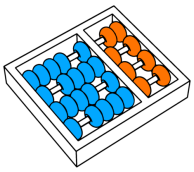
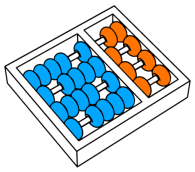


Cristiano Costa Argemon Vieira

"Um Modelo de Escalonamento de Requisições de Máquinas Virtuais em Provedores de IaaS Considerando Diferentes Requisitos dos Usuários"

CAMPINAS
2016



Universidade Estadual de Campinas
Instituto de Computação

Cristiano Costa Argemon Vieira

"Um Modelo de Escalonamento de Requisições de Máquinas Virtuais em Provedores de IaaS Considerando Diferentes Requisitos dos Usuários"

Orientador(a): **Prof. Dr. Edmundo Roberto Mauro Madeira**
Co-Orientador(a): **Prof. Dr. Luiz Fernando Bittencourt**

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutor em Ciência da Computação.

Este exemplar corresponde à versão da Tese apresentada à banca examinadora por Cristiano Costa Argemon Vieira, sob orientação de Prof. Dr. Edmundo Roberto Mauro Madeira.

Assinatura do Orientador(a)

CAMPINAS
2016

Instituto de Computação
Universidade Estadual de Campinas

Um Modelo de Escalonamento de Requisições de Máquinas Virtuais em Provedores de IaaS Considerando Diferentes Requisitos dos Usuários

Cristiano Costa Argemon Vieira

10 de junho de 2016

Banca Examinadora:

- Prof. Dr. Edmundo Roberto Mauro Madeira (*Orientador*)
- Prof. Dr. Alfredo Goldman Vel Lejbman
  Instituto de Matemática e Estatística - USP
- Prof. Dr. Bruno Richard Schulze
  Laboratório Nacional de Computação Científica
- Prof. Dr. Christian Esteve Rothenberg
  Faculdade de Engenharia Elétrica e de Computação - Unicamp
- Prof. Dr. Flavio Keidi Miyazawa
  Instituto de Computação - UNICAMP
- Prof. Dr. Fabio Luciano Verdi
  Universidade Federal de São Carlos (*Suplente*)
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação - UNICAMP (*Suplente*)
- Prof. Dr. Nelson Fonseca
  Instituto de Computação - UNICAMP (*Suplente*)

Abstract

Customers of Cloud providers run applications with different Quality of Service (QoS) requirements. However, the main item treated in SLAs is currently the time of use and the implications for their violation, neglecting other important factors in an SLA. To address this issue, in this thesis, we propose a scheduling model to alocate the requests of multiple users to resources from different IaaS providers in order to decrease the monetary cost without violating the request QoS. We introduce an SLA that considers three important aspects of QoS: reliability, performance and security. We present new strategies based on integer linear program (ILP) formulation and heuristic to compute the scheduling. In addition, we discuss the experimental results obtained from the implementation of our strategy.

Resumo

Usuários de provedores de nuvens executam aplicações com diferentes necessidades de QoS. Contudo, os principais itens tratados nos SLAs atualmente são o tempo de utilização e as implicações quanto a sua violação, negligenciando outros fatores importantes em um SLA. Para tratar esta questão, nesta tese, propomos um modelo de escalonamento para alocar as requisições de múltiplos usuários sobre recursos de provedores públicos de IaaS objetivando o menor custo monetário sem violar a QoS das requisições. Introduzimos um SLA que considera três importantes aspectos de QoS: confiabilidade, desempenho e segurança. Apresentamos novas estratégias para computar o escalonamento. Além disso, apresentamos e avaliamos os resultados experimentais obtidos a partir da implementação da nossa estratégia.

Agradecimentos

Agradeço primeiramente a Deus por eu ter saúde e por eu ter amigos e pessoas especiais em minha vida.

À minha família, principalmente à minha mãe, Mara Regina, pelo incentivo, auxílio e apoio incondicional não só durante o doutorado, mas em todos os momentos da minha vida.

À minha esposa por estar ao meu lado, por constituir uma linda família comigo, e principalmente, pelo carinho e pela dedicação aos nossos filhos Lucas, Caio e Guilherme. Aos meus filhos Lucas, Caio e Guilherme pela paciência e por entenderem a importância desta etapa em nossas vidas.

Ao meu orientador Prof. Edmundo R. M. Madeira e ao meu co-orientador Prof. Luiz F. Bittencourt pela orientação, atenção, disponibilidade e parceria durante o desenvolvimento da tese e dos artigos.

Aos colegas do DINTER com os quais, durante esta caminhada, compartilhei angústias, dificuldades e, finalmente, as alegrias com as conquistas alcançadas no término do doutorado de cada um de nós.

À CAPES por financiar parte deste trabalho.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Com os avanços tecnológicos ocorridos nas últimas décadas, alguns serviços básicos e essenciais são entregues de uma forma completamente transparente. Serviços de utilidade pública como água, gás, eletricidade e telefonia tornaram-se fundamentais para nossa vida diária e são explorados através de um modelo de cobrança baseado no seu uso. As infraestruturas existentes permitem entregar tais serviços em qualquer lugar e a qualquer hora, de forma que possamos simplesmente acender a luz, abrir a torneira ou usar o fogão. O uso desses serviços é cobrado de acordo com as diferentes políticas para o consumidor.

Recentemente, o amadurecimento das tecnologias de virtualização, bem como o crescimento da taxa de transmissão de dados e a popularização da Internet contribuíram para o fortalecimento da computação em nuvem [53] [66]. Este paradigma atrai cada vez mais usuários a consumirem e provedores a oferecerem serviços em nuvem por meio do modelo *pay-as-you-go*. Neste cenário, um crescente número de empresas está adotando serviços de computação em nuvem, tais como: Infraestrutura como um Serviço (IaaS), na qual são oferecidos recursos de infraestrutura sob demanda; Plataforma como um Serviço (PaaS), na qual é implantada uma plataforma sobre os recursos de infraestrutura e fornece um alto nível de *Application Programming Interfaces* (APIs) específicas; e Software como um Serviço (SaaS), na qual é oferecida uma implementação de aplicações específicas utilizando a capacidade da nuvem [65].

Os usuários terceirizam suas demandas computacionais para os provedores públicos e pagam de acordo com sua utilização. Especialmente, considerando serviços de infraestrutura, usuários alugam capacidade de processamento a partir de provedores de IaaS (Amazon EC2[1], FlaxiScale [2], Google Compute Engine[3], ElasticHosts[4], CloudSigma[5] e Joyent-

[1]Elastic Cloud Computing - hwwps://aws.amazon.com/ec2
[2]FlexiScale Cloud Comp and Hosting - http://www.flaxiscale.com
[3]Google Compute Engine - https://cloud.google.com/compute/
[4]ElasticHosts - http://www.elastichosts.com/
[5]CloudSigma - https://www.cloudsigma.com/

Cloud[6]) sob a forma de máquinas virtuais (VMs - *Virtual Machines*) onde os usuários podem implantar suas aplicações. Além disso, os provedores de IaaS oferecem serviços de comunicação com balanceamento de carga para distribuição automática do tráfego de entrada em várias instâncias de VMs; e armazenamento de objeto que pode ser utilizado como uma extensão do armazenamento da instância de VM.

Existem inúmeros objetivos complexos intrínsecos aos pontos de vista do usuário e do provedor na comercialização de VMs através do modelo *pay-as-you-go*. Fundamentalmente, o provedor objetiva encontrar uma disposição ótima para as VMs para fornecer seus serviços. Em geral, computar uma disposição ótima é um problema complexo e demanda um tempo computacional que, por vezes, é inviável visto que o provedor deve computar a disposição de maneira *on-line*. Uma ótima escolha considera, por exemplo, termos do *Service Level Agreement* (SLA), consumo de energia, desempenho, isolamento e *Customer Satisfaction Level* (CSL) [10, 64, 17, 48, 11] e objetiva, principalmente, aumentar o lucro do provedor. Por outro lado, sob o ponto de vista do usuário, o objetivo é beneficiar-se da computação em nuvem obtendo alta disponibilidade, redução de custos, balanceamento de carga e melhoria na tolerância a falhas [38][22, 37, 54, 33, 6, 20]. A seleção dos recursos depende das suas necessidades.

Nesta tese, abordamos o cenário de uma instituição departamental dividida em unidades (como uma universidade com vários departamentos, faculdades e campus) onde os usuários/pesquisadores de cada unidade possuem uma demanda por processamento para implantar seus serviços computacionais e utilizam recursos de provedores de IaaS para fazer face a essa demanda. A principal contribuição nesta tese é um modelo de escalonamento que recebe requisições de vários usuários pertencentes à instituição e computa o escalonamento destas requisições sobre um conjunto de VMs de provedores públicos de IaaS. Duas formulações de Programação Linear de Inteiro (PLIs)) são propostas para computar o escalonamento.

Os serviços dos usuários requerem diferentes requisitos de confiabilidade, desempenho e segurança. A confiabilidade está relacionada ao modelo de cobrança com o qual os provedores denotam como os usuários serão cobrados pelas VMs que utilizarem[6][7][34]. O desempenho de uma VM reflete diretamente no custo do aluguel da VM e no tempo de execução de um serviço implantado sobre a VM. Atualmente, muitos tipos de capacidade de processamento (quantidade de núcleos dos processadores, quantidade de memória e outros) são oferecidos pelos provedores. Já o isolamento contribui para obter um ambiente mais seguro. Vários usuários distintos, pertencentes a mesma instituição, utilizam recursos do mesmo provedor. O compartilhamento da comunicação entre as VMs de diferentes usuários pode levar a um ambiente com pouca segurança. Isto possibilita vários tipos de vulnerabilidades, tais como o uso mal intencionado da rede [49]. Várias características

[6]JoyentCloud - https://www.joyent.com/

devem ser levadas em consideração durante o escalonamento das VMs para o usuário executar suas aplicações.

As características são definidas em um SLA oferecido pelo provedor de IaaS. À medida em que são consideradas mais características no SLA, o usuário consumidor da nuvem encontra mais dificuldades em escolher o melhor conjunto de recursos para atender suas necessidades e reduzir os custos, pois a quantidade de combinações de alocação de aplicações e recursos com diferentes desempenhos e custos é extremamente alta. Este problema é tratado na literatura como sendo um problema de escalonamento. Em sua forma geral, o escalonamento de serviços em um conjunto de recursos é um problema NP-Completo [43]. Em razão disso, durante os anos, uma variedade de heurísticas têm sido desenvolvidas na tentativa de se obter um bom balanceamento entre tempo de execução, complexidade e qualidade do escalonamento [30].

Por implantar os serviços sobre vários provedores ao invés de utilizar apenas um, os usuários podem obter benefícios tais como redução de custo, balanceamento de carga e melhor tolerância a falhas e também evitar ficar preso a um único provedor. Neste contexto, várias plataformas como *Brokers* [28][13][12][25][59] têm sido propostas na literatura para oferecer o serviço de descoberta e a alocação de recursos sobre múltiplos provedores de IaaS ajudando os usuários neste processo de decisão. Além disso, o *broker* oferece uma interface uniforme para gerenciar as VMs com funcionalidades como: implantar, monitorar, terminar VMs, dentre outras, com independência de tecnologia de um provedor específico. Desta forma, o *broker* deve utilizar um *software* adaptador para fornecer uma interface para múltiplos provedores que traduz estas operações para a API específica de cada provedor. Um exemplo comercial de *Broker* é o *CompuNext*[7] que personaliza e publica serviços de provedores públicos de IaaS (AWS[8], Azure[9], Softlayer[10]) e SaaS (O365[11], Salesforce[12], Box[13]).

O objetivo deste trabalho é propor um modelo de escalonamento para alocar requisições de usuários de uma instituição sobre múltiplos provedores objetivando minimizar o custo monetário para a instituição e localizar os melhores recursos que atendem aos requisitos solicitados pelos usuários.

Consideramos três principais aspectos relacionados às características das necessidades dos usuários: (1) diferentes requisitos de confiabilidade; (2) diferentes requisitos de processamento; e (3) diferentes requisitos de segurança. Pela combinação destas carac-

[7]https://www.computenext.com/platform/enterprise-cloud-brokerage/
[8]https://aws.amazon.com/
[9]https://azure.microsoft.com
[10]http://www.softlayer.com/
[11]https://www.microsoftstore.com/
[12]https://www.salesforce.com/
[13]https://www.box.com/

terísticas, nossa proposta permite determinar a melhor maneira de atender às necessidades do usuário do sistema de escalonamento. Não classificamos o sistema de escalonamento como um *broker*, pois, ele não oferece uma interface com funções que podem ser traduzidas para a API específica de cada provedor, como um *broker* deve fornecer.

Como contribuições parciais para compor o modelo de escalonamento que estamos propondo, apresentamos:

(i) Um modelo conceitual de mapeamento das necessidades do usuário: esse modelo permite mapearmos as necessidades dos usuários sobre os serviços oferecidos pelos provedores de IaaS;

(ii) Um novo modelo de cobrança denominado *time-slotted reservation*(TS): esse modelo de cobrança melhora a utilização de instâncias de VMs do tipo RE;

(iii) Um mapeamento conservador das necessidades do usuário: é uma possibilidade de mapeamento na qual a QoS das requisições do usuário não são violadas;

(iv) Um mapeamento flexível das necessidades do usuário: é uma outra possibilidade de mapeamento a qual flexibiliza a utilização de VMs do tipo RE e SP para diminuir o custo do escalonamento. Contudo, pode produzir situações com violação da QoS das requisições dos usuários por utilizar instâncias de VMs do tipo SP;

(v) Uma PLI para computar o escalonamento utilizando o mapeamento conservador: formulamos uma PLI com restrições de mapeamento para garantir a QoS das requisições.

a formulação de PLI utiliza o mapeamento conservador para computar o escalonamento

(vi) Uma estratégia baseada em redundância para diminuir as situações onde ocorrem violações de QoS do mapeamento flexível: a estratégia baseada em redundância permite evitar as situações de violação de QoS duplicando as requisições e executando-as concomitantemente;

(vii) Um algoritmo heurístico baseado no grau de utilização de uma VM para computar a redundância: esta heurística verifica a quantidade de requisições com possibilidade de violação de QoS que utiliza este parâmetro para computar a redundância; e

(viii) Uma PLI baseada no grau de locação de VMs para computar a redundância: esta PLI computa a redundância objetivando diminuir o custo da locação das VMs que serão utilizadas na redundância.

Realizamos simulações dos algoritmos apresentados a fim de avaliarmos os resultados dos escalonamentos.

