR, no qual identifica-se que a maioria dos trabalhos focam na perspectiva dos provedores e que ainda existem muitas questões em aberto, por isso propõe-se uma definição, uma classificação e três taxonomias. No último trabalho, Carvalho *et al.* (2018b) propõem uma arquitetura que deve selecionar provedores para hospedar os microsserviços de uma aplicação através de uma das soluções indicadas neste trabalho, observando a aplicação que deve ser hospedada em múltiplos provedores, os provedores disponíveis e os requisitos de um arquiteto de *software*.

Tabela 17. Artigos a partir do Trabalho de Doutorado.

Referência	Qualis	Situação
J. Carvalho, D. Vieira, and F. Trinta, "Greedy Multi-cloud Selection Approach to Deploy an Application Based on Microservices," in PDP 2019, 2019.	A2	Publicado
J. Carvalho, D. Vieira, and F. Trinta, "Dynamic Selecting Approach for Multi-cloud Providers," in Cloud Computing – CLOUD 2018, 2018, pp. 37–51.	A2	Publicado
J. O. Carvalho, F. Trinta, D. Vieira, and O. A. C. Cortes, "Evolutionary solutions for resources management in multiple clouds: State-of-the-art and future directions," Future Generation Computer System, vol. 88, pp. 284–296, 2018.	A2	Publicado
J. O. De Carvalho, F. Trinta, and D. Vieira, "PacificClouds : A Flexible MicroServices based Architecture for Interoperability in Multi-Cloud Environments," in Closer 2018, 2018, pp. 448–455.	A2	Publicado

8.2 Limitações

Embora as soluções aqui propostas tragam benefícios, estas não avaliam a sensibilidade dos resultados de acordo com a variação de requisitos e prioridades. Este trabalho também não contempla uma comparação entre as soluções propostas da qualidade dos resultados,

pois há a necessidade de um cenário que possa ser utilizado em todas as soluções. Além disso, experimentos com dados reais tanto por parte da aplicação quanto por parte dos provedores de nuvem não foram realizados como em 50% dos trabalhos científicos devido a falta de base de dados que contemple as informações necessárias (HONGZHEN *et al.*, 2016).

8.3 Trabalhos Futuros

Neste trabalho foi realizada uma análise de desempenho das soluções propostas para a seleção de múltiplos provedores. Um trabalho futuro é uma análise sensitiva em relação aos serviços e provedores selecionados em cada uma das soluções. Uma análise sensitiva deve verificar se ocorre mudança nos serviços e ou provedores selecionados de acordo com a variação dos valores e/ou das prioridades dos requisitos. Outro trabalho é uma comparação entre as soluções propostas baseada na qualidade dos serviços selecionados em cada experimento. A qualidade do serviços está relacionada com os valores e prioridades do requisitos.

Em relação ao PacificClouds esta tese tratou a parte referente a seleção de provedores de nuvem para implantação de uma aplicação baseada em microsserviços. Outras pesquisas que podem ser realizadas em relação ao PacificClouds são: (i) um estudo sobre o processo de implantação automática dos microsserviços a partir da seleção dos provedores de nuvem, pois cada provedor possui suas especificidades; (ii) uma pesquisa sobre como monitorar múltiplos provedores de nuvem para obter informações sobre o uso dos recursos pelos microsserviços, bem como uma análise destas informações; (iii) um estudo sobre quando e como migrar microsserviços de um provedor para outro baseado na análise de informações do monitoramento. Em relação ao quando migrar significa quais são os parâmetros para decidir migrar um microsserviço, como por exemplo: todas as vezes que ocorrer uma violação dos limiares dos requisitos ou todas as vezes que aparecer uma melhor oferta ou pelo tamanho da variação ou ainda pelo histórico. O como migrar um microsserviço significa como mudar um microsserviço de um provedor para outro que minimize o *timedown* e que ocorra a menor perda de informações; (iv) uma pesquisa sobre como indicar para o arquiteto de *software* valores de requisitos, através dos quais podem obter uma melhor oferta.

REFERÊNCIAS

AL-FAIFI, A. M.; SONG, B.; ALAMRI, A.; ALELAIWI, A.; XIANG, Y. A survey on multi-criteria decision making methods for evaluating cloud computing services. **Journal of Internet Technology**, National Dong Hwa University * Computer Center, Taiwan, Republic of China, v. 18, n. 3, p. 473–494, 2017.

ALBOANEEN, D. A.; TIANFIELD, H.; ZHANG, Y. Glowworm swarm optimisation based task scheduling for cloud computing. In: **Proceedings of the Second International Conference on Internet of Things, Data and Cloud Computing**. New York, NY, USA: ACM, 2017. (ICC '17), p. 152:1–152:7. ISBN 978-1-4503-4774-7.

ALRIFAI, M.; RISSE, T. Combining global optimization with local selection for efficient qos-aware service composition. In: **Proceedings of the 18th International Conference on World Wide Web**. New York, NY, USA: ACM, 2009. (WWW '09), p. 881–890. ISBN 978-1-60558-487-4.

ANASTASI, G. F.; CARLINI, E.; COPPOLA, M.; DAZZI, P. QoS-aware genetic Cloud Brokering. **Future Generation Computer Systems**, Elsevier B.V., v. 75, p. 1–13, 2017. ISSN 0167739X.

ARDAGNA, D.; NITTO, E. D.; MILANO, P.; PETCU, D.; SHERIDAN, C.; BALLAGNY, C.; ANDRIA, F. D.; MATTHEWS, P. MODAC LOUDS : A Model-Driven Approach for the Design and Execution of Applications on Multiple Clouds. **Modeling in Software . . .**, p. 50–56, 2012. ISSN 2156-788.

ASLAM, S.; ISLAM, S. ul; KHAN, A.; AHMED, M.; AKHUNDZADA, A.; KHAN, M. K. Information collection centric techniques for cloud resource management: Taxonomy, analysis and challenges. **Journal of Network and Computer Applications**, Elsevier Ltd, v. 100, n. November, p. 80–94, 2017. ISSN 10958592.

BABU, A. A.; RAJAM, V. M. A. Resource Scheduling Algorithms in Cloud Environment - A Survey. **2017 Second International Conference on Recent Trends and Challenges in Computational Models (ICRTCCM)**, p. 25–30, 2017.

Bharath Bhushan, S.; Pradeep Reddy, C. H. A Qos aware cloud service composition algorithm for geo-distributed multi cloud domain. **International Journal of Intelligent Engineering and Systems**, v. 9, n. 4, p. 147–156, 2016. ISSN 21853118.

BROGI, A.; FAZZOLARI, M.; IBRAHIM, A.; SOLDANI, J.; WANG, P.; INFORMATICA, D.; CARRASCO, J.; CUBO, J.; DUR, F.; PIMENTEL, E.; MAL, U. D.; NITTO, E. D.; ELETTRONICA, D.; BIOINGEGNERIA, I.; MILANO, P.; ANDRIA, F. D. Adaptive management of applications across multiple clouds : The SeaClouds Approach. v. 18, n. 1, p. 1–14, 2015.

CANFORA, G.; PENTA, M. D.; ESPOSITO, R.; VILLANI, M. L. An approach for qos-aware service composition based on genetic algorithms. In: **Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation**. New York, NY, USA: ACM, 2005. (GECCO'05), p. 1069–1075. ISBN 1-59593-010-8.

CARVALHO, J.; VIEIRA, D.; TRINTA, F. Dynamic Selecting Approach for Multi-cloud Providers. In: LUO, M.; ZHANG, L.-J. (Ed.). **Cloud Computing – CLOUD 2018**. Cham: Springer International Publishing, 2018. p. 37–51. ISBN 978-3-319-94295-7.

CARVALHO, J.; VIEIRA, D.; TRINTA, F. Greedy Multi-cloud Selection Approach to Deploy an Application Based on Microservices. In: **PDP 2019**. [S.l.: s.n.], 2019.

CARVALHO, J. O.; TRINTA, F.; VIEIRA, D.; CORTES, O. A. C. Evolutionary solutions for resources management in multiple clouds: State-of-the-art and future directions. **Future Generation Computer Systems**, v. 88, p. 284–296, 2018. ISSN 0167739X.

CARVALHO, J. O. D.; TRINTA, F.; VIEIRA, D. PacificClouds : A Flexible MicroServices based Architecture for Interoperability in Multi-Cloud Environments. In: **Closer 2018**. [S.l.: s.n.], 2018. p. 448–455.

CHEN, Y.; HUANG, J.; LIN, C.; SHEN, X. Multi-Objective Service Composition with QoS Dependencies. **IEEE Transactions on Cloud Computing**, v. 7161, n. c, p. 1–1, 2016. ISSN 2168-7161.

COSTACHE, S.; DIB, D.; PARLAVANTZAS, N.; MORIN, C. Resource management in cloud platform as a service systems: Analysis and opportunities. **Journal of Systems and Software**, Elsevier Inc., v. 132, p. 98–118, 2017. ISSN 01641212.

DANDRIA, F.; BOCCONI, S.; CRUZ, J. G.; AHTES, J.; ZEGINIS, D. Cloud4SOA: Multi-cloud application management across paas offerings. **Proceedings - 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012**, p. 407–414, 2012.

DEMIRCI, M. A Survey of Machine Learning Applications for Energy-Efficient Resource Management in Cloud Computing Environments. **2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)**, p. 1185–1190, 2015.

DIEDRICH, F.; KEHDEN, B.; NEUMANN, F. Multi-objective problems in terms of relational algebra. In: BERGHAMMER, R.; MÖLLER, B.; STRUTH, G. (Ed.). **Relations and Kleene Algebra in Computer Science**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 84–98. ISBN 978-3-540-78913-0.

DING, S.; WANG, Z.; WU, D.; OLSON, D. L. Utilizing customer satisfaction in ranking prediction for personalized cloud service selection. **Decision Support Systems**, Elsevier B.V., v. 93, p. 1–10, 2017. ISSN 01679236.

DRAGONI, N.; GIALLORENZO, S.; LAFUENTE, A. L.; MAZZARA, M.; MONTESI, F.; MUSTAFIN, R.; SAFINA, L. Microservices: Yesterday, today, and tomorrow. In: ____. **Present and Ulterior Software Engineering**. Cham: Springer International Publishing, 2017. p. 195–216. ISBN 978-3-319-67425-4.

FARD, H. M.; PRODAN, R.; FAHRINGER, T. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. **IEEE Transactions on Parallel and Distributed Systems**, v. 24, n. 6, p. 1203–1212, 2013. ISSN 10459219.

FEHLING, C.; LEYMANN, F.; RETTER, R.; SCHUPECK, W.; ARBITTER, P. **Cloud Computing Patterns**. Vienna: Springer Vienna, 2014. 239–286 p. ISBN 978-3-7091-1567-1.

FERRER, A. J.; HERNÁNDEZ, F.; TORDSSON, J.; ELMROTH, E.; ALI-ELDIN, A.; ZSIGRI, C.; SIRVENT, R.; GUITART, J.; BADIA, R. M.; DJEMAME, K.; ZIEGLER, W.; DIMITRAKOS, T.; NAIR, S. K.; KOUSIOURIS, G.; KONSTANTELI, K.; VARVARIGOU,

T.; HUDZIA, B.; KIPP, A.; WESNER, S.; CORRALES, M.; FORGÓ, N.; SHARIF, T.; SHERIDAN, C. OPTIMIS: A holistic approach to cloud service provisioning. **Future Generation Computer Systems**, v. 28, n. 1, p. 66–77, 2012. ISSN 0167-739X.

FREY, S.; FITTKAU, F.; HASSELBRING, W. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. International Conference on Software Engineering (ICSE-13). San Francisco, CA, USA,18–26 May 2013. p. 512–521, 2013.

GAVVALA, S. K.; JATOTH, C.; GANGADHARAN, G. R.; BUYYA, R. QoS-aware cloud service composition using eagle strategy. **Future Generation Computer Systems**, Elsevier B.V., v. 90, p. 273–290, 2019. ISSN 0167739X.

GOLDBARG, M. C.; GOLDBARG, E.; LUNA, H. P. L. **Otimização combinatória e meta-heurísticas**: algoritmos e aplicações. Rio de Janeiro: Elsevier, 2016. ISBN 978-85-352-7812-5.

GONZALEZ, N. M.; CARVALHO, T. C. M. d. B.; MIERS, C. C. Cloud resource management: towards efficient execution of large-scale scientific applications and workflows on complex infrastructures. **Journal of Cloud Computing**, Journal of Cloud Computing: Advances, Systems and Applications, v. 6, n. 1, 2017. ISSN 2192113X.

GOVINDARAJAN, K.; KUMAR, V. S.; SOMASUNDARAM, T. S. 2016 IEEE Eighth International Conference on Advanced Computing (ICoAC) A Distributed Cloud Resource Management Framework for High-Performance Computing (HPC) Applications. p. 1–6, 2016.