Esta tese está organizada como descrito a seguir: no Capítulo 2, introduzimos as definições e os conceitos básicos relativos à computação em nuvem. Em seguida, no Capítulo 3, descrevemos os trabalhos relacionados a *Brokers*, bem como outras estratégias para realizar o escalonamento e posicionamos nossas contribuições na literatura. No Capítulo 4, apresentamos uma modelagem proposta para o escalonamento. A definição do problema de escalonamento abordado nesta tese e os algoritmos propostos para resolvê-lo são apresentados e avaliados no Capítulo 5. Em seguida, no Capítulo 6, descrevemos a conclusão e o planejamento para trabalhos futuros.

Capítulo 2

Conceitos Básicos e Definição do Problema

Neste capítulo, apresentamos as definições e os conceitos básicos utilizados nesta tese. Na Seção 2.1, definimos o modelo de computação em nuvem e sua arquitetura, suas características, bem como o modelo de serviços e de implantação. Em seguida, na Seção 2.2 caracterizamos as VMs oferecidas pelos provedores de IaaS considerando a maneira como são cobradas pela utilização. Na Seção 2.3 apresentamos uma caracterização das necessidades do usuário. As definições e as notações específicas utilizadas nos algoritmos propostos são descritas no momento em que os algoritmos são apresentados.

2.1 Computação em Nuvem

Computação em nuvem é uma tendência recente de tecnologia cujo objetivo é proporcionar serviços de Tecnologia da Informação (TI) sob demanda com pagamento baseado no uso [65][9]. Com isso, os usuários estão movendo seus dados e aplicações para a nuvem. Na computação em nuvem os recursos de TI são fornecidos como um serviço, permitindo aos usuários acessarem os serviços sem a necessidade de conhecimento sobre a tecnologia utilizada. Assim, os usuários e empresas passaram a acessar os serviços sob demanda e independente de localização, o que aumentou a quantidade de serviços disponíveis.

Várias definições foram propostas para o termo Computação em Nuvem[16]. Vaquero et al. [52] realizaram uma comparação entre mais de 20 definições. Uma das definições mais utilizadas foi produzida pelo *National Institute of Standards and Technology* (NIST)[39]:

Cloud Computing é um modelo de acesso à rede de forma sob demanda, conveniente e úbiqua a um conjunto compartilhado de recursos computacionais configuráveis (rede,

armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e disponibilizados com um mínimo esforço de gerenciamento ou interação com o provedor.

Nesta tese, consideramos a definição do NIST que explana, na nossa opinião, os principais aspectos da computação em nuvem. O modelo de computação em nuvem descrito pelo NIST é composto por cinco características essenciais, três modelos de serviço e quatro modelos de implantação. Essas características e modelos de computação em nuvem são detalhados nas próximas seções.

2.1.1 Características Essenciais

Um sistema baseado em computação em nuvem contém várias características que o torna atrativo para os usuários, tais como:

1. Serviços sob-demanda: Um consumidor pode requisitar e utilizar recursos computacionais, tais como, tempo de processamento, armazenamento e comunicação para suprir suas necessidades automaticamente sem intervenção humana com cada provedor de serviço.

2. Acesso amplo à rede: Os recursos estão disponíveis na rede e podem ser acessados por diferentes dispositivos: celulares, *tablets*, notebooks e PCs;

3. Arcabouços de recursos: Os recursos computacionais disponibilizados pelos provedores estão em um arcabouço para servir a vários consumidores utilizando um modelo de múltipla locação, com recursos físicos e virtuais distribuídos dinamicamente e redistribuídos de acordo com as necessidades dos usuários. Existe uma sensação de transparência na qual o consumidor geralmente não tem controle ou não sabe sobre a real localização dos recursos fornecidos, mas, é possível especificar uma localização maior como: um país, um estado ou um datacenter. Exemplos de recursos incluem: armazenamento, processamento, memória e largura de banda.

4. Elasticidade: Está relacionada à capacidade de aumentar ou diminuir os serviços fornecidos, em alguns casos, de maneira automática para escalar rapidamente de acordo com a demanda. Para o consumidor, os recursos disponíveis, muitas vezes, parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e em qualquer tempo.

5. Serviço suficiente: Sistemas de nuvem controlam e otimizam o uso dos recursos, aproveitando automaticamente uma capacidade medida em algum nível de abstração adequado para o tipo de serviço (por exemplo armazenamento, desempenho,

largura de banda e usuários ativos). A utilização dos recursos pode ser monitorada, controlada e reportada fornecendo transparência para ambos os provedores e consumidores do serviço utilizado.

2.1.2 Modelo de Serviços

O paradigma de computação em nuvem utiliza um modelo de negócios orientado a serviços. Os recursos oferecidos são fornecidos sob demanda [65]. Os serviços de nuvens oferecidos podem ser agrupados em três categorias: Infraestrutura como Serviço, Plataforma como Serviço e Software como Serviço. Descrevemos as três categorias a seguir:

- Infraestrutura como Serviço(IaaS): O provedor de IaaS é parte responsável por prover toda a infraestrutura necessária para outras categorias de serviço, tais como, a PaaS e o SaaS. Seu principal objetivo é tornar mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir sistemas operacionais e aplicativos. Um provedor de IaaS possui algumas características, tais como uma interface única para administração da infraestrutura, API (*Application Programming Interface*) para interação com *hosts*, *switches*, balanceadores, roteadores e o suporte para a adição de novos equipamentos de forma simples e transparente. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento e aplicativos implantados, e, eventualmente, seleciona componentes de rede, tais como *firewalls*. O termo IaaS se refere a uma infraestrutura computacional baseada em técnicas de virtualização de recursos de computação. Esta infraestrutura pode escalar dinamicamente, aumentando ou diminuindo os recursos de acordo com as necessidades das aplicações. Do ponto de vista de economia e aproveitamento do legado, ao invés de comprar novos servidores e equipamentos de rede para a ampliação de serviços, pode-se aproveitar os recursos ociosos disponíveis e adicionar novos servidores virtuais à infraestrutura existente de forma dinâmica.

- Plataforma como Serviço(PaaS): Um provedor de PaaS oferece uma infraestrutura de alto nível de integração para implementar e testar aplicações na nuvem. O usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistemas operacionais ou armazenamento, mas tem controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. São fornecidos sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de softwares. Em geral, os desenvolvedores dispõem de ambientes escaláveis, mas eles têm que

aceitar algumas restrições sobre o tipo de software que se pode desenvolver, desde limitações que o ambiente impõe na concepção das aplicações até a utilização de banco de dados do tipo chave-valor, ao invés de banco de dados relacionais. Do ponto de vista do negócio, uma PaaS permitirá aos usuários utilizarem serviços de terceiros, aumentando o uso do modelo de suporte no qual os usuários se inscrevem para solicitações de serviços de TI ou de resoluções de problemas pela Web. Com isso, é possível descentralizar uma certa carga de trabalho e responsabilidades nas equipes de TI das empresas.

- Software como Serviço(SaaS): Corresponde à capacidade de fornecer aplicações implantadas sobre a infraestrutura da nuvem para os consumidores utilizarem. As aplicações são acessadas por diversos dispositivos clientes através de interfaces, tais como navegador web ou aplicação em dispositivo móvel. O consumidor não gerencia ou controla as características da infraestrutura da nuvem, incluindo comunicação, servidores, sistemas operacionais, armazenamento ou mesmo capacidades individuais da aplicação, com a possível exceção de configurações específicas do usuário da aplicação.

De acordo com as três categorias de serviços descritas anteriormente, é inteiramente possível que um provedor de SaaS ou PaaS implante sua nuvem sobre um provedor de IaaS. Na prática é mais fácil para um provedor de IaaS oferecer os serviços também de PaaS, como é feito pela Google, Salesforce e Azure [5] [65]. Isso é significante pelo fato do provedor de PaaS oferecer serviços de escalabilidade, confiabilidade e segurança aos desenvolvedores.

2.1.3 Modelos de Implantação

Muitas questões devem ser consideradas quando optamos por implantar os serviços de uma empresa em uma nuvem. Alguns provedores de serviços estão mais interessados em diminuir os custos operacionais para os clientes (provedores de nuvem pública), enquanto outros estão mais interessados em oferecer alta confiabilidade e segurança (provedores de nuvem privada). Portanto, existem diferentes tipos de nuvens:

- Nuvem pública: No modelo de implantação público, a infraestrutura de nuvens é disponibilizada para o público em geral, sendo acessada por qualquer usuário que conheça a localização do serviço. Neste modelo de implantação não podem ser aplicadas restrições de acesso quanto ao gerenciamento de redes, e menos ainda, aplicar técnicas de autenticação e autorização.

- Nuvem privada: No modelo de implantação privado, a infraestrutura de nuvem é utilizada exclusivamente por membros de uma organização, sendo esta nuvem local

ou remota, e administrada pela própria empresa ou por terceiros. Neste modelo de implantação são empregadas políticas de acesso aos serviços. As técnicas utilizadas para prover tais características podem ser em nível de gerenciamento de redes, configurações dos provedores de serviços e utilização de tecnologias de autenticação e autorização. Um exemplo deste modelo é o cenário de uma universidade e seus departamentos. A universidade pode estar interessada em disponibilizar serviços para seus departamentos e outros órgãos desta instituição não devem ter acesso a esses serviços.

- Nuvem híbrida: No modelo de implantação híbrido, existe uma composição de duas ou mais nuvens, que podem ser privadas ou públicas e que permanecem como entidades únicas ligadas por uma tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações.

- Nuvem comunitária: A infraestrutura da nuvem é fornecida para uso exclusivo de uma comunidade específica de consumidores de uma organização que têm preocupações comuns relacionadas aos requisitos de segurança, missão, política e outras. Ela pode ser proprietária, gerenciada e operada por uma ou mais organizações na comunidade.

Selecionar o melhor modelo de implantação depende das necessidades dos negócios do usuário. Por exemplo, aplicações de computação extensiva são melhor implantadas sobre nuvens públicas pois apresentam uma melhor relação de custo benefício [8]. Em alguns casos, o modelo de nuvem híbrida pode ser mais interessante para uma empresa. Este tende a ser um dos modelos mais utilizados por empresas de médio porte [5].

2.2 Caracterização das VMs Oferecidas por Provedores Públicos de IaaS

O modelo de comercialização de software e serviços de tecnologia tem se modificado na última década. Tradicionalmente, a comercialização era feita no modelo em que o usuário paga apenas uma vez pelo uso ilimitado. No modelo de negócios baseado em nuvem, o consumidor paga ao provedor somente o que foi consumido. Este modelo é denominado "pague pelo que usar"(*pay-as-you-go*). Considerando serviços oferecidos por provedores de IaaS, instâncias de VMs são alugadas para usuários e são cobradas pelo tempo de utilização.

Utilizamos nesta tese o termo *modelo de cobrança* como sendo a especificação do SLA que descreve as características e a maneira com a qual uma VM será cobrada pelo provedor.

Na prática, podemos classificar os modelos de cobrança em duas categorias: modelo de custo fixo e modelo de custo variável. Os modelos de cobrança de custo fixo mais utilizados são o *on-demand* (OD) e o *reserved* (RE). O custo pago pelos recursos em um modelo de cobrança de custo fixo é determinado no SLA e não é alterado com o tempo. Desta forma instâncias de VM que obedecem o modelo de cobrança OD ou RE estão disponíveis para o usuário por um determinado custo. Contudo, para ter o direito de utilizar instâncias RE, as quais apresentam custo menor comparado ao das instâncias OD, o usuário tem que pagar uma taxa inicial para reservar a instância por um período longo de tempo (por exemplo: um ano ou três anos). Por outro lado, as instâncias de custo variável, tais como a baseada em leilão denominada *spot* (SP) oferecida pala Amazon EC2, podem ter o preço variado de acordo com algum critério do provedor. O critério mais utilizado atualmente é a quantidade de recursos ociosos no provedor [4][48]

Considerando os três tipos de modelos de cobranças (OD, RE e SP), os modelos OD e RE são interrompidos apenas se o usuário efetuar a interrupção do funcionamento da VM. Por outro lado, o modelo SP, com um custo menor, pode ser interrompido também pelo provedor considerando o custo ofertado pelo usuário e o custo corrente estipulado pelo provedor. Desta forma, a comercialização de instâncias de VMs SP oferece um custo menor, mas também, com potencial interrupção em sua execução. Mesmo com a baixa QoS oferecida, instâncias SP estão sendo cada vez mais utilizadas [48][4][12].

Muitos provedores oferecem múltiplos tipos de VMs, cada uma com diferentes capacidades e com diferentes custos monetários. Face a esta expressiva disponibilidade de serviços oferecidos por uma variedade de provedores de IaaS, atualmente, a questão mais importante não é decidir em utilizar ou não os serviços de provedores de IaaS, mas sim, como escolher o melhor recurso disponível. Deste modo, quando os consumidores utilizam provedores públicos de IaaS, eles podem ter como objetivo diminuir o custo sem diminuir a qualidade necessária. Para alcançar isto, os usuários concordam com os termos especificados no SLA oferecido pelos provedores no momento em que alugam as VMs.

2.3 Caracterização das Necessidades do Usuário de IaaS

Os usuários solicitam VMs de provedores de IaaS para executar suas aplicações. Desta forma, definimos esta necessidade do usuário por uma única VM como sendo uma *requisição de VM*. O usuário pode realizar inúmeras requisições para executar aplicações com diferentes níveis de requisitos de QoS. Conforme descrito anteriormente, IaaS é o modelo que mais requer conhecimento do usuário que utiliza o serviço. A utilização de recursos computacionais, tais como, confiabilidade, desempenho, armazenamento, co-

municação, segurança oferecidos por provedores de infraestrutura, requer que o usuário especifique detalhadamente sua necessidade. Desta forma, o usuário poderá escolher os melhores recursos para satisfazer suas necessidades. Nesta tese, consideramos três características referentes às necessidades dos usuários que são abordadas durante o escalonamento: confiabilidade, desempenho e segurança. As características são descritas a seguir.

1. Confiabilidade: Esta característica está relacionada à confiabilidade do serviço considerando se o serviço pode sofrer atraso ou ser interrompido. Considerando o tempo de execução como o principal parâmetro de confiabilidade, as necessidades do usuário podem conter critérios elegíveis como, prioridade, garantia de execução e cumprimento do prazo. Utilizando uma adequada caracterização desta demanda, é possível propor um escalonamento que atenda com mais qualidade às necessidades do usuário referentes aos requisitos da requisição.