GROZEV, N.; BUYYA, R. Inter-cloud architectures and application brokering: taxonomy and survey. **Software: Practice and Experience**, v. 44, n. 3, p. 369–390, 2014.

GUZEK, M.; BOUVRY, P.; TALBI, E. G. A survey of evolutionary computation for resource management of processing in cloud computing [review article]. **IEEE Computational Intelligence Magazine**, v. 10, n. 2, p. 53–67, 2015. ISSN 1556603X.

HAMEED, A.; KHOSHKBARFOROUSHHA, A.; RANJAN, R.; JAYARAMAN, P. P.; KOLODZIEJ, J.; BALAJI, P.; ZEADALLY, S.; MALLUHI, Q. M.; TZIRITAS, N.; VISHNU, A.; KHAN, S. U.; ZOMAYA, A. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. **Computing**, Springer Vienna, v. 98, n. 7, p. 751–774, 2016. ISSN 0010485X.

HAYYOLALAM, V.; Pourhaji Kazem, A. A. A systematic literature review on QoS-aware service composition and selection in cloud environment. **Journal of Network and Computer Applications**, Elsevier Ltd, v. 110, n. February, p. 52–74, 2018. ISSN 10958592.

HEILIG, L.; LALLA-RUIZ, E.; VOSS, S. A cloud brokerage approach for solving the resource management problem in multi-cloud environments. **Computers and Industrial Engineering**, Elsevier Ltd, v. 95, p. 16–26, 2016. ISSN 03608352.

HOGENDIJK, J.; WHITESIDE, a. E. S. D. **Sources and Studies in the History of Mathematics and Physical Sciences**. [S.l.]: Springer US, 2011. 415 p. ISBN 978-0-387-87856-0.

HONGZHEN, X.; LIMIN, L.; DEHUA, X.; YANQIN, L. Evolution of Service Composition Based on Qos under the Cloud Computing Environment. **Proceedings of ICOACS 2016**, v. 2016, p. 66–69, 2016.

ITURRIAGA, S.; NESMACHNOW, S.; DORRONSORO, B.; TALBI, E.-G.; BOUVRY, P. A parallel hybrid evolutionary algorithm for the optimization of broker virtual machines subletting in cloud systems. **Proceedings - 2013 8th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2013**, p. 594–599, 2013.

JATOTH, C.; GANGADHARAN, G.; FIORE, U.; COMPUTING, R. B. SELCLOUD: a hybrid multi-criteria decision-making model for selection of cloud services. **Soft Computing**, p. 1–15, 2018.

JATOTH, C.; GANGADHARAN, G. R.; BUYYA, R. Optimal Fitness Aware Cloud Service Composition using an Adaptive Genotypes Evolution based Genetic Algorithm. **Future Generation Computer Systems**, Elsevier B.V., v. 94, p. 185–198, 2019. ISSN 0167739X.

JENA, R. K. Multi objective Task Scheduling in Cloud Environment Using Nested PSO Framework. **Procedia - Procedia Computer Science**, Elsevier Masson SAS, v. 57, p. 1219–1227, 2015. ISSN 1877-0509.

KARMARKAR, A. CAMP: A Standard for Managing Applications on a PaaS Cloud. In: **Proceedings of the 2014 Workshop on Eclipse Technology eXchange**. New York, NY, USA: ACM, 2014. (ETX '14), p. 1–2. ISBN 978-1-4503-2530-1.

KAUR, P.; MEHTA, S. Resource provisioning and work flow scheduling in clouds using augmented Shuffled Frog Leaping Algorithm. **Journal of Parallel and Distributed Computing**, Elsevier Inc., v. 101, p. 41–50, 2017. ISSN 07437315.

KELLERER, H.; PFERSCHY, U.; PISINGER, D. The multiple-choice knapsack problem. In: ____. **Knapsack Problems**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 317–347. ISBN 978-3-540-24777-7.

KESSACI, Y.; MELAB, N.; TALBI, E. G. A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation. **Cluster Computing**, v. 16, n. 3, p. 451–468, 2013. ISSN 13867857.

KHANAM, R.; KUMAR, R. R.; KUMAR, C. QoS based cloud service composition with optimal set of services using PSO. **Proceedings of the 4th IEEE International Conference on Recent Advances in Information Technology, RAIT 2018**, IEEE, p. 1–6, 2018.

KUMAR, B.; KALRA, M. Discrete Binary Cat Swarm Optimization for Scheduling Workflow Applications in Cloud Systems. p. 1–6, 2017.

KUMAR, N.; PATEL, P. Resource Management using Feed Forward ANN-PSO in Cloud Computing Environment. **Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies - ICTCS '16**, p. 1–6, 2016.

KUMAR, S.; BAHSOON, R.; CHEN, T.; LI, K.; BUYYA, R. Multi-Tenant Cloud Service Composition Using Evolutionary Optimization. **Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS**, v. 2018-December, p. 972–979, 2019. ISSN 15219097.

LEGILLON, F.; MELAB, N.; RENARD, D.; TALBI, E. G. Cost minimization of service deployment in a multi-cloud environment. **2013 IEEE Congress on Evolutionary Computation, CEC 2013**, p. 2580–2587, 2013.

LEGILLON, F.; MELAB, N.; RENARD, D.; TALBI, E.-G. A Multi-objective Evolutionary Algorithm for Cloud Platform Reconfiguration. **2015 IEEE International Parallel and Distributed Processing Symposium Workshop**, p. 286–291, 2015.

LIAQAT, M.; CHANG, V.; GANI, A.; HAMID, S. H. A.; TOSEEF, M.; SHOAIB, U.; ALI, R. L. Federated cloud resource management: Review and discussion. **Journal of Network and Computer Applications**, v. 77, n. May 2016, p. 87–105, 2017. ISSN 10958592.

LIU, H.; XU, D.; MIAO, H. K. Ant colony optimization based service flow scheduling with various QoS requirements in cloud computing. **Proceedings - 1st ACIS International Symposium on Software and Network Engineering, SSNE 2011**, p. 53–58, 2011.

LIU, J.; PACITTI, E.; VALDURIEZ, P.; OLIVEIRA, D. de; MATTOSO, M. Multi-objective scheduling of Scientific Workflows in multisite clouds. **Future Generation Computer Systems**, p. –, 2016. ISSN 0167-739X.

LIU, L.; GU, S.; ZHANG, M.; FU, D. A hybrid evolutionary algorithm for inter-cloud service composition. **Proceedings of 2017 9th International Conference On Modelling, Identification and Control, ICMIC 2017**, v. 2018-March, n. Icmic, p. 482–487, 2018.

LOUTAS, N.; PERISTERAS, V.; BOURAS, T.; KAMATERI, E.; ZEGINIS, D.; TARABANIS, K. Towards a Reference Architecture for Semantically Interoperable Clouds. In: **Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on**. [S.l.: s.n.], 2010. p. 143–150.

LUONG, N. C.; WANG, P.; NIYATO, D.; WEN, Y.; HAN, Z. Resource Management in Cloud Networking Using Economic Analysis and Pricing Models: A Survey. **IEEE Communications Surveys & Tutorials**, v. 19, n. 2, p. 954–1001, 2017. ISSN 1553-877X.

MADNI, S. H. H.; LATIFF, M. S. A.; COULIBALY, Y.; ABDULHAMID, S. M. Recent advancements in resource allocation techniques for cloud computing environment: a systematic review. **Cluster Computing**, Springer US, v. 20, n. 3, p. 2489–2533, 2017. ISSN 15737543.

MALEKLOO, M. H.; KARA, N.; El Barachi, M. An energy efficient and SLA compliant approach for resource allocation and consolidation in cloud computing environments. **Sustainable Computing: Informatics and Systems**, Elsevier Inc., v. 17, p. 9–24, 2018. ISSN 22105379.

MANN, Z. Á. Allocation of Virtual Machines in Cloud Data Centers—A Survey of Problem Models and Optimization Algorithms. **ACM Computing Surveys**, v. 48, n. 1, p. 1–34, 2015. ISSN 03600300.

MANSOURI, Y.; TOOSI, A. N.; BUYYA, R. Data Storage Management in Cloud Environments: Taxonomy, Survey, and Future Directions. **ACM Comput. Surv.**, v. 50, n. 6, p. 91:1—-91:51, 2017. ISSN 0360-0300.

MENNES, R.; SPINNEWYN, B.; LATRE, S.; BOTERO, J. F. GRECO: A Distributed Genetic Algorithm for Reliable Application Placement in Hybrid Clouds. **Proceedings - 2016 5th IEEE International Conference on Cloud Networking, CloudNet 2016**, p. 14–20, 2016.

MERIAM, E.; TABBANE, N. A survey on cloud computing scheduling algorithms. **Proceedings - 2016 Global Summit on Computer and Information Technology, GSCIT 2016**, p. 42–47, 2017.

MEZGÁR, I.; RAUSCHECKER, U. The challenge of networked enterprises for cloud computing interoperability. **Computers in Industry**, v. 65, n. 4, p. 657–674, 2014. ISSN 01663615.

MEZNI, H.; SELLAMI, M. Multi-cloud service composition using Formal Concept Analysis. **Journal of Systems and Software**, Elsevier Inc., v. 134, p. 138–152, 2017. ISSN 01641212.

MIDYA, S.; ROY, A.; MAJUMDER, K.; PHADIKAR, S. Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach. **Journal of Network and Computer Applications**, Elsevier Ltd, v. 103, n. December 2017, p. 58–84, 2018. ISSN 10958592.

MOGHADDAM, M.; DAVIS, J. G. Simultaneous service selection for multiple composite service requests: A combinatorial auction approach. **Decision Support Systems**, Elsevier, v. 120, n. July 2018, p. 81–94, 2019. ISSN 01679236.

MUSTAFA, S.; NAZIR, B.; HAYAT, A.; KHAN, A. U. R.; MADANI, S. A. Resource management in cloud computing: Taxonomy, prospects, and challenges. **Computers and Electrical Engineering**, Elsevier Ltd, v. 47, p. 186–203, 2015. ISSN 00457906.

NAWAZ, F.; ASADABADI, M. R.; JANJUA, N. K.; HUSSAIN, O. K.; CHANG, E.; SABERI, M. An MCDM method for cloud service selection using a Markov chain and the best-worst method. **Knowledge-Based Systems**, Elsevier, v. 159, n. November 2017, p. 120–131, 2018. ISSN 09507051.

NESMACHNOW, S.; CANCELA, H.; ALBA, E. A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. **Applied Soft Computing Journal**, Elsevier B.V., v. 12, n. 2, p. 626–639, 2012. ISSN 15684946.

NEWMAN, S. **Building Microservices**. [S.l.]: O'REILLY, 2015. 280 p. ISBN 978491950357.

NIST. **NIST Cloud Computing Standards Roadmap**. [S.l.], 2011.

NOGUEIRA, E.; MOREIRA, A.; LUCRÉDIO, D.; GARCIA, V.; FORTES, R. Issues on developing interoperable cloud applications: definitions, concepts, approaches, requirements, characteristics and evaluation models. **Journal of Software Engineering Research and Development**, Journal of Software Engineering Research and Development, v. 4, n. 1, p. 7, 2016. ISSN 2195-1721.

OPARA-MARTINS, J.; SAHANDI, R.; TIAN, F. Critical review of vendor lock-in and its impact on adoption of cloud computing. **International Conference on Information Society, i-Society 2014**, Journal of Cloud Computing, p. 92–97, 2015. ISSN 2192-113X.

PADMAVENI, K.; ARAVINDHAR, D. J.; TECHNOLOGY, O. Hybrid memetic and particle swarm optimization for Multi objective scientific wor k flows in cloud. 2016.

PANDA, S. K.; PANDE, S. K.; DAS, S. SLA-based task scheduling algorithms for heterogeneousmulti-cloud environment. **Arabian Journal for Science and Engineering**, v. 43, n. 2, p. 913–933, 2018. ISSN 21914281.

PANDEY, P. Job Scheduling Techniques in Cloud Environment : A Survey. p. 0–3, 2016.

PARIKH, S. M.; PATEL, N. M.; PRAJAPATI, H. B. Resource Management in Cloud Computing: Classification and Taxonomy. 2017.

PAUTASSO, C.; ZIMMERMANN, O.; AMUNDSEN, M.; LEWIS, J.; JOSUTTIS, N. Microservices in Practice, Part 1: Reality Check and Service Design. **IEEE Software**, v. 34, n. 1, p. 91–98, 2017. ISSN 07407459.

PETCU, D. Portability and Interoperability between Clouds : Challenges and Case Study (Invited Paper). **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 6994 LNCS, n. Section 3, p. 62–74, 2011. ISSN 03029743.

PETCU, D. Multi-Cloud: expectations and current approaches. **Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds - MultiCloud '13**, p. 1–6, 2013.

PETCU, D. Multi-Cloud: Expectations and Current Approaches. In: **Proceedings of the 2013 International Workshop on Multi-cloud Applications and Federated Clouds**. New York, NY, USA: ACM, 2013. (MultiCloud '13), p. 1–6. ISBN 978-1-4503-2050-4.

PETCU, D. Consuming Resources and Services from Multiple Clouds. **Journal of Grid Computing**, v. 12, n. 2, p. 321–345, jan 2014. ISSN 1570-7873.

PETCU, D. Consuming Resources and Services from Multiple Clouds: From Terminology to Cloudware Support. **Journal of Grid Computing**, v. 12, n. 2, p. 321–345, 2014. ISSN 15707873.

PETCU, D. Portability in Clouds : Approaches and Research Opportunities, Scalable Computing : Practice and Experience. v. 15, n. 3, p. 251–270, 2014.

PETCU, D.; Di Martino, B.; VENTICINQUE, S.; RAK, M.; MÁHR, T.; Esnal Lopez, G.; BRITO, F.; COSSU, R.; STOPAR, M.; ¿PERKA, S.; STANKOVSKI, V. Experiences in building a mOSAIC of clouds. **Journal of Cloud Computing: Advances, Systems and Applications**, v. 2, n. 1, p. 12, 2013. ISSN 2192-113X.

PETCU, D.; MACARIU, G.; PANICA, S.; CRAČIUN, C. Portable cloud applications - From theory to practice. **Future Generation Computer Systems**, Elsevier B.V., v. 29, n. 6, p. 1417–1430, 2013. ISSN 0167739X.

PHAN, D. H.; SUZUKI, J.; CARROLL, R.; BALASUBRAMANIAM, S.; DONNELLY, W.; BOTVICH, D. Evolutionary multiobjective optimization for green clouds. **Proceedings of the 14th International Conference on Genetic and Evolutionary Computation Conference Companion - GECCO Companion '12**, p. 19, 2012.

PISINGER, D. **Algorithms for Knapsack Problemas**. 200 p. Tese (Doutorado) — University of Copenhagen, 1995.

POULLIE, P.; BOCEK, T.; STILLER, B. A Survey of the State-of-the-Art in Fair Multi-resource Allocations for Data Centers. **IEEE Transactions on Network and Service Management**, v. 15, n. 1, p. 169–183, 2017. ISSN 19324537.

RAHIMI, M. R.; VENKATASUBRAMANIAN, N.; MEHROTRA, S.; VASILAKOS, A. V. MAPCloud: Mobile applications on an elastic and scalable 2-tier cloud architecture. **Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012**, p. 83–90, 2012.

REHMAN, Z.-u.; HUSSAIN, O. K.; HUSSAIN, F. K. User-side cloud service management: State-of-the-art and future directions. **Journal of Network and Computer Applications**, Elsevier, v. 55, p. 108–122, 2015. ISSN 10848045.

RV, R. **Spring Microservices**. [S.l.]: Pack Publishing, 2016. ISBN 9781786466686.

SCHMADER, K. E. **Essentials of CLOUD COMPUTING**. [S.l.: s.n.], 2015. ISBN 978-1-4822-0544-2.

SEGHIR, F.; KHABABA, A. A hybrid approach using genetic and fruit fly optimization algorithms for QoS-aware cloud service composition. **Journal of Intelligent Manufacturing**, Springer US, p. 1–20, 2016. ISSN 15728145.

SETHI, M. **Cloud Native Python: build and deploy applications on the cloud using microservices, AWS, Azure and more**. [S.l.: s.n.], 2017. ISBN 978-1-78712-931-3.

SHEIKHOLESLAMI, F.; NAVIMIPOUR, N. J. Service allocation in the cloud environments using multi-objective particle swarm optimization algorithm based on crowding distance. **Swarm and Evolutionary Computation**, Elsevier B.V., v. 35, n. February, p. 53–64, 2017. ISSN 2210-6502.

SILVA, E. A. N.; LUCRÉDIO, D. Software engineering for the cloud: A research roadmap. **Software Engineering (SBES), 2012 ...**, 2012.

SILVA, G. C.; ROSE, L. M.; CALINESCU, R. A Systematic Review of Cloud Lock-In Solutions. **2013 IEEE 5th International Conference on Cloud Computing Technology and Science**, p. 363–368, 2013.

SINGH, S.; CHANA, I. QoS-Aware Autonomic Resource Management in Cloud Computing. **ACM Computing Surveys**, v. 48, n. 3, p. 1–46, 2015. ISSN 03600300.

SINGH, S.; CHANA, I. A Survey on Resource Scheduling in Cloud Computing: Issues and Challenges. **Journal of Grid Computing**, Journal of Grid Computing, v. 14, n. 2, p. 217–264, 2016. ISSN 15729184.

SINGH, S.; CHANA, I. Cloud resource provisioning: survey, status and future research directions. **Knowledge and Information Systems**, Springer London, v. 49, n. 3, p. 1005–1069, 2016. ISSN 02193116.

SMITH, E.; SHIRER, M. **Worldwide Public Cloud Services Spending Forecast to Reach $210 Billion This Year, According to IDC**. 2019.

SOMASUNDARAM, T. S.; GOVINDARAJAN, K. CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. **Future Generation Computer Systems**, v. 34, p. 47–65, 2014. ISSN 0167-739X.

SOMU, N.; GAUTHAMA, G. R.; KIRTHIVASAN, K.; SHANKAR, S. S. A trust centric optimal service ranking approach for cloud service selection. **Future Generation Computer Systems**, Elsevier B.V., v. 86, p. 234–252, 2018. ISSN 0167739X.

SOSINSKY, B. **Cloud Computing Bible, Wiley Publishing Inc.** [S.l.: s.n.], 2011. 473 p. ISBN 9780470903568.

SOUSA, G.; RUDAMETKIN, W.; DUCHIEN, L. Automated Setup of Multi-Cloud Environments for Microservices-Based Applications. **9th IEEE International Conference on Cloud Computing**, 2016.

SUN, L.; DONG, H.; HUSSAIN, O. K.; HUSSAIN, F. K.; LIU, A. X. A framework of cloud service selection with criteria interactions. **Future Generation Computer Systems**, Elsevier B.V., v. 94, p. 749–764, 2019. ISSN 0167739X.

TANG, M.; DAI, X.; LIU, J.; CHEN, J. Towards a trust evaluation middleware for cloud service selection. **Future Generation Computer Systems**, Elsevier B.V., v. 74, p. 302–312, 2017. ISSN 0167739X.

THOMAS, M. V.; CHANDRASEKARAN, K. Dynamic partner selection in Cloud Federation for ensuring the quality of service for cloud consumers. **International Journal of Modeling, Simulation, and Scientific Computing**, v. 08, n. 03, p. 1750036, 2017. ISSN 1793-9623.

TOCZE, K.; NADJM-TEHRANI, S. Where Resources Meet at the Edge. **IEEE CIT 2017 - 17th IEEE International Conference on Computer and Information Technology**, p. 302–307, 2017.

TOOSI, A. N.; CALHEIROS, R. N.; BUYYA, R. Interconnected Cloud Computing Environments. **ACM Computing Surveys**, v. 47, n. 1, p. 1–47, 2014. ISSN 03600300.

ULLRICH, M.; LASSIG, J.; GAEDKE, M. Towards Efficient Resource Management in Cloud Computing: A Survey. **2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)**, p. 170–177, 2016.

VAQUERO, L. M.; RODERO-MERINO, L.; CACERES, J.; LINDNER, M. A break in the clouds. **ACM SIGCOMM Computer Communication Review**, v. 39, n. 1, p. 50, 2009. ISSN 01464833.

WANG, X.; CAO, J.; XIANG, Y. Dynamic cloud service selection using an adaptive learning mechanism in multi-cloud computing. **Journal of Systems and Software**, Elsevier Inc., v. 100, p. 195–210, 2015. ISSN 01641212.

WANG, Y.; HE, Q.; YE, D.; YANG, Y. Service Selection Based on Correlated QoS Requirements. **2017 IEEE International Conference on Services Computing (SCC)**, p. 241–248, 2017.

Wei-Neng Chen; Jun Zhang. An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 39, n. 1, p. 29–43, 2009. ISSN 1094-6977.

WU, Q.; ISHIKAWA, F.; ZHU, Q.; SHIN, D. H. QoS-Aware Multigranularity Service Composition: Modeling and Optimization. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 46, n. 11, p. 1565–1577, 2016. ISSN 21682232.

YOUSAFZAI, A.; GANI, A.; NOOR, R. M.; SOOKHAK, M.; TALEBIAN, H.; SHIRAZ, M.; KHAN, M. K. Cloud resource allocation schemes: review, taxonomy, and opportunities. **Knowledge and Information Systems**, Springer London, v. 50, n. 2, p. 347–381, 2017. ISSN 02193116.

YU, Q.; CHEN, L.; LI, B. Ant colony optimization applied to web service compositions in cloud computing. **Computers & Electrical Engineering**, v. 41, p. 18–27, 2015. ISSN 0045-7906.

ZEGRARI, F.; IDRISSI, A.; REHIOUI, H. Resource Allocation with Efficient Load Balancing in Cloud Environment. **Proceedings of the International Conference on Big Data and Advanced Wireless Technologies - BDAW '16**, p. 1–7, 2016.

ZENG, L.; BENATALLAH, B.; NGU, A. H.; DUMAS, M.; KALAGNANAM, J.; CHANG, H. QoS-aware middleware for Web services composition. **IEEE Transactions on Software Engineering**, v. 30, n. 5, p. 311–327, 2004. ISSN 00985589.

ZHAN, Z.-H.; LIU, X.-F.; GONG, Y.-J.; ZHANG, J.; CHUNG, H. S.-H.; LI, Y. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. **ACM Computing Surveys**, v. 47, n. 4, p. 1–33, 2015. ISSN 03600300.

ZHENG, H.; FENG, Y.; TAN, J. A fuzzy QoS-aware resource service selection considering design preference in cloud manufacturing system. **International Journal of Advanced Manufacturing Technology**, v. 84, n. 1-4, p. 371–379, 2016. ISSN 14333015.

ZHOU, J.; YAO, X.; LIN, Y.; CHAN, F. T.; LI, Y. An adaptive multi-population differential artificial bee colony algorithm for many-objective service composition in cloud manufacturing. **Information Sciences**, Elsevier Inc., v. 456, p. 50–82, 2018. ISSN 00200255.

ZIADE, T. **Python Microservices Development**. [S.l.]: Pack Publishing Ltd., 2017. ISBN 978-1-78588-111-4.

APÊNDICE A – TRABALHOS RELACIONADOS AO MCRM

Computação Evolutiva no MCRM

Como mencionado no preambulo do Capítulo 2, o gerenciamento de recursos para múltiplas nuvens nesta tese é abordado usando computação evolucionária. Este trabalho se concentra principalmente no gerenciamento de recursos da perspectiva de um arquiteto de *software*, mas poucos trabalhos abordam o gerenciamento de recursos sob o este ponto de vista. Assim, investigou-se as soluções propostas que tratam o gerenciamento de recursos para múltiplas nuvens usando computação evolucionária, independentemente se a solução proposta se concentra no usuário ou provedor. Para isto, os trabalhos mencionados por Guzek *et al.* (2015) referentes a múltiplas nuvens foram estendidos até 2017, e obteve-se 13 artigos que foram publicados entre os anos de 2012 e 2017 sobre o tema em questão. Esta seção descreve essas soluções, observando suas características e analisando-as.

Legillon *et al.* (2013) propuseram uma solução para o problema de disposição de serviços em máquinas virtuais que usam infraestrutura como serviço (IaaS) em ambiente *multi-cloud* para manutenção de uma plataforma viável. Essa abordagem é importante principalmente para *cloud brokers* e especialistas em computação que procuram minimizar custos. A solução proposta é um algoritmo genético (*Genetic Algorithm* (GA)) com várias mutações e cruzamento de operadores específicos para o problemas como *scaling up*, *scaling down*, *splitting*, *merging* e *removing* de VMs. Os autores fizeram experimentos nos quais compararam o GA com o MIP (*Mixed Integer Linear Programming*). O algoritmo proposto fornece soluções quase ótimas para instâncias pequenas. Para instâncias maiores, o desempenho do algoritmo é pior, porém com melhor estabilidade que o MIP, que caiu em alguns casos. Portanto, um bom resultado é alcançado em um período razoável de tempo em problemas pequeno e de médio porte.