2. Desempenho: Esta característica está relacionada ao requisito de desempenho das requisições do usuário. Mensurar corretamente esta característica é importante para encontrar o recurso que melhor atende a demanda solicitada. Recursos subestimados, inevitavelmente prejudicarão o desempenho da aplicação. Por outro lado, recursos super-estimados podem resultar em instâncias ociosas e possível custo desnecessário.

3. Segurança: A preocupação por segurança em nuvem é uma característica que pode influenciar na escolha pela utilização ou não de um serviço. Vários contextos relacionados à segurança devem ser abordados. Nesta tese, consideramos esta característica relacionada à execução de VMs ou serviços de maneira compartilhada ou isolada.

As três características descritas são abordadas no Capítulo 4, onde explicamos a modelagem do sistema de escalonamento.

Capítulo 3

Trabalhos Relacionados

A utilização de provedores de IaaS tem sido cada vez mais comum. A diversidade entre as características dos recursos oferecidos por diferentes provedores, tornam a seleção de recursos um desafio para os usuários. Além de auxiliarmos os usuários na seleção de recursos, utilizamos VM do tipo *spot* para diminuir o custo do escalonamento. Pelo fato das instâncias *spot* poderem ser interrompidas a qualquer momento por aumento do custo ou pela necessidade do provedor em utilizar o recurso físico que hospeda a instância em questão, algumas técnicas podem ser aplicadas para melhorar a qualidade das estratégias que utilizam as instâncias *spot*: a) verificar o melhor lance a ser dado por uma instância reduzindo chance de preempção e pagando o menor valor; b) *checkpointing* para garantir a continuação da execução das aplicações; e c) redundância para reiniciar a aplicação no momento em que houver uma preempção da VM. Além disso, alternativas à configuração atual do modelo de cobrança *spot* têm sido propostas[44] [12]. Em geral, essas técnicas são implementadas em *brokers* e oferecidas aos usuários de serviços de nuvens. Esta é uma maneira de centralizar as funcionalidades que são oferecidas aos usuários e auxiliam os mesmos a ajustarem suas necessidades aos diferentes serviços oferecidos pelos provedores de nuvem.

Fizemos uma análise sobre os principais trabalhos existentes na literatura que estão relacionados com nossa proposta nos seguintes contextos:

- Seleção dos melhores recursos de provedores de IaaS utilizando *Brokers*;
- Utilização de técnicas de tolerância a falhas; e
- Estratégias para estimar o valor de um lance ofertado para uma instância *Spot*.

3.1 Seleção dos Melhores Recursos de Provedores de IaaS

A utilização de provedores públicos de nuvem para estender a capacidade ou para suprir uma necessidade dos recursos tem se configurado como uma maneira popular para alcançar elasticidade no poder de processamento dos recursos da infraestrutura local para diminuir o tempo de conclusão dos serviços. Desta forma, a elasticidade tem sido amplamente explorada nos últimos anos [29][27].

Atualmente, os provedores de IaaS oferecem diversas zonas de disponibilidades para os usuários implantarem suas aplicações. Recursos oferecidos por zonas diferentes podem ter QoS distintos, dada a heterogeneidade da respectiva estrutura física. Unuvar et al. [51] introduziram uma estratégia de predição para identificar e selecionar a zona de disponibilidade de um provedor que maximiza o nível de satisfação do usuário. São considerados como requisitos: a quantidade de recursos que o usuário solicita; (ii) um critério de QoS objetivo que o usuário quer alcançar; (iii) restrições sobre possíveis serviços (como localidade e vazão); e (iv) tipo de instâncias do usuário. Este trabalho difere-se do nosso pois não explora a utilização de VM do tipo *spot* para executar instâncias OD e considera apenas requisições de um usuário.

Uma PLI foi apresentada por Genez et al. [19] para resolver o problema de escalonamento para serviços com dependência em SaaS/PaaS com um SLA de dois níveis. Eles utilizaram instâncias de modelo RE e OD nas simulações. Similarmente, mas para um SLA de apenas um nível, Bittencourt et al. [8] propuseram um algoritmo para escalonar *workflows* o qual considera custo e tempo de conclusão, para selecionar VMs de provedores públicos de IaaS para expandir a capacidade computacional de uma infraestrutura local. Valerio et al. [14] formularam o problema de escalonamento de recursos onde muitos provedores de SaaS precisam de VMs de provedores de IaaS. A formulação tem um esquema de aprovisionamento de duas fases. Na primeira fase, o provedor de SaaS determina o número de instâncias reservadas e on-demand utilizando uma técnica de otimização. Na segunda fase, o provedor de SaaS compete, oferecendo lances para as instâncias que são instanciadas utilizando a capacidade subutilizada do provedor de IaaS utilizando uma estratégia baseada no *Stackelberg game*. Shen et al. [46] propuseram uma estratégia híbrida de escalonamento que minimiza o custo do aluguel por fazer uso de instâncias OD e RE. Eles formularam uma PLI para realizar o escalonamento. Os quatro trabalhos consideram apenas VMs do tipo OD e RE. Além de não considerarem VM do SP no escalonamento, diferem-se do nosso trabalho por considerarem dependência entre as requisições.

Assunção et al. [6] investigaram os benefícios que uma organização pode alcançar utilizando provedores de nuvem computacional para aumentar a capacidade de processamento de uma infraestrutura local. Eles avaliaram o custo de sete estratégias de alocação utiliza-

das por uma organização para diminuir o tempo de resposta das requisições dos usuários. Guo et al. [21] propuseram o *Seagull*, um *framework* para executar aplicações em *cloud bursting* no contexto empresarial. *Seagull* é projetado para alocar aplicações inteiras em nuvens, sempre que necessário. Não trata várias requisições ao mesmo tempo. Somasundaram et al. [47] introduziram um framework para o gerenciamento de recursos em grade e nuvem. O framewark é implantado sobre o Middleware Globus Toolkit [1]. Inicialmente, é feita a descrição semântica dos recursos. Em seguida, é feito um plano de busca por recursos. Os recursos encontrados são classificados de acordo com a similaridade com a demanda. Esta estratégia é integrada a um SLA baseado em um mecanismo que negocia as requisições dos usuários para satisfazer suas necessidades de QoS.

Yao et al. [62] propuseram um serviço de *broker* para ajudar usuários a minimizarem o custo de computação por restringir o tempo de execução de conjuntos de tarefas utilizando instâncias OD, RE e SP a partir de um provedor. O serviço associa cada conjunto de tarefas com restrição de tempo de execução, e sempre tenta utilizar instâncias mais baratas por computação para manter o menor custo. O *broker* considera os modelos de cobrança como um requisito do usuário. Embora permita concorrência entre as requisições para aumentar o paralelismo, não se preocupa com a segurança. Desta forma, é impossível tratar requisições de múltiplos usuários. Similarmente, Lucas-Simarro et al. [36] apresentaram uma arquitetura de *broker* em módulos que pode trabalhar com diferentes estratégias de escalonamento para implantar serviços sobre múltiplas nuvens, baseado sobre diferentes critérios de otimização e diferentes restrições de usuários considerando instâncias OD, RE e SP. A principal diferença em relação ao nosso trabalho é o fato de que ambos [62] e [36] não permitem uma flexibilização no mapeamento das requisições dos usuários para os serviços oferecidos pelos provedores. Além disso, consideram o escalonamento apenas de um usuário.

Genaud et al. [18] descreveram doze estratégias baseadas em heurísticas para o escalonamento *on-line* com o objetivo duplo onde a solução minimiza o custo e o tempo de execução das requisições. Este trabalho difere-se da nossa proposta, principalmente, pelo fato de que objetivamos minimizar o custo e evitar violações de QoS. Dado que as requisições possuem um tempo de execução previamente definido. Além disso, utilizamos uma PLI para computar o escalonamento. Malawski et al. [37] formularam uma programação linear mista (PLIM) para tratar do problema de alocar tarefas sobre múltiplos provedores. O problema de alocação de recursos é definido utilizando uma técnica de otimização baseada em uma linguagem de modelagem denominada AMPL[1].

Também considerando tempo e custo de escalonamento, Fard et al. [15] introduziram um modelo econômico para computar o escalonamento de *workflows* científicos. Uma breve contextualização geral sobre *workflows* científicos foi apresentada por Bittencourt

[1]A Mathematical Programming Language

et al. [7] considerando algoritmos de escalonamento sobre nuvens híbridas.

Wieder et al. [61] apresentou o Conductor que ajuda os usuários a escolherem serviços de nuvem para implantar computação de MapReduce em nuvem diminuindo o custo monetário e o tempo de conclusão. Também para aplicações MapReduce, Palanisamy et al. [42] propuseram um modelo de serviço de nuvem, o qual computa a melhor alocação para as tarefas a fim de aproximar uma otimização de recursos global. O sistema utiliza um método de prazos consistentes, atrasa a execução de certas tarefas e ajuda alcançar uma otimização global e reduzir o custo. Li et al. [33] investigaram a seleção de serviços em modelos de cobrança dinâmicos, tais como instâncias spots, avaliando um conjunto de cinco algoritmos. Os algoritmos são baseados em heurísticas e em otimização combinatória. Os algoritmos foram avaliados pela implantação de um conjunto de serviços através de múltiplos provedores. Os experimentos sugerem que um algoritmo guloso, em alguns casos, produz boas soluções mais rapidamente quando comparado com outros algoritmos.

Houidi et al. [22] apresentaram um *broker* utilizando *switch OpenFlow* para alocar recursos e serviços a partir de múltiplos datacenters para requisições de usuários feitas através do *broker* objetivando reduzir o custo para o cliente. Utilizam uma formulação de PLIM para computar o escalonamento. Oprescu and Kielmann [41] apresentaram o BaTS, um escalonador de tarefas para escalonar grandes conjuntos de tarefas sobre múltiplos provedores com diferentes custos e desempenho de CPU, minimizando o tempo de conclusão enquanto respeita um limite superior no custo. A estratégia não requer informações sobre o tempo de conclusão das tarefas e assume que toda tarefa pode ser interrompida e reescalonada mais tarde. Tanto [22] como [41] modelam o problema considerando que existe uma comunicação entre as requisições. Nossa proposta considera que as requisições independentes uma das outras.

Existe também o Aeolus [2], um software de gerenciamento de nuvem escrito em Ruby[2] o qual roda sobre sistemas Linux. Como um software de gerenciamento, Aeolus permite usuários escolherem entre nuvens públicas, privadas e hibridas utilizando uma biblioteca denominada *DeltaCloud*. Aeolus é muito similar a nossa proposta. Contudo, as principais diferenças entre ambos são as seguintes: o Aeolus não trata os modelos de cobrança; não inclui um escalonador para optimizar a alocação; não roda como um simulador, desta forma, todas as decisões devem ser feitas considerando o mundo real.

Tordsson et al. [50] apresentaram um *broker* como componente do gerenciador de infraestrutura virtual OpenNebula[3] que otimiza a alocação de recursos sobre múltiplas nuvens. Similarmente, Leitner et al. [31] apresentaram uma estratégia para computar o escalonamento de requisições sobre provedores de IaaS objetivando minimizar o custo e a violação de QoS. Os dois trabalhos diferem-se da nossa proposta principalmente por

[2]http://rubyonrails.org/

não considerar as necessidades dos usuários. Nosso trabalho mapeia as necessidades dos usuários para evitar as violações de QoS.

Wang et al. [60] propuseram um serviço de *broker* em nuvem que reserva um grande conjunto de instâncias a partir de provedores e serve aos usuários com menor custo. O *broker* é composto por estratégias dinâmicas para realizar a reserva de instâncias com o objetivo de minimizar o custo utilizando instâncias do tipo RE. Não existe concorrência entre as requisições de diferentes usuários. O principal objetivo é minimizar o custo explorando a maximização da utilização de instâncias do tipo RE. Nossa proposta também utiliza esta estratégia, contudo, propomos um modelo de cobrança para mantermos uma independência os utilizados pelos provedores e os em nosso modelo de escalonamento. Além disso, consideramos também instâncias OD e SP.

Considerando o aspecto de segurança, Marcon et al. [49] introduziram uma estratégia de alocação que aumenta a segurança dos recursos de rede compartilhados entre várias aplicações. Eles agrupam requisições de usuários confiáveis na mesma infraestrutura virtual. Os autores apresentaram uma PLI para computar o escalonamento onde o alto nível de segurança e isolamento é obtido por escalonar cada grupo sobre diferentes infraestruturas virtuais.

Os principal objetivo dos trabalhos descritos nesta seção é a escolha dos melhores recursos para fazer face à necessidade dos usuários. Nosso trabalho complementa os trabalhos descritos anteriormente pelo fato de compartilhar o mesmo objetivo e ainda considerar as características das necessidades do usuário. Isso contribui para computar escalonamentos mais eficientes e eficazes.

3.2 Utilização de Técnicas de Tolerância a Falhas

Quando os consumidores escolhem utilizar instâncias de VMs do tipo SP para executar aplicações é necessário implementar técnicas de tolerância a falhas, tais como redundância e *checkpointing*. Voorsluys and Buyya [59] apresentaram uma estratégia de alocação de recursos que trata o problema de executar tarefas de computação intensiva sobre instâncias de VMs do tipo SP. A solução apresentada utiliza mecanismos que estimam o custo e o tempo de execução das tarefas. Em seguida, utiliza uma técnica de tolerância a falha baseada na duplicação de tarefas antes de realizar o escalonamento. Para cada tarefa, é criada uma réplica. A tarefa original e sua réplica são escalonadas utilizando a mesma política de escalonamento. Contudo, são obrigatoriamente, escalonadas em provedores distintos. Esta estratégia diminui a probabilidade de uma tarefa falhar por conta de uma preempção. Este trabalho é similar ao nosso. Contudo, possui duas grandes diferenças: (1) utilizamos VM do tipo RE para executar a réplica; e (2) consideramos diferentes necessidades do usuário.

Khatua and Mukherjee [28] propuseram uma estratégia de *checkpointing* em um *framework* de aprovisionamento de recursos que aumenta a confiabilidade da execução ao mesmo tempo em que reduz o custo monetário significativamente. Di et al. [13], projetaram um algoritmo adaptativo para otimizar o impacto do *checkpointing* considerando vários custos tais como os com atraso por *checkpointing*/reinício no contexto de computação em nuvem. Quando utilizam a técnica de *checkpointing* em nuvem, uma preocupação deve ser o custo monetário que a utilização da técnica produz, pois pode inviabilizar a utilização de uma instância SP. Dawoud et al. [12] propuseram um novo modelo de cobrança denominado *Elastic Spot Instances* (ESIs) onde ao invés de terminar abruptamente a instância SP, o provedor diminui a capacidade de processamento da instância para diminuir o custo dela. A utilização desta estratégia permite diminuir o atraso com o *checkpointing* em instâncias SP. Jung et al. [25] propuseram uma estratégia de *checkpointing* baseada na migração de VMs interrompidas.

Assim como os trabalhos descritos nessa seção, nosso trabalho utiliza VMs do tipo SP para diminuir o custo monetário relacionado ao aluguel de recursos de provedores de IaaS. Contudo, utilizamos uma estratégia de redundância de requisições para garantir que uma requisição não tenha sua QoS violada no momento em que uma VM do tipo SP é interrompida. A estratégia de redundância é inovadora pois utiliza a concorrência em VMs para garantir a QoS e diminuir o custo monetário.

3.3 Estratégias para Estimar o Valor de um Lance Ofertado por uma Instância Spot

Além de adotar técnicas de tolerância a falhas, o usuário precisa pagar pelas instâncias SP um valor que minimize o custo e não cause uma preempção, a qual produziria uma baixa confiabilidade à execução. Desta forma, é importante estimar o melhor valor do lance a ser pago por uma instância SP. A Amazon publica o valor de suas instâncias SP mas não informa como ele é determinado. Por este motivo, muitos trabalhos analisam as informações do histórico de valores para estimar um custo para as instâncias SP. Baseado no histórico de preços da Amazon Javadi et al. [24], Yehuda et al. [4], Yi et al. [63] e Kaminski and Szufel [26] propuseram estimar melhor o valor das instâncias SP. O primeiro, apresentou um modelo estatístico que ajusta o valor para o preço por hora do dia e por dia da semana. O segundo, realizou uma engenharia reversa sobre como os valores que são gerados e propuseram um modelo matemático que gera valores consistentes com o histórico de valores apresentados pela Amazon. O terceiro, comparou vários esquemas de *checkpointing* em termos de custo monetário e de melhoria de tempo de conclusão. Além disso, avaliaram esquemas que aplicam métodos de predição, baseados no histórico, para

os valores de instâncias *spot* e mostraram como utilizar a migração para melhorar o tempo de conclusão da tarefa, mesmo com preempções, e ainda assim, reduzir os custos. Também considerando o histórico, o quarto trabalho apresentou um algoritmo para otimização de execuções de simulações em VMs SP. A estratégia leva em conta fatores como mensurar os lances para diferentes instâncias e alternar entre elas, o atraso de inicializar uma instância, o tempo real para terminar uma instância e outros.

Lu et al. [35] propuseram uma solução para o aprovisionamento de recursos dinamicamente para executar aplicações em larga escala utilizando instâncias *spot* e *on-demand*. A estratégia é baseada em configurar instâncias de *backup* utilizando instâncias *on-demand* e analisar os lances para instâncias *spot*. A solução trata requisições de um único usuário. Também utilizando apenas instâncias *spot* e *on-demand*, outros três trabalhos foram propostos por Leslie et al. [32], Huang et al. [23] e Menache et al. [40]. Leslie et al. [32] propuseram um *framework* para tratar o problema de alocação de recursos e escalonamento de tarefas com o objetivo de diminuir o custo, cumprimento dos prazos de execução e confiabilidade. Huang et al. [23] propuseram uma ferramenta para usuários para minimizar as despesas ao executar aplicações em nuvens enquanto satisfaz os prazos para conclusão. A ferramenta automaticamente determina qual a quantidade e o tipo da melhor instância a ser utilizada. Embora o trabalho utilize instâncias *spot* para diminuir o custo financeiro, ele não apresenta técnicas para tratar a preempção das instâncias. Um algoritmo de aprendizagem *on-line* para alocação de recursos para tratar o dilema entre o custo de computação e desempenho utilizando instâncias *on-demand* e *spot* foi introduzido por Menache et al. [40]. O algoritmo dinamicamente adapta a alocação de recursos por aprender a partir do desempenho sobre a execução das tarefas enquanto analisa o histórico dos preços de instâncias *spot* e as características da carga de trabalho.

Sharma et al. [45] projetaram uma plataforma de nuvem que revende recursos adquiridos por plataformas nativas de IaaS. A estratégia usa uma mistura de instâncias *spot* e *on-demand* para fornecer alta disponibilidade e garantia de que servidores *on-demand* tenham um custo baixo. Utilizam migração de VM para diminuir o custo e aumentar a confiabilidade. Existem plataformas comerciais que revendem recursos adquiridos a partir de provedores de IaaS, tais como PICLOUD [3] and Heroku[4].

Os trabalhos descritos nesta seção tentam prever o valor de uma instância SP a ser ofertado ao provedor. Isso é importante para diminuir o custo e evitar preempções. Nosso trabalho considera os três modelos de cobrança mais utilizados atualmente: OD, RE e SP. A quantidade de VMs do tipo SP utilizada pelo modelo de escalonamento que estamos propondo serve como um parâmetro para ajustar a qualidade do escalonamento. Por este motivo, o modelo não prevê o custo a ser pago por uma instância SP. Contudo, utiliza

[3]http://www.multyvac.com/

[4]https://devcenter.heroku.com/

uma técnica de redundância para garantir a qualidade das requisições do usuário.

3.4 Considerações Sobre os Trabalhos Relacionados

Embora muitos trabalhos consideram os modelos de cobrança, poucos deles consideram os três modelos mais utilizados atualmente por provedores (OD, RE e SP) no escalonamento. A Tabela 3.4 apresenta uma comparação entre os trabalhos descritos anteriormente e o nosso trabalho.

Trabalho	**Confiab./ Modelo de cobrança**	**Desemp.**	**Segur.**	**Usuário**
Assunção et al. (2010), Guo et al. (2012), Fard et al.(2013) Somasundaram et al.(2014)	OD			1
Unuvar et al.(2014)	OD	SIM		1
Wieder et al.(2012), Li et al.(2013), Houidi et al.(2011) Oprescu and Kielmann(2010), Voorsluys and Buyya(2012) Khatua and Mukherjee(2013),Di et al.(2013) Dawoud et al.(2012), Jung et al. (2013)	SP			1
Javadi et al.(2013), Yehuda et al.(2011) Yi et al.(2012), Kaminski and Szufel(2015)	SP	SIM		1
Genez et al. (2012), Bittencourt et al.(2010) Valerio et al.(2013), Shen et al.(2013)	OD - RE			1
Yao et al.(2014), Lucas-Simarro et al. (2013) Genaud et al.(2011), Malawski et al.(2013)	OD - RE - SP			1
Tordsson et al.(2012), Leitner et al.(2012)	OD - RE - SP	SIM		1
Marcon et al.(2013)		SIM	SIM	n
Wang et al. (2013)	RE			1
Huang et al.(2013), Menache et al.(2014) Lu et al.(2013), Leslie et al.(2013)	SP - OD	SIM		1
Nossa proposta	OD - RE - SP	SIM	SIM	*n*

Podemos observar que o modelo de cobrança mais utilizado é o OD, por conta da alta disponibilidade de instâncias desse tipo. É interessante considerar diferentes modelos de cobrança no contexto de existir diferentes categorias de confiabilidade. Além disso, os trabalhos existentes na literatura consideram apenas algumas das necessidades de QoS dos usuários separadamente. Não contemplam em uma única solução a alocação de recursos considerando diferentes modelos de cobrança com diferentes requisitos de qualidade de serviço e redução do custo monetário.

Nesta tese, introduzimos um modelo de escalonamento que aloca as requisições de usuários de uma instituição sobre múltiplos provedores objetivando o menor custo para a instituição sem violar a necessidade de QoS da requisição dos usuários. Consideramos três principais aspectos relacionados às características das necessidades dos usuários: (1) diferentes requisitos de confiabilidade; (2) diferentes requisitos de processamento; e (3) diferentes requisitos de segurança. Pela combinação destas características, nossa proposta permite determinar a melhor maneira de atender às necessidades do usuário da instituição.

Capítulo 4

Um Modelo de Escalonamento

Como descrito anteriormente, estamos considerando que vários usuários precisam encontrar os melhores recursos, provenientes de vários provedores, para executar suas aplicações. Estamos propondo nesta tese, um sistema de escalonamento que considera as demandas dos usuários e seus requisitos e auxilia na localização dos recursos que melhor atendem aos requisitos solicitados pelos usuários. Neste capítulo, apresentamos na Seção 4.1 a modelagem geral do sistema de escalonamento. Em seguida, propomos uma modelagem para a caracterização das necessidades do usuário considerando: os requisitos de confiabilidade (Seção 4.2); os requisitos de desempenho (Seção 4.3); e os requisitos de segurança (Seção 4.4).

4.1 A Modelagem do Sistema de Escalonamento

O sistema recebe um conjunto de requisições $\mathcal{R}^+$ e computa um escalonamento S utilizando as VMs de provedores públicos de IaaS. A modelagem é composta por três camadas: aplicação, negócios e infraestrutura. A interação entre as três camadas subsidia o processo completo de escalonamento. A Figura 4.1 apresenta a arquitetura do sistema de escalonamento. A seguir, descrevemos cada uma das camadas.

- Camada de Aplicação: A camada de aplicação corresponde a uma interface entre o usuário e o modelo. Por meio desta interface, são submetidas todas as requisições dos usuários, denotadas pelo conjunto $\mathcal{R}^+$, contendo a quantidade e as características dos serviços que necessitam.

- Camada de Negócios: A camada de negócios é composta por dois módulos: Gerenciamento de SLA e Escalonador. O módulo de gerenciamento de SLA é responsável por realizar a adequação das características das requisições demandadas pelos usuários às características dos serviços oferecidos pelos provedores. Desta

forma é possível incluirmos novas características no SLA do modelo mesmo que os serviços oferecidos pelos provedores não possuam tais características, como por exemplo, novos modelos de cobrança, segurança e outros. O módulo escalonador realiza o escalonamento do conjunto $\mathcal{R}^+$ considerando as adequações realizadas pelo módulo de gerenciamento de SLA e utilizando os recursos de VMs dos provedores de IaaS monitorados pela camada de infraestrutura. Descreveremos no próximo capítulo algumas estratégias para realizar o escalonamento.

- Camada de Infraestrutura: A camada de infraestrutura é responsável por manter informações necessárias ao gerenciamento e monitoramento dos provedores de IaaS utilizados durante o escalonamento.

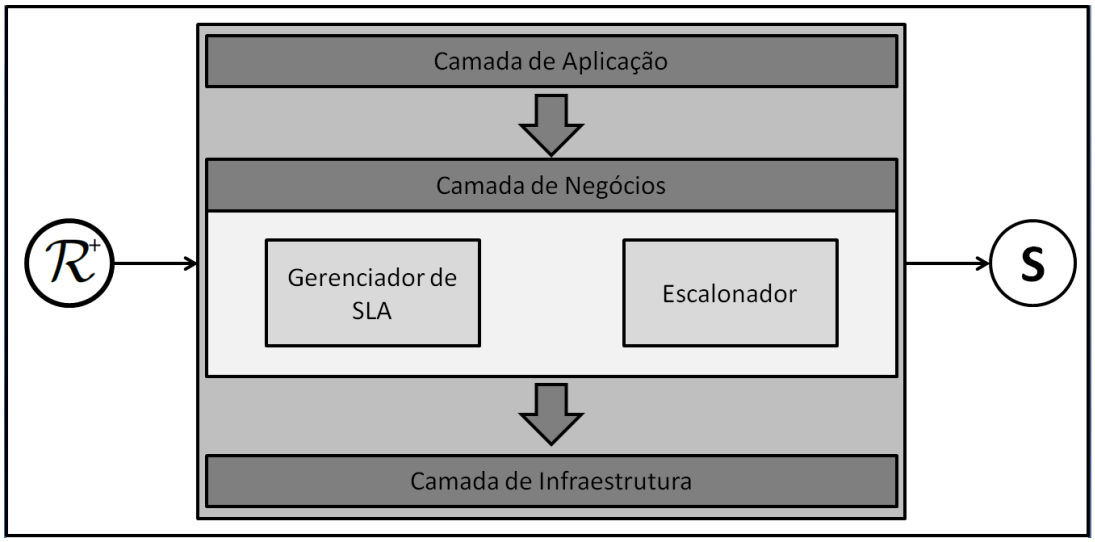


Figura 4.1: A modelagem do sistema de escalonamento

Como mencionado anteriormente, o principal objetivo desta tese é auxiliar o usuário na difícil tarefa de encontrar os melhores recursos para atender suas necessidades. Neste sentido, o modelo de escalonamento que propomos considera três características que são comuns à grande parte dos usuários de nuvens: confiabilidade, desempenho e segurança. Os usuários devem informar estes requisitos durante a solicitação do serviço do provedor. Quanto melhor é a qualidade desta informação, maior é a possibilidade de encontrar recursos com o menor custo e com a garantia de que seus requisitos serão atendidos.

A seguir, apresentamos a modelagem conceitual para tratar cada um dos três requisitos discutidos anteriormente.

4.2 Modelagem dos Requisitos de Confiabilidade

Esta característica está relacionada à confiabilidade do serviço considerando se o serviço pode sofrer atraso ou ser interrompido. Considerando o tempo de execução e a disponibilidade, classificamos o nível de confiabilidade da requisição do usuário em três categorias como apresentada na Tabela 4.1[].

Tabela 4.1: Categorias de Confiabilidade.

Categoria	**Momento de Início**	**Interrupção**
Requisição de Tempo Fixo (FTRx)	imediato	não permitida
Requisição de Tempo Flutuante (FTRt)	pode não ser imediato	não permitida
Requisição de Tempo Variável (VTR)	pode não ser imediato	permitida

Como descrito na Tabela 4.1, na primeira categoria de confiabilidade (FTRx), a requisição deve iniciar imediatamente e a VM não pode ser interrompida durante a execução. Na segunda categoria (FTRt), a requisição pode não iniciar imediatamente, mas, uma vez iniciada, ela não pode ser interrompida. E finalmente, a requisição na terceira categoria (VTR) pode não iniciar imediatamente e pode ser interrompida, isto é, a execução das requisições nesta categoria pode ser fragmentada em partes menores. Por meio da classificação proposta, é possível desenvolver um escalonamento que atenda com mais qualidade as necessidades do usuário referentes à qualidade de serviço da requisição.