A migração e a implantação de *software* corporativo na nuvem ainda enfrentam vários desafios. Frey *et al.* (2013) apresentam um algoritmo genético baseado em simulação CDOXplorer que otimiza as opções de implantação em nuvem (*cloud deployment option*s (CDOs)) para suportar a migração de *software* para a nuvem. Um CDO compreende uma combinação de um ambiente de nuvem específico, uma arquitetura de implantação e regras de reconfiguração em tempo de execução para *scaling* de recursos dinâmicos. Além disso, eles também são destinados a suportar sistemas distribuídos e consideram componentes executados em um nó como um único serviço oferecido a um número arbitrário de componentes em outros

nós. Por fim, o uso de CDOs se torna otimizado em relação ao tempo de resposta, custos e alguns requisitos de SLA. A avaliação dos autores mostrou que o CDOXplorer pode encontrar CDOs que são até 60% melhores que outras abordagens usadas para experimentos.

A computação de alto desempenho aparece como uma das áreas que adotaram a computação em nuvem. No entanto, os computadores usam uma parcela significativa e crescente de energia no mundo. Assim, a computação sensível ao consumo energético é crucial para sistemas de larga escala, onde redução do consumo de energia é uma questão cada vez mais importante. Kessaci *et al.* (2013) propuseram um algoritmo genético multiobjetivo (MO-GA) com vistas a ajudar os provedores a otimizarem três critérios: energia, emissão de gases de efeito estufa e lucro. A arquitetura proposta foca no modelo de serviço *cloud federation* e no modelo de implantação IaaS. De acordo com a avaliação, o GA multiobjetivo proposto melhora, em média, os resultados obtidos principalmente pela heurística. Os resultados mostram que o algoritmo proposto reduz em até 29,4% o consumo de energia, até 26,3% da emissão de CO2, e maximiza até 3,6% o lucro obtido.

O principal objetivo da pesquisa de Somasundaram e Govindarajan (2014) é minimizar o tempo de execução e o custo com base em um ambiente definido para aplicações científicas, como nuvem de computação de alto desempenho ou nuvem de ciência. Os autores projetaram e desenvolveram um *broker* de recursos de nuvem (CLOUDRB) para gerenciar com eficiência os recursos de nuvem e completar tarefas. Eles integraram o CLOUDRB com escalonamento baseado em *deadlines* e com um mecanismo de alocação de recursos baseado em *particle swarm optimization* (PSO) para alocar as solicitações dos usuários aos recursos de um provedor de nuvem de maneira quase ideal. Os autores recriaram experimentos usando o ambiente de programação do Matlab[1]. Os resultados da simulação mostraram a eficácia da solução proposta, minimizando o tempo de conclusão, o custo e a taxa de rejeição de *job* e maximizando o número de *jobs* concluídos com solicitações dentro do prazo e satisfazendo as necessidades do usuário.

Iturriaga *et al.* (2013) concentraram-se em um modelo de negócios para fornecer preços baixos a seus clientes por meio de um *broker* que subtraia recursos de provedores de nuvem IaaS sob demanda. Assim, eles apresentam um novo algoritmo evolucionário híbrido paralelo para resolver o problema da alocação eficiente de uma máquina virtual que solicita recursos pré-reservados de um *cloud broker*, para maximizar o lucro do *broker*. No momento da

[1] https://www.mathworks.com

escrita deste documento, o algoritmo proposto apresentou os melhores resultados em relação aos descritos na literatura. Os resultados revelaram que o algoritmo proposto aumenta a alocação eficiente de uma máquina virtual para 133,8%, e para isso requer apenas 90 s de tempo de execução.

Em 2012, Phan et al. propuseram uma estrutura que move dinamicamente os serviços nos *data centers* da Internet para aumentar o consumo de energia renovável e, ao mesmo tempo, manter o desempenho usando uma *cloud federation* em IaaS. Eles propuseram um algoritmo evolucionário de otimização multiobjetivo para decidir sobre a migração e a disposição de serviço. Os resultados da simulação alcançaram os objetivos do algoritmo proposto, mostrando que ele supera a disposição de serviço baseado em capacidade convencional em relação a objetivos conflitantes.

Rahimi *et al.* (2012) modelam aplicações móveis como um *workflow* de tarefas e apresentam o MAPCloud. Os autores propõe uma solução para decompor de forma ideal o conjunto de tarefas em execução de aplicações móveis em um cliente móvel que usa arquitetura de nuvem em duas camadas, considerando vários QoS, como poder de processamento, preço e atraso. O MAPCloud é uma arquitetura de nuvem hierárquica em camadas que pode ser aproveitada para aumentar o desempenho e a escalabilidade de aplicações móveis. Os autores também propuseram uma heurística eficiente baseada em *simulated annealing* chamada CRAM (*Cloud Resource Allocation for Mobile Applications*) para obter uma solução quase ideal para o problema de alocação de recursos de nuvem em camadas. Os resultados indicam que a heurística CRAM alcança um valor próximo a 84% da solução ótima quando o número de usuários é alto. Também indica uma diminuição no consumo de energia e *delay* em até 32% quando comparado com o uso da nuvem pública, aumenta a escalabilidade em comparação com as nuvens locais, e diminui o preço em 40% se comparado ao uso de apenas uma nuvem pública para aplicações móveis complexas.

Para tratar o escalonamento de tarefas de aplicações *workflow* em sistemas paralelos e distribuídos, Fard *et al.* (2013) propuseram um modelo de precificação e um mecanismo confiável para tarefas de planejamento único. Eles têm por objetivo minimizar custo e *makespan*. Para alcançar tal objetivo, os autores usaram a estratégia de programação biobjetiva (BOSS), um algoritmo baseado em leilão em um ambiente com *multi-cloud* e implementada na camada de *broker*. A seção experimental envolve um extenso estudo realizado usando o simulador GridSim. Os resultados dos algoritmos evolutivos dominam as soluções do algoritmo BOSS em menos de

2% dos casos.

Com o objetivo de reduzir o custo e o tempo de execução das aplicações do consumidor usando IaaS, Heilig *et al.* (2016) abordam o problema de gerenciamento de recursos em nuvem no ambiente *multi-cloud*. Os autores propuseram um algoritmo *efficient biased radom-key genetic*, já que o problema de gerenciamento de recursos em nuvem é um problema de otimização e os consumidores exigem soluções em tempo real e de alta qualidade. Os autores pressupõem que os requisitos de recursos são conhecidos antecipadamente, mas as previsões em tempo real notificam os consumidores se as configurações atuais de recursos precisam ser reajustadas, exigindo, portanto, um apoio contínuo à decisão. No momento da escrita desta tese, os resultados dos experimentos mostraram que a abordagem proposta supera os métodos propostos na literatura sobre um amplo *benchmark* baseado em recursos reais do mercado de nuvens.

Anastasi *et al.* (2017) projetaram e desenvolveram uma abordagem *cloud brokering* que fornece uma solução de implantação otimizada para aplicações baseadas em nuvem através de *multi-cloud*, chamada QBROKAGE. O QBROKAGE usa um algoritmo genético para combinar serviços e recursos de nuvem e procurar encontrar recursos de IaaS para satisfazer os requisitos de QoS de aplicações em nuvem. O QBROKAGE explora apenas as informações que os provedores comerciais provavelmente disponibilizaram para os clientes, como os custos de VM e seus recursos. Os autores realizaram um conjunto de experimentos considerando a aplicação em três camadas e *workflows* de aplicações científicas. Os resultados mostraram que o QBROKAGE pode encontrar soluções quase ótimas, mesmo quando lida com centenas de provedores.

Durante o ciclo de vida, uma aplicação complexa implantada em uma nuvem pública identifica vários fatores que podem motivar a decisão de migrar a aplicação para melhor atender os requisitos. Entre os fatores que podem estimular a migração estão: novos provedores podem entrar no mercado ou mudar suas ofertas, e um serviço pode evoluir e precisar de melhores desempenhos, ou um novo serviço pode aparecer; tais mudanças podem tornar um novo projeto mais lucrativo, ou mesmo necessário. Legillon *et al.* (2015) definem um novo modelo realista para o problema de migração baseado em uma meta-heurística evolutiva multiobjetiva, que é um algoritmo genético que usa operadores específicos em IaaS. O principal objetivo dos autores é reduzir o custo ao mínimo, otimizando três aspectos: custo da VM, simplicidade e custo de migração. Eles concluem que o algoritmo pode encontrar soluções de boa qualidade em todas as instâncias de *benchmark* fornecidas, exceto uma, porque elas não tinham um melhor

score teórico para compará-las com o algoritmo várias vezes testado contra vários conjuntos de cenários de problemas reais.

Govindarajan *et al.* (2016) propuseram um *framework* de gerenciamento de recursos em nuvem distribuído para abordar a escalabilidade na nuvem de computação de alto desempenho. Para isso, o *framework* incorpora uma tabela *hash* distribuída estruturada para evitar o mecanismo de seleção de recursos de tráfego de rede e *particle swarm optimization* (PSO). O mecanismo de escalonamento de recursos baseado em enxames considera o custo e o tempo para escalonar e alocar as solicitações do *job* e os recursos de maneira ideal. Uma avaliação foi realizada para abordagem proposta através de medidas de desempenho, como tempo de resposta e um número bem-sucedido de solicitações de consulta tratadas no sistema proposto e no tradicional. Os resultados da abordagem proposta foram melhores do que do sistema tradicional.

Mennes *et al.* (2016) propuseram um algoritmo genético confiável de disposição de aplicações na nuvem(GRECO), o qual é distribuído para colocar uma aplicação orientada a serviços em uma nuvem híbrida com múltiplas nuvens menores e de diferentes capacidades. O GRECO deve colocar as aplicações nas nuvens de maneira que elas tenham alta disponibilidade. Os resultados dos experimentos mostraram que o tempo de execução é melhor quando ocorre uma pequena queda no desempenho das taxas de disposição de aplicações e adequado para altas taxas de disposição de aplicações.

Outros trabalhos na literatura que abordam o problema de gerenciamento de recursos através da computação evolucionária foram identificado. Jena (2015) aborda o escalonamento de tarefas multiobjetivo utilizando *framework* PSO aninhado. Já Padmaveni *et al.* (2016) usa *hybrid mmetic and particle swarm optimization* para *workflows* científicos multiobjetivos. Zegrari *et al.* (2016) tratam a alocação de recursos com eficiente balanceamento de carga, combinando o algoritmo genético e o algoritmo KHN. Kumar e Patel (2016) abordam o gerenciamento de recursos usando *feed-forward ANN-PSO*. Sheikholeslami e Navimipour (2017) tratam da alocação de serviço usando o algoritmo *multi-objective particle swarm optimization* baseado na distância de aglomeração. Kumar e Kalra (2017) usam *discrete binary cat swarm optimization* para escalonamento de aplicações *workflow*; Alboaneen *et al.* (2017) que propôs a *glowworm swarm optimization* para o escalonamento de tarefas. Kaur e Mehta (2017) tratam o provisionamento de recursos e escalonamento de *workflow* usando o algoritmo *augmented shuffled frog leaping*. Malekloo *et al.* (2018) propuseram uma abordagem de otimização de colônia de formigas (MACO) de múltiplos objetivos, sensível a energia e a QoS para posicionamento e consolidação de VMs.

Por fim, Midya *et al.* (2018) tratam da alocação de recursos e escalonamento de tarefas usando o algoritmo *hybrid adaptative particle swarm optimization* (HAPSO). Todas essas obras usam técnicas novas ou combinadas. Este trabalho concentrou-se em múltiplas nuvens, o que traz mais desafios do que uma única nuvem, como portabilidade e interoperabilidade. Portanto, esses trabalhos não são aqui mais detalhados e comparados a proposta desta tese.

Discussão

A tabela 18 apresenta um resumo das características dos trabalhos descritos nesta seção para melhor analisar como estes trabalhos usam um ambiente de múltiplos provedores de nuvem. Para isto, esta tabela examina sete aspectos: Algoritmo de Computação Evolutiva, Modelo de Serviço, Fator de Seleção, Modelo de Entrega, Perspectiva, Tipo de Aplicação e Objetivo. A tabela 19 mostra a abrangência dos trabalhos descritos nesta seção relacionada às taxonomias propostas nesta tese.

Relacionado ao modelo de serviço, pode-se observar que as soluções só tratam de problemas a nível de IaaS, já que a maioria dos provedores que oferecem os serviços para o nível de PaaS e SaaS são bastante específicos e possuem uma diversidade de serviços. Neste sentido ainda faltam soluções que tratem do problema de gerenciamento de recursos para múltiplas nuvens nos níveis de PaaS e SaaS. Quanto aos fatores de seleção, as soluções propostas usaram, no máximo, três dos seguintes fatores: custo, lucro, tempo de execução, consumo de energia, tempo de escalonamento, de alocação e disponibilidade. No entanto, a maioria utilizou apenas custo e tempo de execução. Porém outros fatores são necessários do ponto de vista de um arquiteto de *software*, como localização na nuvem, tempo de resposta e elasticidade, sendo que esses relacionados apenas ao modelo de serviço IaaS (GUZEK *et al.*, 2015).

Em relação ao modelo de entrega e perspectiva, as soluções propostas são bem distribuídas. O modelo de entrega mostra, além de *cloud federation* e *multi-cloud*, modelos como *science cloud*, *cloud broker*, *cloud hybrid* e HPCC. No contexto desses trabalhos, tais modelos de entrega são considerados uma *cloud federation*. Sobre os objetivos, os trabalhos que se concentram em provedores usam principalmente a *cloud federation* como um modelo de entrega, já que não há interesse em fornecer soluções para gerenciamento de recursos em múltiplas nuvens se não houver nenhum acordo de colaboração entre eles. Da mesma forma, o gerenciamento de recursos em *multi-cloud* é complicado devido à diversidade de recursos e à falta de padrões para APIs, termos, recursos e assim por diante.

TABELA 18. RESUMO DAS CARACTERÍSTICAS DOS TRABALHOS APRESENTADOS NA SEÇÃO A.

Abordagem	Características						
	Algoritmo EC	Modelo de Serviço	Fator de Seleção	Modelo de Entrega	Perspectiva	Tipo de Aplicação	Objetivo
Legillon *et al.* (2013)	GA	IaaS	Custo	Multi-cloud	Usuário ou Provedor	Geral	Disposição de Serviço
Frey *et al.* (2013)	GA	IaaS		Multi-Cloud	Usuário	*Software* Industrial	Migração para a Nuvem
Kessaci *et al.* (2013)	GA Multi-objetivo	IaaS	Consumo de Energia, Emissão de Co2, Lucro	*Cloud Federation*	Provedor	HPC	Disposição de VM
Somasundaram e Govindarajan (2014)	PSO	IaaS	Tempo de Execução, Custo	*Science Cloud*	Usuário	HPC	Alocação de Recurso
Iturriaga *et al.* (2013)	*Parallel Hybrid EA*	IaaS	Profit Broker	*Cloud Broker*	Provedor	Geral	Disposição de VM
Phan *et al.* (2012)	*multi-objective EA*	IaaS	Consumo de Energia	*Cloud Federation*	Provedor	Geral	Disposição de Serviço
Rahimi *et al.* (2012)	*SA optimization*	IaaS	Poder de processamento, Custo, Atraso	*Hybrid Cloud*	Usuário	Aplicação Móvel	Alocação de Recurso
Fard *et al.* (2013)	*Auction-based bi-objective algorithm*	IaaS	*Makespan*, Custo	*Multi-cloud*	Provedor	Aplicação de *Workflow*	escalonamento de Tarefa

Continua na próxima página

. . . continuação

Abordagem	Características						
	Algoritmo EC	Modelo de Serviço	Fator de Seleção	Modelo de Entrega	Perspectiva	Tipo de Aplicação	Objetivo
Heilig *et al.* (2016)	*biased random-key GA*	IaaS	Custo, Tempo de Execução	*Multi-cloud*	Usuário	Geral	Gerenciamento de Recurso
Anastasi *et al.* (2017)	GA	IaaS	Requisitos Funcionais e Não funcionais	*Multi-cloud*	Usuário	Geral	Seleção de Recurso
Legillon *et al.* (2015)	GA	IaaS	Custo, Tempo de Execução	*Multi-cloud*	Usuário	Geral	Migração de Aplicação
Govindarajan *et al.* (2016)	PSO	IaaS	Custo, Tempo de Alocação e de Escalonamento	HPCC	Usuário	HPC	Seleção de Recurso
Mennes *et al.* (2016)	*distributed GA*	IaaS	Disponibilidade	*Hybrid cloud*	Usuário	SBS	Disposição de VM

TABELA 19. RESUMO DOS TRABALHOS SOBRE MCRM USANDO COMPUTAÇÃO EVOLUTIVA RELACIONADO AOS ASPECTOS DAS TAXONOMIAS 4, 6, E 7.

Abordagem	Gerenciamento de Recursos em Múltiplas Nuvens												
	Quando Gerenciar								Como Configurar requisitos			Como Gerenciar	
	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII
Legillon *et al.* (2013)	✘	✘	✔	✘	✘	✘	✘	✘	Estático	Múltiplos	Serviço	Automático	Serviço
Frey *et al.* (2013)	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Kessaci *et al.* (2013)	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Somasundaram e Govindarajan (2014)	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Iturriaga *et al.* (2013)	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Phan *et al.* (2012)	✘	✘	✘	✘	✘	✘	✔	✘	Estático	Único	Serviço	Automático	Serviço
Rahimi *et al.* (2012)	✘	✘	✔	✘	✘	✘	✘	✘	Estático	Múltiplos	Grupo de Tarefas	Automático	Grupo de Tarefas
Fard *et al.* (2013)	✘	✘	✔	✔	✘	✘	✘	✘	Estático	Múltiplos	Tarefa	Automático	Tarefa
Heilig *et al.* (2016)	✔	✔	✘	✘	✘	✔	✘	✘	Estático	Múltiplos	Aplicação	Interativo	Aplicação
Anastasi *et al.* (2017)	✘	✔	✔	✘	✘	✘	✘	✘	Dinâmico	Múltiplos	Aplicação e Serviço	Automático	Aplicação e Serviço
Legillon *et al.* (2015)	✘	✘	✘	✘	✔	✘	✘	✘	Estático	Único	Aplicação	Automático	Aplicação
Govindarajan *et al.* (2016)	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘	✘
Mennes *et al.* (2016)	✘	✘	✔	✘	✘	✘	✘	✘	Estático	Único	Serviço	Automático	Serviço

I:SELEÇÃO DE NUVEM; II:SELEÇÃO DE SERVIÇO NA PRE-IMPLANTAÇÃO; III: DISPOSIÇÃO DE SERVIÇO; IV: DISPOSIÇÃO DE VM; V: MIGRAÇÃO DE NUVEM; VI: SELEÇÃO DE SERVIÇO PÓS-IMPLANTAÇÃO; VII: MIGRAÇÃO DE SERVIÇO; VIII: MIGRAÇÃO DE VM; IX: ESTADO; X: MULTIPLICIDADE; XI: GRANULARIDADE; XII: MODO; XIII: DEFINIÇÃO DO MODO

Considerando o aspecto da aplicação, mais de 50% dos trabalhos propuseram uma solução para um determinado tipo de aplicação e, principalmente, concentram-se no cliente como objetivo. Em relação ao foco, pode-se notar que as soluções propostas se concentram em questões relacionadas aos níveis de IaaS, como posicionamento de serviço, posicionamento de VM, escalonamento de tarefas e migração de aplicações. Portanto, as soluções propostas tornam-se muito mais específicas.

No aspecto do algoritmo evolutivo, pode-se observar que mais de 50% dos trabalhos usaram um algoritmo genético padrão ou algumas de suas variações. Outros algoritmos utilizados são PSO, EA híbrido paralelo, otimização SA e algoritmo bi-objetivo. As soluções propostas empregaram esses algoritmos usaram principalmente o modelo de entrega *cloud federation*. Finalmente, pode-se observar que os resultados dos experimentos são bons e atendem às expectativas.

Além disso, na Tabela 19, pode-se observar oito trabalhos abordando alguns aspectos relacionados às taxonomias propostas. Eles impõem os aspectos para configurar requisitos e de como gerenciar o MCR, portanto os usuários não podem escolher essas configurações.

Surveys na Literatura

Com o objetivo de investigar o que tem sido estudado em relação ao gerenciamento de recursos em múltiplas nuvens da perspectiva de um arquiteto de *software*, apresenta-se aqui o resultado e as principais características dos trabalhos identificados.

As pesquisas foram realizadas nas bases ACM Digital Libray, ScienceDirect e IEEE, e como refinamento foi usado as *strings* de busca apresentadas na Tabela 20 e o período de publicação, entre 2015 e 2018. Como resultado, encontrou-se 92 trabalhos no ACM, 1167 no ScienceDirect e 219 no IEEE. O processo de exclusão foi realizado primeiro por título e por trabalhos repetidos. Depois, os trabalhos tiveram introdução e conclusão examinadas, de modo a excluir trabalhos fora do escopo do trabalho. No final, obteve-se 25 artigos, resumidos na Tabela 21.

Na Tabela 21 é possível observar que apenas um deles foca na perspectiva de um usuário e que três focam no provedor e no usuário. Ainda, pode-se observar que seis artigos se concentram em múltiplas nuvens, mas que apenas um deles aborda o gerenciamento da perspectiva de um usuário. O trabalho que aborda em múltiplas nuvens da perspectiva de um usuário aborda especificamente o gerenciamento de serviços.

TABELA 20. *Strings* DE BUSCA PARA *surveys* DA LITERATURA

ID	Strings de Buscas utilizadas
1	"survey"AND "cloud"AND "resources"AND "management"
2	"state-of-the-art"AND "cloud"AND "resources"AND "management"
3	"taxonomy"AND "cloud"AND "resources"AND "management"
4	"classification"AND "cloud"AND "resources"AND "management"

Além disso, a Tabela 21 pode ser analisada a partir da perspectiva de algoritmos para gerenciamento de recursos, dentre os trabalhos investigados. Apenas dois deles abordam algoritmos de computação evolucionários. Esses dois trabalhos se concentram na perspectiva de um provedor. Portanto, pode-se perceber a falta de trabalhos no que diz respeito ao gerenciamento de recursos da perspectiva de um arquiteto de *software*, focando em múltiplas nuvens e verificando a integralidade das soluções propostas na literatura que atende às demandas de um usuário no gerenciamento de recursos para implantação e execução de aplicações em múltiplas nuvens.

Desafios e Direções futuras para MCRM

Esta seção apresenta os desafios e direções futuras para o gerenciamento de recursos em múltiplas nuvens a partir da perspectiva de um arquiteto de *software*, focando no uso da computação evolucionária.

Nota-se que as obras encontradas se concentram em uma parte específica do gerenciamento de recursos em múltiplas nuvens. Um grande desafio é integrar as soluções para obter um gerenciamento completo de recursos ou mesmo uma solução que aborde todos os aspectos descritos nas taxonomias propostas. Outro desafio é considerar aspectos comuns ao provedor e ao usuário, para que as necessidades de ambos sejam atendidas, como a seleção de serviços.

Em relação ao uso da computação evolucionária no gerenciamento de recursos em múltiplas nuvens, o principal desafio é tornar o algoritmo evolutivo eficiente em um curto tempo de execução, já que um evolutivo tende a se aproximar mais da solução ótima à medida que o tempo tende ao infinito. A tomada de decisão é outro desafio se o algoritmo usado for multiobjetivo, porque a fronteira de Pareto não considera uma solução melhor que a outra, mas que soluções diferentes têm custos/benefícios distintos (DIEDRICH *et al.*, 2008). Associado a este último desafio está o problema em definir quais funções objetivas serão levadas em consideração (pelo menos 2) e por que elas são usadas. Em termos de processamento, uma alternativa seria usar os algoritmos micro-evolucionários que são essencialmente algoritmos

TABELA 21. RESUMO SOBRE GERENCIAMENTO DE RECURSOS EM NUVEM NOS *surveys* DA LITERATURA.