4.2.1 Modelo Conceitual de Mapeamento

Para a PaaS suportar a diversidade existente entre as necessidades do usuário e as características do serviço oferecido pelos provedores, propomos um modelo conceitual de interação entre estas informações, como apresentado na Figura 4.2 [54]. Os avanços e as transformações tecnológicas acontecem rapidamente e podem influenciar negativamente em um modelo estático que não seja capaz de se ajustar a tais transformações. Neste sentido, como pode ser observado na Figura 4.2, o modelo que estamos propondo possui um nível (nível 1 do SLA) para absorver tais transformações. Além disso, vinculamos ao modelo dois níveis de SLA. O SLA de nível 1 correspondente ao sistema de escalonamento e o SLA de nível 2 correspondente aos provedores, respectivamente, denotados por Ω^1 e Ω^2.

O nível de Categorias de de Confiabilidade da Requisição contém as 3 categorias de confiabilidade propostas na Tabela 4.1, (FTRx, FTRt e VTR), as quais são mapeadas para algum modelo de cobrança (SLA de nível 1) pertencente ao nosso sistema de escalonamento. Em seguida, é mapeado para um modelo de cobrança de algum provedor

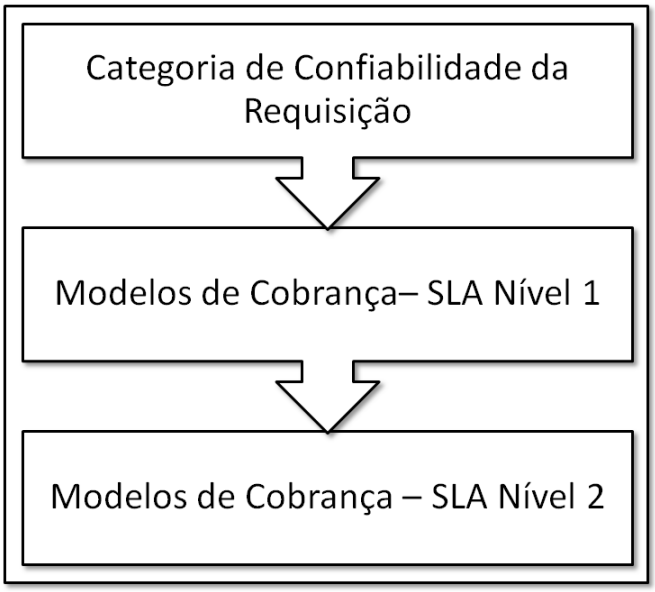


Figura 4.2: Modelo conceitual de mapeamento

público (SLA de nível 2). Desta forma temos a interação entre o usuário e o sistema de escalonamento e em seguida, a interação entre o sistema e os provedores de IaaS. Os modelos de cobrança pertencentes a Ω^2 são OD, RE e SP. Podemos incluir novas características no sistema de escalonamento sem influenciar diretamente no SLA de nível 2 do modelo conceitual, o que permite que novos modelos de cobrança sejam propostos sem interferir nos modelos atualmente disponíveis nos provedores de IaaS. Na próxima subseção propomos um novo modelo de cobrança e descrevemos quais modelos pertencem ao SLA de nível 1 (Ω^1).

4.2.2 Um Novo Modelo de Cobrança

Os modelos de cobrança mais utilizados atualmente são: *On-Demand*, *Reserved* e *Spot*. O modelo de cobrança RE, como descrito anteriormente, exige o pagamento de uma taxa inicial para reservar a instância de VM. Uma vez reservada, o usuário obtém um desconto no momento em que utilizá-la. Esta estratégia pode ser vantajosa para o usuário apenas se ele tem a expectativa de utilizar a VM por um longo tempo. Caso contrário, o custo por unidade de tempo pode ser maior comparado à utilização de uma instância OD. Por exemplo, a Figura 4.3 apresenta cenários onde o consumidor utiliza 10%, 20%, 40%, 60%, 80% e 100% do tempo de uma instância do tipo RE em um e três anos, considerando os preços praticados pela Amazon EC2[1]. Um ano de uso de uma instância OD tem um custo

[1]Amazon - http://aws.amazon.com/ec2/pricing em Abril/2015

de R$ 1.051,20, enquanto que 100% de uso de uma instância RE tem o custo de R$644,22, incluindo a taxa inicial de reserva. Por outro lado, uma utilização de apenas 25% do tempo de uma instância RE tem um custo de R$368,98. O mesmo uso em uma instância OD produz um custo de R$ 262,80. Desta forma, uma baixa utilização de uma instância RE tem um custo mais alto do que uma instância OD. Além disso, é possível observar, por meio da linha "Limite econômico"na Figura 4.3 que, neste cenário, o consumidor deve utilizar mais de 40% de uma instância RE em um ano, e mais de 20% em três anos para que o aluguel de uma instância RE seja uma melhor opção do que uma instância OD.

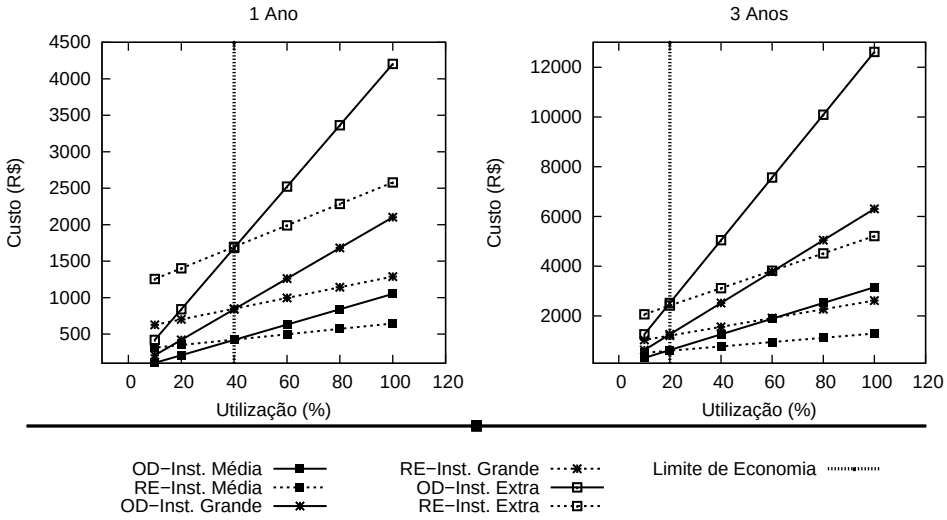


Figura 4.3: Comparativo entre o custo de instâncias OD e RE em um e três anos.

Esta análise mostra que o usuário pode reduzir os custos aumentando a utilização das instâncias RE. Neste sentido, propomos um novo modelo de cobrança denominado "Time-Slotted Reservation - TS". Onde o usuário escalona a requisição em janelas de tempo. Neste modelo, as requisições não podem ser interrompidas. Isto possibilita melhor utilização das instâncias RE, minimizando o custo por unidade de tempo. Adicionamos este modelo de cobrança no conjunto Ω^1, bem como os modelos OD e SP. Desta forma, temos $\Omega^1 = \{OD|TS|SP\}$ presentes na camada de negócios do sistema de escalonamento.

4.2.3 Modelo de Mapeamento Conservador (MC)

A maneira com a qual o modelo conceitual foi proposto, em níveis, permite que uma série de mapeamentos seja possível. A Figura 4.4 apresenta uma possibilidade de mapeamento entre os níveis do modelo conceitual que será explorada nesta tese. O mapeamento tem como objetivo mapear as categorias de confiabilidade para os modelos de cobrança pertencentes ao conjunto Ω^1 SLA de nível 1. Em seguida, são mapeados para os modelos

pertencentes ao conjunto Ω^2 no SLA de nível 2. O modelo SP no nível 2 tem pouca garantia de disponibilidade. Desta forma, ele pode receber apenas mapeamentos do modelo SP do nível 1. Os modelos OD e TS no nível 1 podem ser mapeados para ambos modelos OD e RE no nível 2. Contudo, existe uma prioridade de escalonamento no mapeamento do modelo RE sobre o modelo OD (denotado pelos rótulos numerados) com o objeto de aumentar a utilização no modelo RE do nível 2 do SLA.

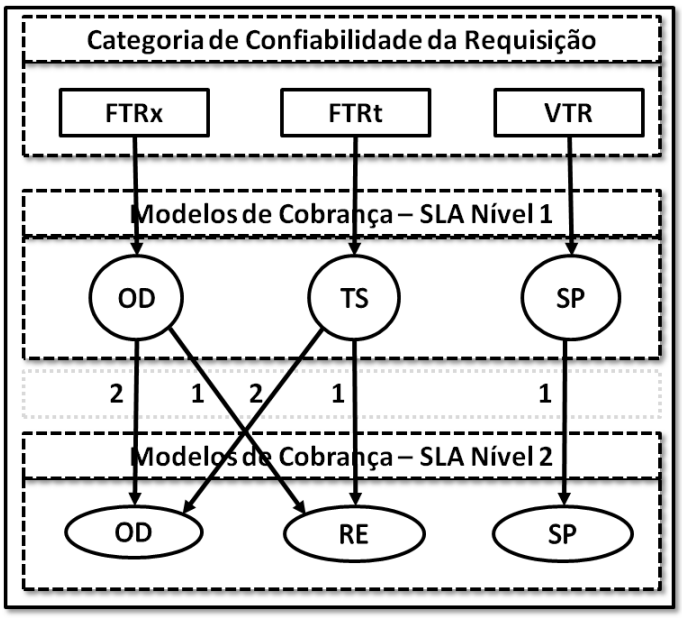


Figura 4.4: Mapeamento Conservador

Em geral, o custo de uma instância OD é maior que o custo de uma instância RE, e o custo de uma instância RE é maior que o custo de uma instância SP. Desta forma, a prioridade de utilização das instâncias tende a ser SP, RE e OD, nesta ordem. Definimos o mapeamento apresentado na Figura 4.4 como sendo o Mapeamento Conservador (MC) pelo fato de não violar os requisitos estabelecidos nas categorias de confiabilidade que são mapeadas para os modelos de cobranças pertencentes aos dois níveis de SLA.

4.2.4 Modelo de Mapeamento Flexível (MF)

O mapeamento conservador apresentado na Figura 4.4 assegura a qualidade de serviço. Isto é, os requisitos de qualidade de serviço relacionados à requisição não serão violados, pois estão sendo mapeados para modelos de cobrança que garantem a qualidade. Contudo, pode produzir um custo alto por utilizar instâncias OD e RE para alocar requisições FTRx e FTRt. Uma outra estratégia de mapeamento dos modelos de cobrança no nível 1 para o nível 2 é utilizar as instâncias de VM do tipo SP. Estas instâncias possuem menor custo

monetário do que a OD e RE. Um mapeamento alternativo ao mapeamento conservador é apresentado na Figura 4.5, onde é permitido o mapeamento de requisições FTRt para instâncias SP com custo menor.

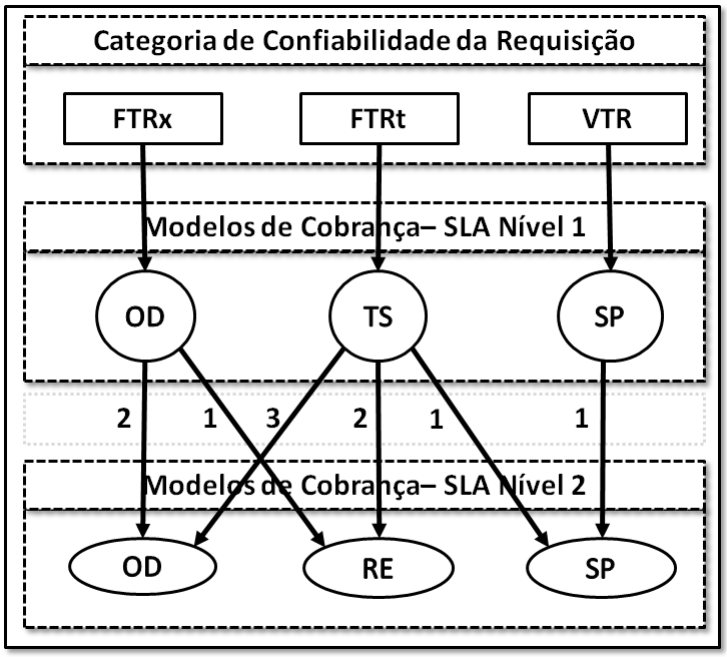


Figura 4.5: Mapeamento Flexível

Denominamos o mapeamento apresentado na Figura 4.5 como sendo um *mapeamento flexível* (MF). Utilizando o mapeamento flexível, o custo do escalonamento pode ser inferior quando comparado ao custo do escalonamento obtido utilizando o mapeamento conservador. Contudo, mapear requisições FTRt para instâncias do tipo SP pode gerar violação da qualidade de serviço da requisição, pois, como mencionado anteriormente, uma instância SP pode ser interrompida pelo provedor de IaaS. Para contornar isso, propomos a execução redundante destas requisições que será apresentada no Capítulo 5.

4.3 Modelagem dos Requisitos de Desempenho

Consideramos que o desempenho de uma requisição está relacionada à velocidade de processamento com a qual a requisição é executada durante um certo tempo. Não computamos o desempenho. Este é um valor fornecido pelo usuário durante a especificação da requisição. Por outro lado, os provedores de IaaS oferecem diferentes VMs com diferentes características de desempenho. Para que o escalonamento tenha um resultado razoável para o usuário, é importante que ele associe, de maneira inteligente, as necessida-

des especificadas através da requisição, às características da VM oferecida pelo provedor. Para realizar a associação da necessidade de desempenho solicitada pelo usuário com a capacidade de desempenho oferecida pelo provedor, definimos ρ como uma métrica para o desempenho Uma requisição r tem como um de seus atributos a necessidade de desempenho, denotada como $r(qos_d)$. Esta característica especifica quantos processadores são necessários para executar a requisição durante o tempo de duração definido em $r(d)$ [57].

Por outro lado, a instância de VM ψ tem um parâmetro $\psi(qos_d)$ associado que representa a capacidade de desempenho que ela oferece. A estratégia para incluir a dimensão de desempenho no SLA é fazer um mapeamento das requisições para as instâncias de VM. Utilizamos a seguinte notação $r(qos_d) \rightarrow \psi(qos_d)$, se $r(qos_d) = \psi(qos_d)$, isto é, a requisição r pode ser alocada para a instância de VM ψ se a necessidade de desempenho da requisição é igual a capacidade de desempenho da instância de VM. Esta não é uma condição suficiente para que a requisição r seja alocada na instância ψ, as outras dimensões (confiabilidade e segurança) também devem ser consideradas. Consequentemente temos uma relação entre a necessidade de desempenho de uma requisição r, denotado por $r(qos_d)$, e a capacidade de desempenho da VM ψ, denotada por $\psi(k)$ e definida no SLA.