Abordagem	Recursos	Ambiente	Perspectiva	Objetivo	Tipo da Aplicação	Algoritmo
Babu e Rajam (2017)	Geral	*Cloud*	Usuário	Polícia de Escalonamento de Recurso	Geral	-
Hameed *et al.* (2016)	Geral	*Cloud*	Provedor	Técnicas de Alocação de recursos com eficiência de energia	Geral	-
Yousafzai *et al.* (2017)	Geral	*Cloud*	Provedor	Esquema de alocação de recursos	Geral	-
Meriam e Tabbane (2017)	Geral	*Cloud*	Provedor	Algoritmo de Escalonamento de *Job* para nuvem	Geral	Algoritmo de escalonamento de *Job*
Tocze e Nadjm-Tehrani (2017)	*Edge Computing*	*Edge Computing*	Provedor	Taxonomia para gerenciamento de recursos *edge* e revisão de trabalhos	Geral	-
Demirci (2015)	Geral	*Cloud*	Provedor	Eficiência de energia	Geral	*Machine Learning*
Guzek *et al.* (2015)	Geral	*Cloud*	Provedor	Computação evolutiva para gerenciamento de recursos de processamento	Geral	Computação evolutiva
Liaqat *et al.* (2017)	Geral	*Cloud Federation*	Provedor	Gerenciamento de recursos	Geral	-
Ullrich *et al.* (2016)	Geral	*Cloud*	Aplicação	Identificação de similaridades e diferenças entre soluções	Geral	-
Luong *et al.* (2017)	*Cloud Networking*	*Cloud Networking*	Provedor e Usuário	Analisar aplicações de modelos econômicos e de precificação para desenvolver algoritmos adaptativos e protocolos para gerenciamento de recursos em *cloud networking*	Geral	-

Continua na próxima página

... continuação

Abordagem	Recursos	Ambiente	Perspectiva	Objetivo	Tipo da Aplicação	Algoritmo
Gonzalez *et al.* (2017)	Geral	*Multiple Clouds*	Provedor	gerenciamento de recurosos para *workflows* e aplicações científicas em larga escala	*Workflows* e aplicações científicas em larga escala	-
Pandey (2016)	Geral	*Cloud*	Provedor e Usuário	Técnicas de alocação de recursos	Geral	-
Poullie *et al.* (2017)	Geral	*Cloud*	Provedor	Alocação de multi-recursos	Geral	-
Aslam *et al.* (2017)	Geral	*Cloud*	Provedor	Técnicas centrada na coleção de informações para CRM	Geral	-
Costache *et al.* (2017)	Geral	*Cloud*	Provedor	Decisões de projeto de PaaS para analisar diferentes domínios de aplicações	Geral	-
Madni *et al.* (2017)	Geral	Cloud	Provedor e Usuário	Algoritmos e esquema de alocação de recursos	Geral	-
Parikh *et al.* (2017)	Geral	*Cloud*	Provedor	Visão sequencial detalhada no gerenciamento de recursos	Geral	-
Mustafa *et al.* (2015)	Geral	*Cloud* ou *Hybrid*	Provedor	Técnicas de gerenciamento de recursos	Geral	-
Singh e Chana (2015)	Recurso autonômico	*Cloud*	Provedor	Gerenciamento de recurso autonômico	Geral	-
Singh e Chana (2016a)	Geral	*Cloud*	Provedor	Algoritmo de escalonamento de recursos	General	-
Singh e Chana (2016b)	Geral	*Cloud*	Provedor	Provisionamento de recursos	Geral	-

Continua na próxima página

. . . continuação

Abordagem	Recursos	Ambiente	Perspectiva	Objetivo	Tipo da Aplicação	Algoritmo
Mansouri *et al.* (2017)	Armazenamento de dados	*Multiple Clouds*	Provedor	Validar taxonomias e identificar áreas para pesquisas futuras	Aplicação com uso intensivo de dados	-
Mann (2015)	VM	*Hybrid*	Provedor	Alocação de VM	Geral	Algoritmo de Otimização
Rehman *et al.* (2015)	Serviço	*Multi-cloud*	Usuário	Gerenciamento de serviços	Geral	-
Zhan *et al.* (2015)	Geral	*Cloud*	Provedor	Escalonamento de recursos e abordagens evolucionária	Geral	Computação evolutiva

evolutivos com uma população pequena (entre 5 e 7 indivíduos). Estudos mostraram que esta abordagem é muito eficiente em termos de desempenho, tanto no tempo de computação quanto na qualidade da solução (NESMACHNOW *et al.*, 2012). Finalmente, o último desafio trata dos parâmetros usados nos algoritmos. Parâmetros errados podem levar a um algoritmo ruim, que pode ser resolvido de várias maneiras, como por meio do *design* de experimentos, que permite selecionar os parâmetros através de execuções *offline* e com o uso de algoritmos adaptativos ou auto-adaptativos.

Desta forma, pode-se identificar algumas direções futuras para o gerenciamento de recursos em nuvens múltiplas, entre elas abordagens que tratam o problema da perspectiva de um arquiteto de *software*, abordagens que integram aspectos de gerenciamento de recursos da perspectiva de um arquiteto de *software* e as que integram tanto da perspectiva de um arquiteto de *software* como de um provedor. Existem também soluções que propõem o uso da computação evolucionária para aspectos de gerenciamento de recursos que ainda não foram abordados, bem como soluções para aspectos já tratados que utilizam outros algoritmos de computação evolutiva, que são na sua maioria híbridos e alguns deles são mencionados na Seção A. Além disso, ainda existe a necessidade de soluções para verificar em quais aspectos cada algoritmo é mais eficiente. Finalmente, faz-se necessário tratar os desafios inerentes à computação evolucionária, descritos no parágrafo anterior.

APÊNDICE B – ALGORITMOS PRA OS MODELOS UM^2K, UM^2Q E LM^2K

Algoritmo Dinâmico para UM^2K

Algoritmo 1 tem como entrada o conjunto de requisitos da aplicação e o conjunto das capacidades dos provedores disponíveis e como saída um conjunto dos provedores para hospedar cada microsserviço de uma aplicação.

Primeiro, o Algoritmo 1, na linha 4, identifica-se os serviços de um provedor disponível que atendem a todos os requisitos de um microsserviço. Em seguida, classifica-os calculando a fase de escala do SAW, na linha 6, e na linha 7, a fase de ponderação do SAW para cada serviço. Em seguida, na linha 8, os serviços são combinados, de forma que cada combinação tenha todos os serviços requeridos por um microsserviço. O processo é repetido em todos os provedores disponíveis. Em seguida, o *score* é calculado na linha 13 e o custo, na linha 14, para cada combinação. Todo o processo entre as linhas 2 e 13 é repetido para todos os microsserviços de uma aplicação, resultando em um conjunto de combinações candidatas para cada microsserviço (SAMS), cada uma delas tem seu *score* e custo.

Finalmente, das linhas 21 a 29, seleciona-se uma combinação para hospedar um microsserviço dentre as combinações candidatas em SAMS. Para isso, cria-se uma matriz (Sol) usando como linhas cada valor de custo de combinação possível, ou seja, entre 1 e AP.c (limiar para o orçamento de uma aplicação definido por um arquiteto de *software*) e como colunas o custo de cada combinação candidata para todos os microsserviços de uma aplicação. A Matriz Sol é preenchida com os valores de *score* de acordo com o microsserviço ao qual pertence a combinação. A fim de obter as combinações com a maior *score*, de forma que, quando combinadas, não excedam o valor do orçamento da aplicação (AP.c), conforme mapeamento feito para o problema da mochila de múltipla escolha, descrito nas Eqs. 4.17, 4.18, 4.19, 4.20, 4.21, 4.22 e 4.23.

Algoritmo Guloso para UM^2K

Nesta subseção, os Algoritmos 2 e 3 são descritos para o modelo UM^2K usando um algoritmo guloso. O Algoritmo 2 representa o primeiro e segundo níveis enquanto o Algoritmo 3 representa o terceiro nível, pelo qual é possível observar todo o processo de seleção proposto.

O algoritmo 2 tem, como entrada, um conjunto de requisitos de uma aplicação e o

Algoritmo 1: Algoritmo dinâmico para o modelo UM^2K

```
input  : Conjunto dos Requerimentos de uma Aplicação ap
         Conjunto das Capacidades dos Provedores Disponíveis cp
output : Conjunto de Provedores prvdsMs para hospedar cada microsserviço de uma aplicação

foreach ms of the ap do
    ssp ← inicia com conjunto vazio;
    foreach sp of the cp do
        candServ ← discoveryCandServ (ms,sp);
        if (notempty(candServ)) then
            matReq ← sawScalingPh (candServ) ;                  // formulas 4.8 e 4.9
            candServSC ← sawScorePh (candServ,matReq,ap.weight) ;    // formula 4.10
            ssp ← ssp + (sp.number,combMsSp (candServSC)) ;          // formula 4.11
        end
    end
    sms ← inicia com conjunto vazio;
    foreach comb of the ssp do
        combSC ← calculateCombSC (comb) ;                       // formula 4.13
        combCost ← calculateCombCost (comb) ;                   // formula 4.14
        sms ← sms + (comb,combSC,combCost);
    end
    sams ← sams + (ms.number,sms);
end
// O algoritmo entre as linhas 19 e 29 indica a seleção de provedores de
nuvem usando programação dinâmica para MCKP.;
sol ← inicia a matrix com 0;
last ← 0;
foreach sms of the sams do
    for j ← (last + 1) to sizeof(sms) + last do
        for cost ← 1 to ap.c do
            if (combCost[j] > cost) then sol(cost, j) ← maxLine(sol(cost), ms[j − 1].number);
            else if (ms[j].number == 1) then sol(cost, j) ← combSC[j];
            else if (ms[j].number ≠ ms[j − 1].number) then
              sol(cost, j) ← max(maxLine(sol(cost − combCost[j]), ms[j − 1].number) +
              combSC[j], maxLine(sol(cost), ms[j − 1].number));
            else sol(cost, j, ms[j].number) ←
              max(maxLine(sol(cost − combCost[j]), ms[j].number − 1) +
              combSC[j], maxLine(sol(cost), ms[j].number − 1));
        end
    end
    last ← sizeof(sms) + last;
end
```

conjunto das capacidades dos provedores disponíveis e, como saída, um conjunto de provedores para hospedar todos os microsserviços de uma aplicação. Para isso, primeiro, na linha 5, descobre-se os serviços de um provedor que atendem a todos os requisitos de um microsserviço. Em seguida, classifica-os através da fase de escala do SAW na linha 7 e da fase de ponderação do SAW na linha 8. Em seguida, combina-se os serviços, para que cada combinação tenha todos os serviços requeridos por um microsserviço. O processo é repetido em todos os provedores disponíveis. Em seguida, na linha 13, o *score* é calculado e, na linha 14, o custo para cada combinação candidata. Todo o processo entre as linhas 1 e 19 é repetido para todos os microsserviços de uma aplicação, resultando em um conjunto de combinações candidatas em cada provedor para cada microsserviço (SAMS), e cada um delas tem seu *score* e seu custo.

Algoritmo 2: Algoritmo Guloso para o modelo UM^2K - Primeiro e Segundo Níveis

```
input  : Conjunto de requerimentos de uma aplicação ap
         Conjunto das capacidades dos provedores disponíveis cp
output : Conjunto de provedores prvdsMs para hospedar os microsserviços de uma aplicação
sams ← inicia com conjunto vazio ; // lista das combinações candidatas com score e
 custo para todos os microsserviços
foreach ms of the ap do
    ssp ← inicia com conjunto vazio ; // lista das combinações candidatas de um
     microsserviço em todos os provedores
    foreach sp of the cp do
        candServ ← discoveryCandServ (ms, sp);
        if not empty (candServ) then
            matReq ← sawScalingPh (candServ);
            candSerSC ← sawScorePh (candServ,matReq,ap.weight);
            ssp ← ssp + (sp.number,combMsSp (candSerSC));
        end
        sms ← inicia com conjunto vazio ; // lista das combinações candidatas, o
         score e o custo de um microsserviço em todos provedores
        foreach comb of the ssp do
            combSC ← calculateCombSC (comb);
            combCost ← calculateCombCost (comb) ;
            sms ← sms + (comb, combSC, combCost);
        end
        sams ← sams + (ms.number, sms);
    end
end
```

O Algoritmo 3 representa o terceiro nível da solução gulosa para o modelo UM^2K, o qual é baseado no algoritmo descrito por Kellerer *et al.* (2004). Assim, na linha 2, classifica-se as combinações candidatas para cada microsserviço pelo custo da combinação. Essas combinações

candidatas pertencem ao SAMS descrito na Eq.4.15. Em seguida, entre as linhas 4 e 8 e entre as linhas 9 e 13, exclui-se as combinações candidatas dominadas e lp-dominadas para cada microsserviço de acordo com (KELLERER *et al.*, 2004). Portanto, exclui-se a combinação candidata *k* que tem custo maior ou igual ao custo da combinação *j* e *score* menor que o *score* da combinação *j*. Depois, uma solução inicial é criada contendo a combinação com menor custo de cada microsserviço. Assim, entre as linhas 14 e 16, subtrai-se do orçamento o primeiro custo de combinação candidatas de cada microsserviço.

depois de gerar a solução inicial, uma instância do problema da mochila é criada entre as linhas 17 e 27. Para isso, na linha 20, atribui-se ao *SC* a diferença entre o *score* da combinação *j* e o *score* da combinação *j-1* para cada microsserviço. Na linha 21, atribui-se para *Cost* a diferença entre o custo da combinação *j* e o custo da combinação *j-1* para cada microsserviço. Além disso, na linha 22, calcula-se a eficiência incremental; na linha 23, acrescenta-se para *r* uma tupla que contém o identificador do microsserviço *i*, o identificador da combinação candidata *j*, o *score* resultante da linha 20, o custo resultante da linha 21 e a eficiência incremental *e*. Na linha 28, classifica-se em ordem decrescente *r* de acordo com a eficiência incremental.

depois de identificar as combinações com maior custo benefício, a solução final é construída colocando as combinações selecionadas em Z. Para isto, entre as linhas 29 e 33, adiciona-se para *Z* uma tupla que contém o identificador de microsserviço *i*, o identificador de provedor *sms[0].sp*, o identificador de combinação que este caso é 1 e *score* da primeira combinação candidata de um microsserviço. Além disso, entre as linhas 34 e 41, obtém-se as combinações para hospedar cada microsserviço. Para isso, na linha 36, atualiza-se o orçamento com a diferença entre ele e o custo *r[k]*. Em seguida, entre as linhas 37 e 39, compara-se o identificador de microsserviço de *r[k]* com o identificador de microsserviço do item *z*. Depois, atualiza-se o *score* do item *z* com o *score r[k]*. Finalmente, retorna-se *z*, o qual contém todas as combinações de serviços para hospedar todos os microsserviços, e cada combinação pode pertencer a um provedor distinto.