4.4 Modelagem dos Requisitos de Segurança

No modelo que estamos propondo, os usuários realizam requisições de VMs para executar suas aplicações ou serviços. Como descrito anteriormente, o escalonamento considerando requisições de usuários distintos torna a questão de segurança ainda mais relevante. Por este motivo, incluímos a segurança como uma dimensão do SLA utilizado no modelo para realizar o escalonamento [58]. Nosso objetivo é oferecer a possibilidade de que o usuário especifique um nível de isolamento ou compartilhamento para executar seus serviços. Um grau maior de isolamento, pode resultar em um custo monetário maior. Por outro lado, se as requisições de diferentes usuários puderem ser executadas em VMs compartilhadas, o custo monetário tende a ser menor.

Definimos um modelo conceitual (Figura 4.6) para geração dos grupos de requisições, os quais são classificados em duas categorias: isolado e compartilhado. Os grupos na primeira categoria agrupam as requisições em conjuntos de usuários distintos. Neste caso, uma requisição de um usuário não é executada ao mesmo tempo com uma requisição de outro usuário (as requisições estão isoladas por usuários). Os grupos pertencentes à categoria compartilhado são compostos por requisições de vários usuários.

A partir do modelo conceitual para geração de grupos de requisições apresentado na Figura 4.6 descrevemos, a seguir, quatro possibilidades de geração apresentadas na Figura 4.7:

- Isolamento completo - Figura 4.7(a): Requisições entre usuários distintos não são

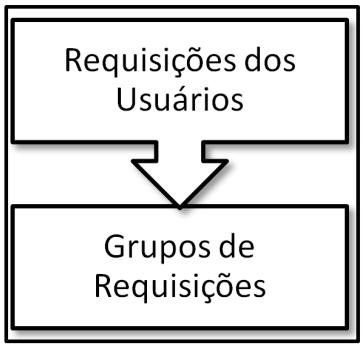


Figura 4.6: Modelo conceitual para geração de grupos de requisições

executadas na mesma VM. Para isso, é criado um grupo para cada usuário;

- Isolamento parcial - Figura 4.7(b): Parte das requisições de usuários distintos é atribuída a um mesmo grupo (g_0) e pode ser executada na mesma VM. Contudo, uma outra parte das requisições continua isolada completamente;
- Compartilhamento entre grupos confiáveis - Figura 4.7(c): Utilizando este modelo de geração é possível o compartilhamento de grupos de requisições de usuários confiáveis. Desta forma, um grupo é criado para mais de um usuário;
- Compartilhamento completo - Figura 4.7(d): Uma requisição pode ser executada com qualquer outra. Apenas um grupo é criado utilizando este modelo de geração de grupos.

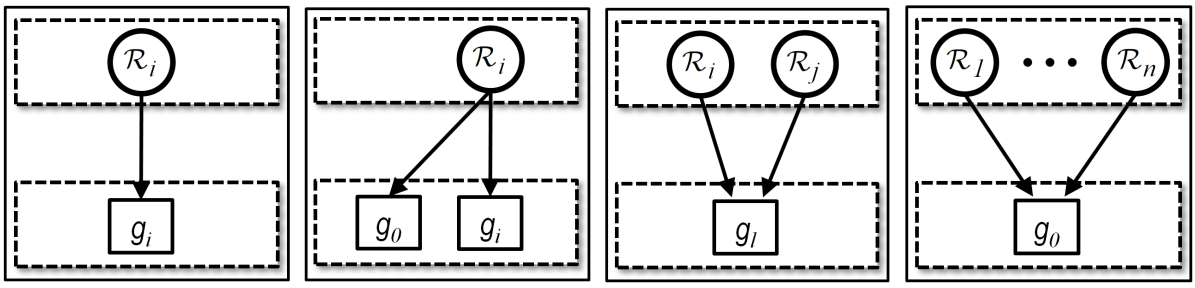


Figura 4.7: Possibilidades de modelos de geração de grupos

O modelo é bastante flexível e pode considerar vários níveis de isolamento ou compartilhamento de requisições. Se por um lado temos o isolamento completo apresentado na Figura 4.7(a), por outro lado temos o mapeamento denominado *compartilhamento total* apresentado na Figura 4.7(d).

Outras possibilidades podem ser incluídas como mapeamento para que o escalonamento atenda às necessidades dos usuários do sistema.

Capítulo 5

Escalonamento

Como mencionado anteriormente, o problema tratado nesta tese é o de propor um sistema que oferece o serviço de escalonamento de requisições de uma instituição sobre os serviços oferecidos por provedores de IaaS. A instituição é composta por diferentes usuários com requisitos distintos. Esse problema pode ser simples quando consideramos um único usuário e uma única requisição. Contudo, sua complexidade aumenta no momento em que consideramos o cenário onde temos vários usuários, várias requisições de cada usuário e vários provedores oferecendo diversos tipos de serviços.

O objetivo do sistema é minimizar o custo monetário para a instituição e localizar os melhores recursos que atendem aos requisitos solicitados pelos usuários. Como descrito no Capítulo 4, os requisitos tratados nesta tese são: confiabilidade, desempenho e segurança.

Neste capítulo, apresentamos o problema de escalonamento na Seção 5.1. Durante o processo de escalonamento os mapeamentos conservador e flexível devem ser tratados de maneira separada, pois possuem características bastante distintas e produzem escalonamentos que diferenciam-se principalmente pelo custo e pela garantia de QoS. Propomos estratégias para computar o escalonamento considerando cada mapeamento separadamente. Contudo, três procedimentos necessários para a caracterização das requisições segundo os requisitos de categorias de serviço, desempenho e segurança são comuns durante a computação dos escalonamentos: 1) mapeamento de categorias de QoS; 2) mapeamento da necessidade de desempenho; e 3) geração dos grupos de isolamento. Por este motivo, apresentamos na Seção 5.2 os três procedimentos antes de apresentarmos, nas Seções 5.3 e 5.4, os algoritmos para computar o escalonamento utilizando, respectivamente, os modelos de mapeamento conservador e flexível. Em cada seção os algoritmos são avaliados através de simulações onde são mostrados os resultados juntamente com as respectivas análises.

5.1 O Problema de Escalonamento

O escalonamento de recursos baseado em um SLA em computação em nuvem tem como objetivo encontrar os recursos solicitados pelos usuários considerando o SLA oferecido pelos provedores. Este é um problema NP-difícil. Estratégias de escalonamento ingênuas podem resultar em um grande impacto na quantia paga pelo usuário para utilizar os recursos, potencialmente aumentando o custo de se utilizar os serviços de provedores de nuvens. No cenário considerado nesta tese, cada usuário u_i submete um conjunto $\mathcal{R}_i$ de requisições r_j, $0 < j \leq n$ e $0 < i \leq n_u$. Desta forma, denotamos n como sendo a quantidade de requisições pertencentes ao conjunto $\mathcal{R}_i$ e n_u como sendo a quantidade de usuários que submeteram requisições ao sistema. Definimos $r = \{qos_c, qos_d, qos_s, d, d\alpha\}$ como sendo uma requisição de uma única instância de VM, onde:

- $r(qos_c)$: representa a necessidade de confiabilidade, que pode ser FTRx, FTRt e VTR.
- $r(qos_d)$: representa a necessidade de desempenho;
- $r(qos_s)$: representa a necessidade de segurança;
- $r(d)$: representa o tempo de duração da instância;
- $r(d\alpha)$: representa um relaxamento sobre $r(d)$; e

Por definição, temos $d = d\alpha, \forall\ r \in \mathcal{R}$, tal que, $r(qos_c) = \{FTRx\}$; e $d < d\alpha, \forall r \in \mathcal{R}$, tal que, $r(qos_c) = \{FTRt, VTR\}$.

Definimos como $\mathcal{R}^+ = \bigcup \mathcal{R}_i$, $0 < i \leq n_u$, o conjunto de todas as requisições de todos os usuários.

Consideramos que as requisições de VMs realizadas pelo usuário devem ser alocadas sobre o conjunto $\Psi = \bigcup \Psi_{p_i} = \{\psi_1, \psi_2, ..., \psi_m\}$ de VMs dos provedores de IaaS, onde $0 < i < p_n$. Denotamos m e p_n como sendo, respectivamente, a quantidade de VMs e a quantidade de provedores de IaaS disponíveis para o escalonamento. Cada VM é definida como $\psi = \{qos_c, qos_d, c\}$, onde:

- qos_c: representa a confiabilidade na execução da VM;
- qos_d: representa a capacidade de desempenho da VM; e
- c: representa o custo por alugar uma unidade de tempo da VM.

O conjunto de VMs durante o processo de escalonamento já está disponível e deste modo estamos considerando um conjunto pré-determinado de recursos já alugados sobre

múltiplos provedores. Otimizar o uso dos recursos alugados permite ao usuário determinar quais VMs são necessárias para atender as requisições considerando suas características e reduzir os custos.

O problema tratado nesta tese é descrito como a seguir:

Compute um escalonamento S para um conjunto $\mathcal{R}^+$ de requisições de VMs sobre um conjunto Ψ de instâncias de VMs de provedores de IaaS com o objetivo de alcançar o menor custo de alocação sem violar a qualidade de serviço das requisições.

A Figura 5.1 apresenta o cenário que estamos considerando nesta tese.

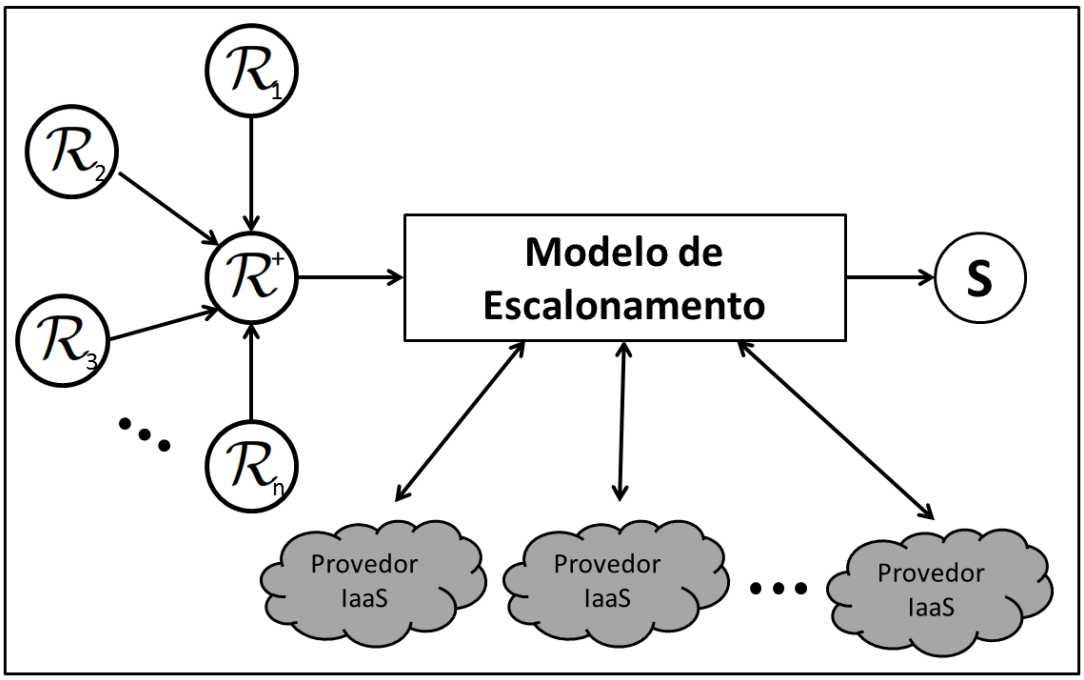


Figura 5.1: Cenário para submissão das requisições

Como observado na Figura 5.1, dado um conjunto $\mathcal{R}^+$ de n requisições, devemos obter um escalonamento $S = \{t, q, c, E\}$ onde:

- S(t): representa o tempo total de execução de $\mathcal{R}$;
- S(q): representa a quantidade de requisições atendidas no escalonamento computado;
- S(c): representa o custo do escalonamento; e
- S(E): representa o escalonamento computado.

Sobre a perspectiva do usuário é necessário verificar quantas requisições podem ser atendidas, e a que custo, utilizando VMs alugadas dos provedores. Uma requisição deve ser atendida com o menor custo respeitando suas características. Objetivamos atender a maior quantidade de requisições possível. Assumimos que as requisições não possuem dependência entre si e utilizamos o modelo de implantação de nuvem pública. Desta forma, todas as requisições são alocadas sobre os provedores públicos de IaaS. Este trabalho pode ser estendido para considerar que o sistema de escalonamento tenha um conjunto de recursos locais e que utilize os recursos públicos apenas quando a demanda por recursos seja superior a quantidade de recursos existentes na infraestrutura local.

5.2 Adequação às Necessidades do Usuário

A seguir, apresentamos nossas estratégias para realizar a adequação das solicitações dos usuários aos serviços oferecidos atualmente pelos provedores de IaaS.

5.2.1 Mapeamento das Categorias de Serviço

Ambos os modelos de mapeamento conservador e flexível mapeiam diretamente uma categoria de serviço para um modelo de cobrança do modelo pertencente ao SLA de nível 1. Desta forma temos o seguinte mapeamento para ambos os modelos:

1. Requisições de Tempo Fixo (FTRx) são mapeadas para o modelo de cobrança *on-demand*.

2. Requisições de Tempo Flutuante (FTRt) são mapeadas para o modelo de cobrança *time-slotted-reservation*.

3. Requisições de Tempo Variável (VTR) são mapeadas para o modelo de cobrança *spot*.

Como descrito anteriormente, esta é uma possibilidade de mapeamento e outras podem ser propostas. Pelo fato de realizarmos um mapeamento direto, o procedimento é trivial. Caso seja necessária alguma outra computação, pode ser feita durante este procedimento.