Algoritmo baseado em Cota do Orçamento para UM^2Q

Nesta subseção, o Algoritmo 4 é descrito para o modelo UM^2Q, o qual apresenta o pseudocódigo com mais detalhes de todos os níveis do que o diagrama da Figura 38. O algoritmo possui duas entradas: o conjunto de requisitos dos microsserviços de uma aplicação e o conjunto das capacidades dos provedores de nuvem disponíveis. Ele tem como saída o conjunto dos

Algoritmo 3: Algoritmo Guloso para o modelo UM^2K - Terceiro nível

```
for i ← 0 to sizeof(sams) − 1 do
    sorByCost (sams[i]);
end
for i ← 0 to sizeof(sams) − 1 do
    for j ← 0 to sizeof(sams[i]) − 1 do
        if dominated (sams[i][j + 1], sams[i][j]) then remove (sams[i][j + 1], sams[i]);
    end
end
for i ← 0 to sizeof(sams) − 1 do
    for j ← 0 to sizeof(sams[i]) − 1 do
        if lp-dominated (sams[i][j + 2], sms[i][j + 1], sms[j]) then remove (sams[i][j + 1]);
    end
end
foreach sms of the sams do
    budGet ← budGet − sms[0]cost ;
end
r ← [] ; i ← 0 ; k ← 0 ;
foreach sms of the sams do
    for j ← 1 to sizeof(sms) − 1 do
        SC ← sms[j].SC − sms[j − 1].SC ;
        Cost ← sms[j]cost − sms[j − 1]cost ;
        e ← sms[j] + (sms[j].SC)/(sms[j]cost) ;
        r[k] ← r[k] + (i, j, SC, Cost, e);
        k ← k + 1 ;
    end
    i ← i + 1 ;
end
sortDecreasingByE (r);
z ← [] ; i ← 0 ;
foreach sms of the sams do
    z ← z + (i, sms[0]sp, 1, sms[0].SC) ;
    i ← i + 1 ;
end
k ← 0 ;
repeat
    budGet ← budGet − r[k]cost ;
    foreach itemZ of the z do
        if (r[k][0] == itemZ[0]) then
            aux ← z;
            removeItem(itemZ, aux);
            insertItem(r[k], aux);
            res ← checkReqs(aux);
            if (res == true) then itemZ[3] ← r[k].SC;
        end
    end
    k ← k + 1 ;
until (budGet == 0 or budGet ≤ r[k]cost);
return (z) ;
```

provedores para hospedar todos os microsserviços.

Para alcançar os objetivos do modelo UM^2Q, o Algoritmo 4, na linha 6, descobre os serviços candidatos em um único provedor, no qual um serviço candidato é um serviço de nuvem que atende a todos os requisitos de um microsserviço. Em seguida, nas linhas 8 e 9, ele classifica os serviços candidatos, de acordo com as Eqs. 4.8, 4.9 e 4.10. Depois, na linha 10, combina todos os serviços candidatos necessários para compor um microsserviço de forma que o custo da combinação seja menor ou igual à cota de microsserviço de acordo com as Eqs. 4.25, 4.27, 4.28 e 4.33. Em seguida, na linha 11, ele adiciona todas as combinações candidatas de um provedor a uma lista de combinações candidatas para um microsserviço. Então, ele calcula o *score* da combinação de acordo com a Eq. 4.31 entre as linhas 13 e 17. Ele retorna à combinação com a maior *score* na linha 19. Depois, entre as linhas 20 e 24, verifica se existe mais de uma combinação com a maior *score*, caso exista, verifica qual dessas combinações tem a maior prioridade, de acordo com as preferências de um arquiteto de *software*. Todo o processo é repetido em todos os provedores disponíveis para o mesmo microsserviço. Assim, ao final desse nível, ele tem um conjunto de combinações para um microsserviço, sendo um de cada provedor candidato. Um provedor candidato é um provedor que possui pelo menos uma combinação candidata para compor um microsserviço.

O terceiro nível tem como objetivo selecionar uma combinação para cada microsserviço, então, primeiro, na linha 28, verifica-se dentre os provedores candidatos qual possui o maior *score*. Se mais de um provedor possuir o maior *score*, na linha 29, calcula-se o custo médio da combinação de acordo com as Eqs. 4.34, 4.35 e 4.36. Em seguida, na linha 30, ele verifica as combinações que possuem o custo médio. Nas linhas 31 e 32, se houver mais de uma combinação com o custo médio, ele verifica qual combinação tem a maior *score*. Se houver mais de uma combinação com a maior *score* na linha 33, ela obtém a combinação com a maior prioridade de acordo com as preferências de um arquiteto de *software*. Assim, ao final deste nível, ele tem uma combinação de um provedor para hospedar um microsserviço. Todo o processo é repetido para todos os microsserviços de uma aplicação. Ao final, ele retorna um conjunto de combinações, o qual contém uma combinação para cada microsserviço.

Algoritmo Dinâmico para LM^2K

O algoritmo dinâmico para o modelo LM^2K é descrito nesta subseção, o qual está dividido em duas partes. A primeira parte, ilustrada pelo Algoritmo 5, a qual representa o

Algoritmo 4: Algoritmo para o modelo UM^2Q

```
input  : Conjunto dos requisitos dos microsserviços ap
         Conjunto das capacidades dos provedores cp
output : Conjunto de Provedores prvdsMs para hospedar microsserviços

foreach ms of the ap do
    combLtPr ← inicia com conjunto vazio ;
    combsMS ← inicia com conjunto vazio ;
    combLtMS ← inicia com conjunto vazio ;
    foreach sp of the cp do
        candServ ← discoveryCandServ (ms, sp) ;
        if (not empty(candServ)) then
            matReq ← sawScalPh (candSr);
            candSrSC ← sawSCPh (candSr, matReq, ap.wgt);
            combsMS ← combsMS + combMsSp (candSrSC, quotaMS);
            combLtPr ← combLtPr + (sp.number, combsMS);
        end
        combLtPrSC ← inicia com conjunto vazio;
        foreach comb of the combLtPr do
            combSC ← calculateCombSC (comb);
            combLtPrSC ← combLtPrSC + (comb, combSC);
        end
        combLtPrHg ← initialize with empty set;
        combLtPrHg ← combLtPrHg + combHgtSC (combLtPrSC);
        if (not empty(combLtPrHg)) then
            if (sizeof(combLtPrHg) > 1) then combPrMS ← combHgtPrty (combLtPrHg,
              prty);
            else combPrMS ← combLtPrHg;
        end
        combLtMS ← combLtMS + combPrMS;
    end
    if (not empty(combLtMS)) then
        if (sizeof(combLtMS) > 1) then
            combLtPrHg ← combLtPrHg + combHgtSC ( combLtMS);
            if (not empty(combLtMS)) then if (sizeof(combLtMS) > 1) then avgCost ←
              averageCost (combLtMS);
            combLtMSAgCost ← combAvgCost (combLtMS, avgCost);
            if (sizeof(combLtMSAgCost) > 1) then
                finalCbMS ← finalCbMS + combHgtSC (combLtMS);
                if (sizeof(finalCbMS) > 1) then finalCbMS ← combHgtPrty (combLtMS,
                  prty);
                else finalCbMS ← combLtPrHg;
            end

            else finalCbMS ← combLtMS

        end
    end
    combLtFinal ← combLtFinal + (ms.number, finalCbMS);
end
return (combLtFinal);
```

primeiro e o segundo níveis do modelo para as três soluções: dinâmica, gulosa e bioinspirada em colônia de formigas. A segunda parte representa o terceiro nível do modelo e é ilustrada pelo Algoritmo 6. Além disso, o Algoritmo 5 para a soluções possui como entrada um conjunto de requisitos de uma aplicação, um conjunto das capacidades dos provedores disponíveis, um conjunto dos *links* de comunicação para os serviços de cada provedor e para os provedores disponíveis, e um conjunto contendo o fluxo de execução para as atividades dos microsserviços e para os microsserviços de uma aplicação. Além disso, o Algoritmo 6 possui como saída um conjunto dos provedores para hospedar cada microsserviço de uma aplicação.

Algoritmo 5: Algoritmo para modelo LM^2K - Primeiro e Segundo Níveis

```
input : Conjunto dos Requerimentos de uma Aplicação ap
        Conjunto das Capacidades dos Provedores Disponíveis cp
        Conjunto de Links de Comunicação entre Serviços nos Provedores e entre Provedores lSer
        e lPr
        Conjunto de Fluxo de Execução entre as tarefas dos Microsserviços e entre Microsserviços
        fSer e fPr
output : Conjunto de Provedores prvdsMs para hospedar microsserviços
sams ← inicia com conjunto vazio ; // lista das combinações candidatas com score e
 custo para todos os microsserviços
foreach ms of the ap do
    ssp ← inicia com conjunto vazio ; // lista das combinações candidatas de um
     microsserviço em todos os provedores
    foreach sp of the cp do
        candServ ← discoveryCandServ (ms,sp);
        if (notempty(candServ)) then
            matReq ← sawScalingPh (candServ);
            candSerSC ← sawScorePh (candServ,matReq,ap.weight);
            ssp ← ssp + (sp.number,combMsSp (candSerSC))
        end
    end
    sms ← inicia com conjunto vazio ; // lista das combinações candidatas, score e
     custo de um microsserviço em todos os provedores
    foreach comb of the ssp do
        combSC ← calculateCombSC (comb,lSer,fSer);
        combCost ← calculateCombCost (comb,lSer,fSer);
        sms ← sms + (comb,combSC,combCost);
    end
    sams ← sams + (ms.number,sms);
end
```

O Algoritmo 5, na linha 5, descobre os serviços de um provedor disponível que atendem a todos os requisitos de um microsserviço. Em seguida, nas linhas 7 e 8, ele os classifica usando SAW. Em seguida, na linha 9, ele combina os serviços, de forma que cada combinação

tenha todos os serviços requeridos por um microsserviço. O processo é repetido em todos os provedores disponíveis. Além disso, o Algoritmo 5, na linha 14, calcula o *score* e, na linha 15, o custo para cada combinação. Para calcular os valores do *score* e do custo de cada combinação, o algoritmo considera o fluxo de execução das atividades de um microsserviço, bem como os valores de disponibilidade, tempo de resposta e custo de cada *link* de comunicação entre os serviços necessários para um microsserviço em um mesmo provedor. Todo o processo entre as linhas 2 e 19 é repetido para todos os microsserviços de uma aplicação, resultando em um conjunto de combinações candidatas para cada microsserviço (SAMS), cada um delas tem seu *score* e custo.

O Algoritmo 6 descreve o mapeamento do processo de seleção do modelo LM^2K para o problema da mochila de múltiplas escolhas. Assim, entre as linhas 1 e 33 o algoritmo seleciona uma combinação para hospedar um microsserviço entre as combinações candidatas de um microsserviço em SAMS. Para isso, o algoritmo cria uma matriz (Sol), na qual cada linha contém uma combinação candidata de um microsserviço, e as colunas contêm os orçamentos para uma aplicação sendo que a última coluna contém o valor o limiar do orçamento definido pelo arquiteto de *software*. A Matriz Sol é preenchida com os valores de *score* observando a combinação candidata e o microsserviço que ela pertence, para obter as combinações com maior *score* de forma que, quando combinadas, não excedam o valor do orçamento da aplicação (AP.c).

O Algoritmo 6 difere do Algoritmo 1 no preenchimento da matriz. O Algoritmo 6 considera os valores de disponibilidade, tempo de resposta e custo para os *links* de comunicação entre os provedores para calcular os requisitos de uma aplicação. Nas linhas 9 e 10, bem como nas linhas 19 e 20, o algoritmo calcula a disponibilidade e o tempo de resposta entre duas combinações de dois microsserviços distintos, observando o fluxo de comunicação entre eles, de forma que a soma do valores dos *scores* das duas combinações só é realizado se a disponibilidade, tempo de resposta e custo atenderem às necessidade da aplicação.