5.2.2 Mapeamento da Necessidade de Desempenho

Como descrito na Seção 4.3, utilizamos a métrica ρ como sendo uma unidade para medirmos o desempenho. Desta forma, podemos relacionar a necessidade de desempenho requisitada pelo usuário à capacidade de desempenho oferecida pela VM. Dependendo da quantidade de VMs disponíveis para o escalonamento e a demanda dos usuários, pode ser

mais fácil ou mais difícil alocar uma requisição em uma instância de VM. Para aumentar a probabilidade de sucesso de uma requisição ser alocada, propomos um método de decomposição da requisição antes de submetê-la ao escalonador. O processo de decomposição de uma requisição r consiste em variarmos proporcionalmente os parâmetros $r(d)$ e $r(qos_d)$ de acordo com um fator Δ. Considere ama requisição r_i com tempo de duração $r(d) = 10$ e $r(qos_d) = 1$. Suponha que exista apenas uma VM ψ com $\psi(qos_d) = 2$. Desda forma, a requisição não seria alocada. Contudo, variamos a necessidade de desempenho de r para $r(qos_d) = 2$, $\Delta = 2$. Assim, r pode ser executada em ψ na metade do tempo de execução.

Durante o processo de decomposição de r_i geramos um conjunto de novas requisições denotado por $\mathcal{R}^d_{r_i} = \{r^d_{i,1}, r^d_{i,2}, r^d_{i,3}, ..., r^d_{i,f}\}$, onde $r^d_{i,j} = \{qos_c, \Delta qos_d, qos_s, \frac{d}{\Delta}, d\alpha\}, 1 \leq j \leq f$. Denotamos como f a quantidade de capacidades de desempenho distintas entre as instâncias de VM. Desta maneira, redefinimos a requisição por inserir um parâmetro Δ, o qual é variado antes do escalonamento para produzir requisições similares com diferentes necessidades de desempenho e tempos de duração. Observe que quanto mais aumentarmos o desempenho, menor é o tempo de duração. Como descrito anteriormente, ρ é uma métrica para o desempenho. Desta forma, as instâncias de VMs oferecidas pelos provedores podem ser classificadas como $k = \{\rho, 2\rho, 3\rho, ..., f\rho\}$ de acordo com a capacidade de desempenho oferecida. É importante observar que o processo de decomposição das requisições é utilizado para criar novas possibilidades de escalonamento por gerar requisições similares. Contudo, apenas uma requisição do conjunto de decomposição $\mathcal{R}^d_{r_i}$ da requisição r_i deve ser escalonada e executada.

Apresentamos na Fig. 5.2, um exemplo ilustrativo com possibilidades de decomposição de uma requisição. No eixo "x"temos a necessidade de desempenho da requisição, $r(qos_d)$. No eixo "y", temos o tempo de duração da requisição, $r(d)$. Seja r uma requisição com $r(d) = d'$ e $r(qos_d) = \rho$. A requisição pode ser decomposta em duas novas requisições variando o valor de Δ em um conjunto $\{1, 2, 4\}$. Desta maneira, temos $|\mathcal{R}^d_r| = 3$. Observe que com $\Delta = 1$ temos a requisição original, a qual pertence ao conjunto $\mathcal{R}^d_r$.

As Figuras 5.2(a), 5.2(b) e 5.2(c) mostram as requisições geradas utilizando, respectivamente, $\Delta = 1, \Delta = 2$ e $\Delta = 4$. A requisição com tempo de duração $r(d) = d'$ e necessidade de desempenho $r(qos_d) = \rho$ pode ser executada nas instâncias de VMs com $\psi(k) = \rho$, $\psi(k) = 2\rho$ e $\psi(k) = 4\rho$ sem que ocorra a violação de QoS por atraso. A duração da requisição é proporcional à capacidade de desempenho da instância de VM na qual ela será executada.

Analisando as possibilidades de valores de Δ, observamos que se $1 < \Delta$, a requisição no conjunto $\mathcal{R}^d_r$ terá um tempo de execução menor que o tempo de execução da requisição original. Por outro lado, se $0 < \Delta < 1$, a requisição no conjunto $\mathcal{R}^d_r$ executará em uma instância com menos desempenho e, consequentemente, com maior tempo de execução.

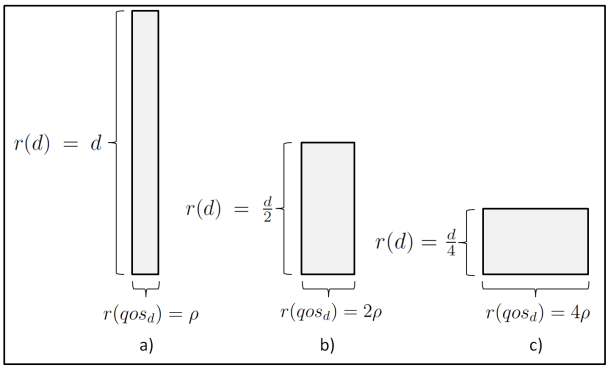


Figura 5.2: Possibilidades de execução considerando o desempenho da VM

5.2.3 Geração de Grupos de Isolamento e Compartilhamento

Como descrito anteriormente, o modelo de escalonamento considera requisições de vários usuários simultaneamente. Algumas requisições podem ser alocadas em uma mesma VM considerando o grau de isolamento ou compartilhamento especificado pelo usuário para a requisição. Neste contexto, a segurança torna-se uma característica importante no escalonamento.

Estamos propondo o isolamento de requisições que não podem ser escalonadas com outras requisições na mesma VM. A estratégia para garantir o isolamento das requisições é baseada no agrupamento de requisições de usuários confiáveis em grupos do conjunto $\mathcal{G} = \{g_0, g_1, g_2, ..., g_{n_g}\}$. Isto é, agrupamos requisições que podem ser alocadas na mesma VM durante o escalonamento. Denotamos n_g como sendo o número arbitrário de grupos criados. Na Figura 4.7 propomos 4 modelos de geração de grupos: (i) grupos com isolamento completo; (ii) grupos com isolamento parcial; (iii) compartilhamento entre grupos confiáveis; e (iv) grupo com compartilhamento completo.

O Algoritmo 1 computa a geração dos grupos com isolamento completo. Recebe como entrada os seguintes parâmetros: o conjunto $\mathcal{R}^+$ de requisições a serem agrupadas e a quantidade n_u de usuários. Durante a geração dos grupos criamos um grupo g_i para cada conjunto de requisições $\mathcal{R}_i$ pertencente ao usuário u_i. Utilizamos a necessidade de segurança definida na requisição, $r(qos_s)$, para realizar o agrupamento das requisições.

A computação dos grupos com isolamento parcial é feita através do Algoritmo 2 utilizando uma tabela binária T^{rc} de tamanho $(n_u X n)$ contendo a marcação das requisições que podem ser compartilhadas (se estiver marcada com 1 ela pode ser compartilhada, 0 caso contrário). Caso a requisição r do usuário u_i esteja marcada, a requisição será atribuída ao grupo g_0, linha 3. Caso contrário, será atribuída ao grupo do usuário e não será compartilhada.

O Algoritmo 3 computa os grupos de usuários confiáveis utilizando o vetor V_{uc} de

Algoritmo 1 Gerador de Grupos com Isolamento Completo

Entrada: $\mathcal{R}^+$, n_u
Saída: $\mathcal{R}^+$
1: **Para** i de 1 até n_u **faça**
2: **Para todo** $r \in \mathcal{R}_i$ **faça**
3: $r(qos_s) \leftarrow i$;
4: **fim Para**
5: **fim Para**
6: Retorne $\mathcal{R}^+$

Algoritmo 2 Gerador de Grupos com Isolamento Parcial

Entrada: $\mathcal{R}^+$, n_u, T^{rc}
Saída: $\mathcal{R}^+$
1: **Para** u de 1 até n_u **faça**
2: **Para todo** $r \in \mathcal{R}_u$ **faça**
3: $r(qos_s) \leftarrow u - T^{rc}_{ur} \times u$;
4: **fim Para**
5: **fim Para**
6: Retorne $\mathcal{R}^+$

tamanho n_u que relaciona usuários aos pares que são confiáveis um ao outro. Atribuímos as requisições destes usuários ao grupo g_k. Atualizamos o valor de k para cada par de usuários.

A computação dos grupos completamente compartilhados é feita utilizando o Algoritmo 4. Neste caso, todas as requisições são atribuídas ao grupo g_0.

A complexidade assintótica dos Algoritmos 1, 2, 3, e 4 é $O(n \times n_u)$, pois classificamos todas as n requisições de cada um dos n_u usuários.

Com os grupos gerados a partir dos modelos descritos anteriormente, o escalonamento pode ser realizado seguindo as necessidades de segurança dos usuários.

5.3 O Escalonamento Utilizando o Mapeamento Conservador

O modelo de mapeamento conservador descrito na Seção 4.2.3 propõe um mapeamento em que as características das requisições não sofrem violações. Propomos o Algoritmo 5 que utiliza os três procedimentos descritos na Seção 5.2, que realizam a adequação às necessidades do usuário, e uma PLI para computar o escalonamento do conjunto de requisições $\mathcal{R}^+$ sobre o conjunto de instâncias de VMs Ψ alugadas a partir dos provedores de IaaS e disponibilizadas para o escalonamento.

Algoritmo 3 Gerador de Grupos de Usuários Confiáveis

Entrada: $\mathcal{R}^+$, n_u, V_{uc}
Saída: $\mathcal{R}^+$
1: $K \leftarrow 1$
2: **Para** i de 1 até n_u **faça**
3: **Se** $V_{uc}[i] <> 0$ **então**
4: **Para todo** $r \in \mathcal{R}_i$ **faça**
5: $r(qos_s) \leftarrow k$
6: **fim Para**
7: **Para todo** $r \in \mathcal{R}_{V_{uc}[i]}$ **faça**
8: $r(qos_s) \leftarrow k$
9: **fim Para**
10: **fim Se**
11: $k \leftarrow k + 1$
12: **fim Para**
13: Retorne $\mathcal{R}^+$

Algoritmo 4 Gerador de Grupos Completamente Compartilhado

Entrada: $\mathcal{R}^+$, n_u
Saída: $\mathcal{R}^+$
1: **Para** i de 1 até $|\mathcal{R}^+|$ **faça**
2: $r(s) \leftarrow 0$;
3: **fim Para**
4: Retorne $\mathcal{R}^+$

A PLI-1 utilizada na linha 4 computa a melhor solução, contudo, eleva a complexidade assimptótica da solução para um fator exponencial. Denotamos a opção utilizada para geração dos grupos de isolamento como m^g.

Apresentaremos na próxima seção a formulação da PLI-1 e em seguida, os resultados experimentais obtidos por meio da execução da solução proposta no Algoritmo 5.

5.3.1 Uma Formulação de PLI para Computar o Escalonamento

Formulamos uma PLI para computar o escalonamento de $\mathcal{R}^+$ sobre Ψ. A ideia central da formulação é considerar o tempo de duração da requisição $r(d)$ e tratar cada unidade de tempo de $r(d)$ separadamente, respeitando as limitações impostas pelo problema. Desta forma, os três principais itens tratados na PLI são: a requisição r, o tempo t e a máquina virtual ψ. Estes três itens referem-se ao cenário onde a requisição r é executada em uma unidade de tempo t na máquina virtual ψ. Com isso, para que uma requisição seja executada em uma máquina virtual, ela deve ser executada em $r(d)$ unidades de tempo.

Algoritmo 5 Escalonamento Utilizando Mapeamento Conservador

Entrada: $\mathcal{R}^+$, Ψ, n_u
Saída: S_F
1: $\mathcal{R}^+ \leftarrow$ *Mapeamento_Categorias_de_QoS*($\mathcal{R}^+$, Ω^1)
2: $\mathcal{R}^+ \leftarrow$ *Mapeamento_Desempenho*($\mathcal{R}^+$)
3: $\mathcal{R}^+ \leftarrow$ *Mapeamento_Seguranca*($\mathcal{R}^+$, m^g, n_u)
4: $S_F \leftarrow$ PLI-1($\mathcal{R}^+$, Ψ, n_u, Ω^2)
5: return S_F

A PLI-1 utiliza as variáveis binárias u, v, w, x, y e z e as constantes C, M e K como a seguir:

- $u_{t,\psi}$: Assume o valor 1 se a VM ψ executa alguma requisição no instante t. Caso contrário, o valor 0;
- $v_{g,\psi}$: Assume o valor 1 se a VM ψ executa alguma requisição do grupo g. Caso contrário, o valor 0;
- w_r: Assume o valor 1 se a requisição r é executada. Caso contrário, o valor 0;
- $x_{r,\psi}$: Assume o valor 1 se a requisição r é executada na VM ψ independente do instante t. Caso contrário, o valor 0;
- $y_{r,t,\psi}$: Assume o valor 1 se a requisição r é executada na VM ψ no instante t. Caso contrário, o valor 0;
- $z_{r,t,\psi}$: Assume o valor 1 se a requisição r com $r(\delta) = \{FTRt\}$ inicia a execução no instante t sobre a VM ψ. Caso contrário, o valor 0;
- $C_{t,\psi}$: Constante que assume o custo por unidade de tempo por utilizarmos a VM ψ. Caso contrário, o valor 0.
- $\mathcal{M}$: Constante suficientemente grande que assume um peso para cada requisição.
- $\mathcal{K}$: Constante suficientemente grande utilizada para garantir que requisições FTRt iniciem apenas uma vez.