Algoritmo Guloso para LM^2K

O Algoritmo 5 representa o primeiro e segundo níveis para as três soluções do modelo LM^2K. O Algoritmo 7 ilustra o terceiro nível do processo de seleção usando um algoritmo guloso para o modelo LM^2K. Assim, na linha 2, as combinações candidatas são classificadas para cada microsserviço pelo custo da combinação. Entre as linhas 4 e 8 e entre as linhas 9 e 13, exclui-se as combinações candidatas dominadas e lp-dominadas para cada microsserviço

Algoritmo 6: Algoritmo dinâmico para o modelo LM^2K - Terceiro Nível

```
sol ← inicia a matriz com 0;
last ← 0;
foreach sms of the sams do
    for j ← (last + 1) to sizeof(sms) + last do
        for cost ← 1 to ap.c do
            if (combCost[j] > cost) then sol (cost,j) ← maxLine (sol (cost),ms [j-1].number);
            else if ms [j ].number == 1 then sol (cost,j) ← combSC [j ];
            else if (ms [j ].number ≠ ms [j- 1].number) then
                (combSC,pr) ← discoveryPrComb (maxLine (sol (cost- combCost [j ]),ms
                  [j-1].number)) ;
                res ← clAvaRT (combSC,pr,combSC [j ],ms [j ].number,lPr,fPr) ;
                if res == true then
                    sol (cost,j) ← max ( combSC + combSC [j ], maxLine (sol (cost),ms
                      [j-1].number));
                end
                else
                    sol (cost,j) ← maxLine (sol (cost),ms [j-1].number);
                end
            end
            else
                (combSC,pr) ← discoveryPrComb (maxLine (sol (cost- combCost [j ]),ms [j
                  ].number - 1));
                res ← clAvaRT (combSC,pr,combSC [j ],ms [j ].number,lPr,fPr);
                if res == true then
                    sol (cost,j) ← max ( combSC + combSC [j ], maxLine (sol (cost),ms [j
                      ].number-1));
                end
                else
                    sol (cost,j) ← maxLine (sol (cost),ms [j ].number - 1);
                end
            end
            ;
            ;
        end
    end
    last ← sizeof(sms) + last;
end
```

de acordo com (KELLERER *et al.*, 2004). Entre as linhas 14 e 16, subtrai-se do orçamento o primeiro custo de combinação candidatas de cada microsserviço. Entre as linhas 17 e 27, uma instância do problema da mochila é criado. Para isso, na linha 20, atribui-se ao *SC* a diferença entre o *score* da combinação *j* e o *score* da combinação *j-1* para cada microsserviço. Na linha 21, atribui-se para *Cost* a diferença entre o custo da combinação *j* e o custo da combinação *j-1* para cada microsserviço. Na linha 22, calcula-se a eficiência incremental; na linha 23, acrescenta-se para *r* uma tupla que contém o identificador do microsserviço *i*, o identificador da combinação candidata *j*, o *score* resultante da linha 20, o custo resultante da linha 21 e a eficiência incremental *e*. Depois, na linha 28, classifica-se em ordem decrescente *r* de acordo com a eficiência incremental.

depois de identificar as combinações candidatas com maior lucro, a solução final é construída colocando as combinações selecionadas em Z. Assim, entre as linhas 29 e 33, adiciona-se para *Z* uma tupla que contém o identificador de microsserviço *i*, o identificador de provedor *sms[0].sp*, o identificador de combinação que este caso é 1 e *score* da primeira combinação candidata de um microsserviço. Entre as linhas 34 e 41, obtém-se as combinações para hospedar cada microsserviço. Para isso, na linha 36, atualiza-se o orçamento com a diferença entre ele e o custo *r[k]*. Entre as linhas 37 e 39, o identificador de microsserviço de *r[k]* é comparado com o identificador de microsserviço do item *z*. Depois, o *score* do item *z* é atualizado com o *score r[k]* se os requisitos dos *links* de comunicação atendem às necessidades de um aplicação. Finalmente, retorna-se *z*, o qual contém todas as combinações de serviços para hospedar todos os microsserviços, e cada combinação pode pertencer a um provedor distinto.

Algoritmo Bioinspirado em Colônia de Formigas para LM^2K

O algoritmo para o modelo LM^2K bioinspirado em colônia de formigas possui duas partes, como as outras soluções para este modelo. A primeira contém o primeiro e segundo níveis, os quais estão descritos no Algoritmo 5. Esta subseção descreve o Algoritmo 8, o qual representa o terceiro nível do processo de seleção para o modelo LM^2K bioinspirado em colônia de formigas, o qual é mapeado para problema da mochila de múltiplas escolhas. O limiar do orçamento é a capacidade da mochila e o feromônio é o lucro.

O algoritmo tem como entrada um conjunto com o fluxo de execução de uma aplicação, um conjunto com as capacidades dos *links* de comunicação entre provedores, e com todas as combinações candidatas para cada um dos microsserviços obtidas do primeiro e

Algoritmo 7: Algoritmo Guloso para o modelo LM^2K - Terceiro nível

```
foreach sms of the sams do
    sorByCost (sms);
end
foreach sms of the sams do
    for j ← 0 to sizeof(sms) − 1 do
        if dominated (sms[j + 1], sms[j]) then remove (sms [j +1]);
    end
end
foreach sms of the sams do
    for j ← 0 to sizeof(sms) − 1 do
        if lp-dominated (sms[j + 2], sms[j + 1], sms[j]) then remove (sms [j +1]);
    end
end
foreach sms of the sams do
    budGet ← budGet − sms[0]cost ;
end
r ← [] ; i ← 0 ; k ← 0 ;
foreach sms of the sams do
    for j ← 1 to sizeof(sms) − 1 do
        SC ← sms[j].SC − sms[j − 1].SC ;
        Cost ← sms[j]cost − sms[j − 1]cost ;
        e ← sms[j] + (sms[j].SC)/(sms[j]cost) ;
        r[k] ← r[k] + (i, j, SC, Cost, e);
        k ← k + 1 ;
    end
    i ← i + 1 ;
end
sortDecreasingByE (r);
z ← [] ; i ← 0 ;
foreach sms of the sams do
    z ← z + (i, sms[0]sp, 1, sms[0].SC) ;
    i ← i + 1 ;
end
k ← 0 ;
repeat
    budGet ← budGet − r[k]cost ;
    foreach itemZ of the z do
        if (r[k][0] == itemZ[0]) then
            aux ← z;
            removeItem(itemZ, aux);
            insertItem(r[k], aux);
            res ← checkReqs(aux);
            if res == true then itemZ[3] ← r[k].SC;
        end
    end
    k ← k + 1 ;
until (budGet == 0 or budGet ≤ r[k]cost);
return (z) ;
```

segundo níveis. Além disso, é configurado um conjunto de feromônios inicias para cada uma das combinações, onde o mesmo está de acordo com a seguinte Eq. B.1. O processo possui `m` iterações, e *nAnts* formigas. Em cada ciclo deste algoritmo, as `nAnts` formigas são usadas para construir soluções individuais.

Os microsserviços pertencentes ao fluxo de execução de uma aplicação são os grupos da mochila. Assim, o primeiro grupo do qual deve ser selecionada uma combinação candidata é o primeiro microsserviço do fluxo de execução. A combinação candidata do primeiro grupo é selecionada aleatoriamente. Em seguida, o feromônio local é atualizado, de acordo com Eq. B.2. Para os outros grupos do fluxo de execução são selecionadas as combinações candidatas que possuem maior probabilidade, de acordo com a Eq. B.4.

Uma formiga termina a construção de uma solução quando a solução contém uma combinação para cada um dos microsserviços do fluxo de execução. Após cada formiga ter construído uma solução, a melhor solução dessa iteração é identificada. Então o feromônio é atualizado de acordo com a melhor solução, de acordo com a Eq. B.3. O algoritmo termina quando o número máximo de ciclos foi executado.

Neste trabalho, todos os valores de feromônios são inicialmente definidos para τ_0. τ_0 é o valor mínimo de todos os valores de feromônios. Assim, na Eq. B.1, AvaMin e AvaMax significam o menor e o maior valor de disponibilidade para uma aplicação, respectivamente. RTMin e RTMax significam o tempo de resposta mínimo e máximo, respectivamente, para uma aplicação. CostMin e CostMax significa o custo mínimo e custo máximo respectivamente. As variáveis, priAva, priRT e priCost, significam, respectivamente, a prioridade para o requisito de disponibilidade, tempo de resposta e custo.

$$\tau_0 = \frac{AvaMin}{AvaMax} * priAva + \frac{RTMin}{RTMax} * priRT + \frac{CostMin}{CostMax} * priCost \tag{B.1}$$

O feromônio local (τ_{ij}) da Eq. B.2 é calculado baseado em τ_0 e ρ, o qual é um parâmetro de decrescimento de τ, $0 < \rho < 1$. O feromônio global (τ) descrito na Eq. B.3 é calculado de acordo com ρ e o *score* da solução.

$$\tau_{ji} = (1-\rho) * \tau_{ji} + \rho * \tau_0 \tag{B.2}$$

$$\tau = (1-\rho) * \tau + \rho * score \tag{B.3}$$

A atratividade de uma combinação depende do feromônio depositado na mesma, e a combinação selecionada em um grupo é aquela que possui maior atratividade. A atratividade de uma combinação é calculada pela Eq. B.4 (LIU *et al.*, 2011). Na Eq. B.4, β é um parâmetro que determina a importância relativa da informação heurística e do nível de feromônio, quanto maior for β maior a influência da informação heurística no processo de seleção; q_0 é um parâmetro que determina a probabilidade de escolher o melhor componente e $0 \leqslant q_0 \leqslant 1$; q é um número aleatório no intervalo [0,1]; η_{ji} é o *score* da combinação selecionada. De acordo com Wei-Neng Chen e Jun Zhang (2009), se $q_0 = 1$, as formigas escolherão apenas o componente com o maior valor, pois q_0 sempre será maior que q, o que faz com que o algoritmo se torne uma busca gulosa. No entanto, se $q_0 = 0$, o algoritmo será mais estocástico e menos convergente.

$$p = \begin{cases} max_{ji}\{\eta_{ji}^{\beta} * \tau_{ji}\} & \text{if } q \leqslant q_0 \\ \frac{\eta_{ji}^{\beta} * \tau_{ji}}{\sum_{ji}\{\eta_{ji}^{\beta} * \tau_{ji}\}} & \text{caso contrário} \end{cases} \tag{B.4}$$

Os detalhes do Algoritmo 8 são descritos a seguir. Primeiro, entre as linhas 3 e 33 todo o processo de seleção é repetido `m` vezes, a fim de escolher a melhor solução encontrada pelas formigas em `m` iterações. Entre as linhas 10 e 22, cada formiga deve construir uma solução em cada iteração. Para isso, entre as linhas 13 e 15, uma formiga deve obter uma combinação candidata de cada um dos microsserviços observando o fluxo de execução da aplicação. Se a combinação a ser selecionada pertença ao primeiro microsserviço do fluxo de execução, a mesma é selecionada de forma aleatória, caso contrário, a combinação deve ser selecionada de acordo com p (Eq. B.4). Na linha 16, o feromônio local é atualizado. Nas linhas 17 e 21, os dados dos *links* de comunicação e da combinação selecionada são armazenados.

Depois que uma formiga encontrou uma solução, entre as linhas 23 e 26, a solução encontrada é comparada com a melhor solução da iteração, observando o custo benefício das soluções (*profit*). Para isso, *profit* deve observar o lucro e o custo de cada combinação pertencente a solução, bem como dos *links* de comunicação. Depois que todas as formigas encontram suas soluções em uma iteração tem-se a melhor solução encontrada para a iteração, e ela é comparada com a melhor solução dentre as demais iterações, observando novamente o custo benefício, ou seja, o *profit*. Ao final, na linha 34, depois de `m` iterações, o algoritmo retorna a melhor solução encontrada.

Algoritmo 8: Algoritmo ACO para o modelo LM^2K - Terceiro Nível

```
solucao sApp ← conjunto vazio ;                          // armazena a solução final
links lApp ← conjunto vazio ;                  // armazena os links entre provedores
repeat
    solucao sAnts ← conjunto vazio;
    links lAnts ← conjunto vazio;
    foreach eachAnt of the nAnts do
        solucao sTerm ← conjunto vazio;
        links lTerm ← conjunto vazio;
        foreach eachTerm of the flowTerm do
            solucao sMS ← conjunto vazio;
            links lMS ← conjunto vazio;
            foreach eachMS of the eachTerm do
                combCandidates ← conjunto combinações candidatas para eachMS;
                if eachMS == primeiro microsserviço then Selecionar uma combinação comb
                  pertencente a combCandidates aleatoriamente;
                else Selecionar uma combinação comb pertencente a combCandidates com
                  probabilidade p;
                updatePheLocal (τ_comb,ρ,τ_0);
                lMS ← { lMS ∪ prvdComb };
                sMS ← { sMS ∪ comb };
            end
            lTerm ← { lTerm ∪ lMS };
            sTerm ← { sTerm ∪ sMS };
        end
        if (profit (sTerm,lTerm) > profit (sAnts,lAnts)) then
            sAnts ← sTerm;
            lAnts ← lTerm
        end
    end
    if profit (sAnts,lAnts) > profit (sApp,lApp)) then
        sApp ← sAnts;
        lApp ← lAnts
    end
    updatePheGlobal (τ,ρ,sApp);
until m;
return (sApp)
```