Formulamos a função objetivo $F = \sum_{t \in T} \sum_{\psi \in \Psi} (u_{t,\psi} \times C_{t,\psi}) - \sum_{r \in \mathcal{R}} (w_r \times \mathcal{M})$ que computa o escalonamento de $\mathcal{R}^+$ sobre Ψ objetivando o menor custo de alocação sem violar os requisitos da requisição. Desta maneira, devemos minimizar F sujeito às seguintes restrições rescritas a seguir.

$$\sum_{t=1}^{r(d\alpha)} y_{r,t,\psi} = r(d) \times x_{r,\psi}; \forall r \in \mathcal{R}, \forall \psi \in \Psi \tag{5.1}$$

A restrição (R-5.1) especifica que se uma requisição for executada, ela deve ser executada em $r(d)$ unidades de tempo e em uma única VM. A variável $x_{r,\psi}$ informa se a requisição r é executada. É possível que nem toda requisição seja atendida devido à incompatibilidade dos requisitos e das características das VMs disponíveis para o escalonamento.

$$\sum_{g \in G} v_{g,\psi} \leq 1; \forall \psi \in \Psi \tag{5.2}$$

$$x_{r,\psi} \leq v_{r(g),\psi}; \forall \psi \in \Psi, \forall r \in \mathcal{R} \tag{5.3}$$

As restrições (R-5.2) e (R-5.3) trabalham juntas e especificam que as requisições escalonadas em uma VM devem pertencer ao mesmo grupo de isolamento ou compartilhamento. A primeira especifica que uma VM ψ deve atender requisições de no máximo um grupo. Nem todas as VMs são utilizadas em um cenário onde há poucas requisições. A segunda especifica que se a requisição r executar na VM ψ, a VM ψ deve executar somente requisições do grupo da requisição de r, denotado como $r(g)$.

$$\sum_{\psi \in \Psi} x_{r,\psi} = w_r; \forall r \in \mathcal{R} \tag{5.4}$$

A restrição (R-5.4) especifica que se uma requisição for escalonada, ela contribuirá para o custo total.

$$\sum_{r \in \mathcal{R}} (y_{r,t,\psi} \times r(qos_d)) = u_{t,\psi} \times \psi(k); \forall t \in \left[1, r(d\alpha)\right], \forall \psi \in \Psi \tag{5.5}$$

A restrição (R-5.5) especifica que a soma do poder de desempenho das requisições executadas em uma unidade de tempo não deve ultrapassar o poder de desempenho da instância de VM na qual as requisições foram alocadas.

$$\sum_{r \in R} y_{r,t,\psi} = 0; \forall t \in T; \psi \in \Psi; MF(r, \psi) = 0 \tag{5.6}$$

A garantia de execução conforme o modelo de mapeamento entre SLA de nível 1 e SLA de nível 2 apresentado na Fig. 4.4 é feita pela restrição (R-5.6) em conjunto com a função de mapeamento FM().

$$r(d) - \mathcal{K} \times (1 - z_{r,t,\psi}) \leq \sum_{s=t}^{r(d)+t-1} y_{r,s,\psi} \leq r(d) + \mathcal{K} \times (1 - z_{r,t,\psi}) \tag{5.7}$$
$$\forall r \in \mathcal{R}, \forall t \in \Big[1, r(d\alpha) - r(d) + 1\Big], \forall \psi \in \Psi, r(qos_d) = \{FTRx, FTRt\}$$

$$\sum_{t=1}^{r(d\alpha)-r(d)+1} z_{r,t,\psi} = x_{r,\psi}; \forall r \in \mathcal{R}, \forall \psi \in (\Psi^{od} \bigcup \Psi^{re}) \tag{5.8}$$

As restrições (R-5.7) e (R-5.8) asseguram que requisições do tipo FTRt sejam executadas de maneira atômica.

$$v_{g,\psi}, w_r, x_{r,\psi}, y_{r,t,\psi}, z_{r,t,\psi} \in \{0,1\}; \forall g \in G, \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \forall \psi \in \Psi \tag{5.9}$$

A restrição (R-5.9) assegura que as variáveis v, w, x, y e z sejam binárias.

$$\mathcal{R}_r^d \leq L \times (1 - w_r); \forall r \in \mathcal{R} \tag{5.10}$$

Somente uma requisição do conjunto $\mathcal{R}_r^d$, obtido durante o processo de decomposição da requisição r, deve ser atendida. A restrição (R-5.10) assegura esta propriedade.

A PLI-1 computa o melhor escalonamento com o maior número de requisições atendidas. Uma requisição não pode ser atendida parcialmente. Contudo, o número de VMs disponíveis pode não ser suficiente para atender todas as requisições. Neste caso, a PLI-1 retorna o escalonamento com o maior número de requisições que podem ser atendidas e o menor custo de escalonamento.

O valor de uma constante $\mathcal{M}$ é atribuído para cada requisição w_r atendida. Isto significa que quanto maior o número de requisições atendidas, menor o resultado da PLI-1. Seja C_t o custo computado pela PLI-1. Então, $S.q = \lfloor \frac{-C_t}{\mathcal{M}} \rfloor + 1$ e $S.c = (S.q \times \mathcal{M}) - C_t$. Seja d_m e ψ_m, respectivamente, o maior tempo de execução entre todas as requisições $r \in \mathcal{R}$ e o maior custo por unidade de tempo entre todas as VMs $\psi \in \Psi$. Neste sentido, $S.q$ e $S.t$ podem ser computados como descrito anteriormente somente se $\mathcal{M} > d_m \times \psi_m$. Esta é a maneira como $\mathcal{M}$ deve ser definida. Caso contrário, o resultado da PLI-1 pode ser comprometido.

5.3.2 Configuração dos Experimentos

Implementamos o Algoritmo 5 para computar o escalonamento utilizando o modelo conservador. As métricas observadas neste são:

- custo do escalonamento;
- quantidade de requisições atendidas;
- quantidade utilizada de cada tipo de instância de VM;
- quantidade de unidades de tempo atendidas; e
- quantidade de requisições geradas pelo processo de decomposição da requisição.

Utilizamos o termo "requisições atendidas"como sendo as requisições que foram escalonadas.

O algoritmo foi implementado utilizando JAVA e o IBM ILOG CPLEX com a configuração padrão para resolver a PLI-1.

Realizamos 6 experimentos, E_i, $1 \leq i \leq 6$, cada um com 30 conjuntos de requisições, $E_i = \{R_1, R_2, ..., R_{30}\}$. Em cada experimento, o número de requisições em cada conjunto é E_i: $|R_j| = i \times 10$, $1 \leq j \leq 30$. Cada conjunto R_j de requisições é composto por 20% de requisições FTRx, 40% de requisições FTRt e 60% de requisições VTR. Além disso, configuramos o tempo de execução $r(d) \leftarrow random(1,20)$ e o tempo de relaxamento $r(d\alpha) \leftarrow random(1, 80)+r(d)$ de cada requisição em cada conjunto $R_j \in E_i$. Não apresentamos os dados computados em $E_1 = 10$, pois este experimento contém uma quantidade inexpressiva de requisições (E_1: $|R_j| = 10$) face à quantidade de VMs disponíveis.

Como apresentado na Tabela 5.1 temos recursos disponíveis de três provedores. O custo de cada VM é proporcional ao custo atualmente exercido pela Amazon EC2.

Tabela 5.1: Conjunto de VMs disponíveis para o escalonamento.

Provedor	**Desempenho**	**OD**	$	**RE**	$	**SP**	$	**TOTAL**
1	1ρ	1	70.00	0	0	0	0	2
	2ρ	1	140.00	0	0	0	0	
2	1ρ	0	0	1	40.00	0	0	2
	2ρ	0	0	1	100.00	0	0	
3	1ρ	0	0	0	0	6	8.00	12
	2ρ	0	0	0	0	6	16.00	
							Total	**16**

Variamos a quantidade de usuários $n_u = \{1, 2, 3, 4\}$. Com isso, podemos observar os resultados quando o conjunto de requisições $\mathcal{R}^+$ é composto por: apenas um usuário ($n_u = 1$), dois usuários ($n_u = 2$), três usuários ($n_u = 3$) e quatro usuários ($n_u = 4$). Utilizamos o modelo de geração de grupos completamente isolado (Figura 4.7-a). Além disso, variamos a composição da necessidade de desempenho das requisições, considerando seu desempenho, em dois contextos:

- 100% com $r(qos_d) = 1$: Neste caso, todas as requisições necessitam de apenas uma unidade de processamento (1ρ). Denotaremos esta composição como k^1; e

- 50% com $r(qos_d) = 1$ e 50% com $r(qos_d) = 2$: Neste caso, metade das requisições necessitam de uma unidade de processamento (1ρ), e a outra metade necessita de duas unidades de processamento (2ρ). Denotaremos esta composição como k^2.

Como apresentado a seguir, estas variações influenciam no custo.

5.3.3 Avaliação dos Resultados Obtidos

A Figura 5.3 apresenta o custo do escalonamento obtido na execução dos cinco experimentos (E_2, E_3, E_4, E_5 e E_6) utilizando o Algoritmo 5 quando variamos a quantidade de usuários n_u e a necessidade de desempenho k. As Figuras 5.3(a), 5.3(b), 5.3(c) e 5.3(d) apresentam, respectivamente a variação em $n_u = 1$, $n_u = 2$, $n_u = 3$ e $n_u = 4$. A variação de k é comparada em cada uma das figuras.

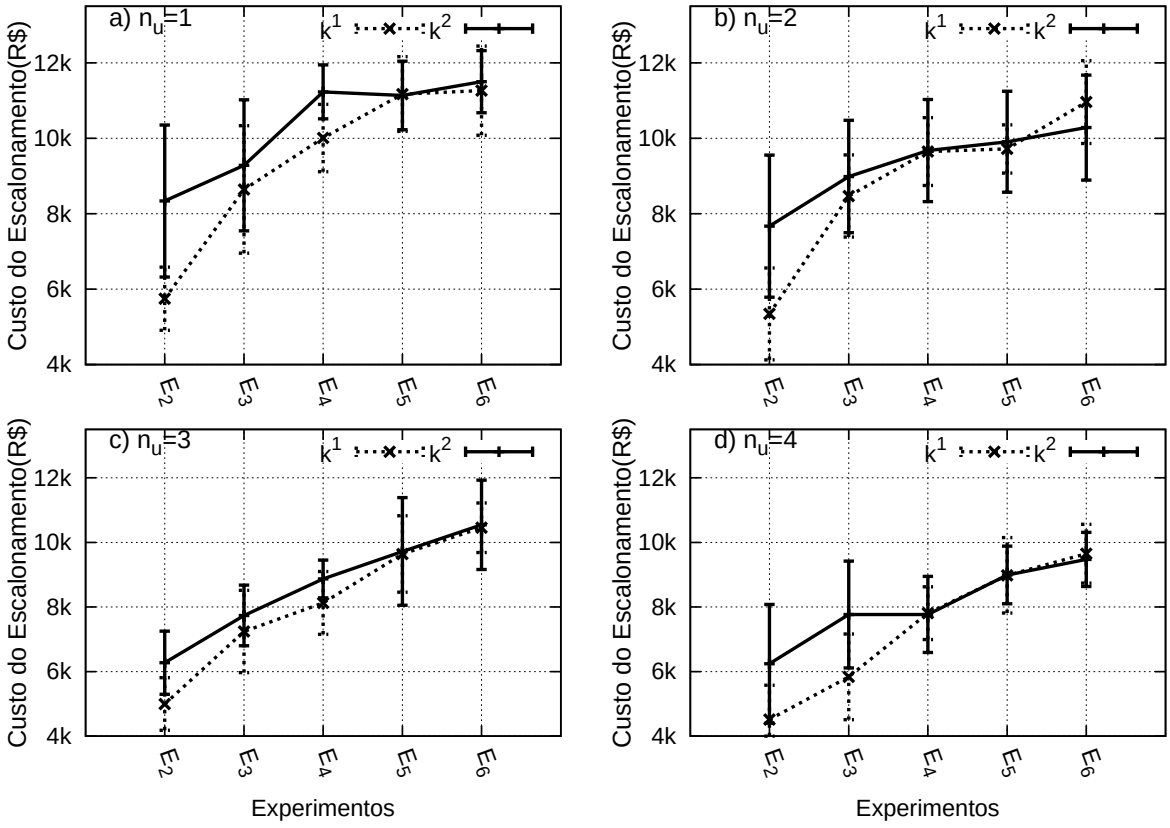


Figura 5.3: Custo do Escalonamento variando o valor de $r(qos_d)$ e de n_u

Ao variamos o valor de $r(qos_d)$, observamos que o custo do escalonamento, quando temos 100% das requisições com k^1, é inferior ao custo do escalonamento considerando k^2. Isso deve-se ao fato de que quando temos apenas um tipo de necessidade de desempenho, temos menos restrições ao escalonamento. Consequentemente, mais possibilidades de alocar uma requisição em uma VM. A diferença entre o custo é maior quando observamos o Experimento E_2, comparado ao Experimento E_6. Isso é justificado pelo fato de termos menos requisições em E_2 (cada conjunto tem 20 requisições) e mais requisições em E_6 (cada conjunto tem 60 requisições). Com menos requisições no experimento temos mais chances de alocar as requisições, pois existem mais unidades de tempo livres na linha de tempo de execução de uma VM. Além disso, são utilizadas as VMs com custo mais baixo. À medida em que aumentamos a quantidade de requisições nos experimentos, temos menos chances de alocar as requisições, pois as unidades de tempo das VMs são ocupadas. Neste caso, várias requisições podem concorrer às mesmas unidades de tempo na mesma VM. Inicialmente, são utilizadas as VMs com menor custo, em seguida com maior custo. Desta forma aumentamos o custo do escalonamento.

Ao observarmos a variação do número de usuários, verificamos que o custo do escalonamento obtido no Experimento E_2 quando temos um usuário $n_u = 1$ para o k^1 é aproximadamente 68% do valor obtido com k^2. Observando o experimento E_6, esta diferença diminui e até mesmo fica negativa (Figuras 5.3-b e 5.3-b). Além disso, observamos que o custo do escalonamento é maior para $n_u = 1$ e menor para $n_u = 4$. O Experimento E_2 com um usuário $n_u = 1$ (Figura 5.3-a) tem um custo 30% maior comparado ao escalonamento com quatro usuários $n_u = 4$ (Figura 5.3-d).

Os custos obtidos são compostos pelo custo por unidade de tempo de locação VMs do tipo: *on-demand* (OD), *reserved* (RE) e *spot* (SP). Desta forma, a quantidade de unidades de tempo alocadas em cada tipo de VM influencia diretamente no custo total do escalonamento. Outro fator determinante no custo é a quantidade de requisições atendidas. Em geral, quanto maior a quantidade de requisições atendidas, maior é o custo do escalonamento.

A Figura 5.4 apresenta: (a) a quantidade de requisições atendidas; (b) a quantidade de VMs do tipo OD utilizadas; (c) a quantidade de VMs do tipo RE utilizadas; e (d) a quantidade de VMs do tipo SP utilizadas para k^1. As mesmas informações obtidas para k^2 são apresentadas na Figura 5.5.

Observando a Figura 5.4(a), verificamos que à medida em que aumentamos a quantidade de requisições nos experimentos, diminuímos a quantidade de requisições atendidas. Isso acontece pelo fato de que não termos VMs suficientes para atender todas as requisições. No Experimento E_2 para $n_u = 1$ temos 99% de requisições atendidas. No Experimento E_6 temos 80%. Para quatro usuários ($n_u = 4$), o Experimento E_2 atinge 86% e o Experimento E_6 atinge 74%. A diferença entre a quantidade de requisições aten-

didas nos Experimentos E_2 e E_6 para n_1 é de aproximadamente 20%. Esta diferença diminui para 15% quando observamos os experimentos em n_4. Isso acontece pelo fato de que estamos utilizando o modelo de geração de grupos completamente isolados (Figura 4.7-a) nos experimentos. Com isso, quanto maior a quantidade de usuários, menores são as possibilidades de casamento entre as demandas de uma requisição e a oferta de uma VM.

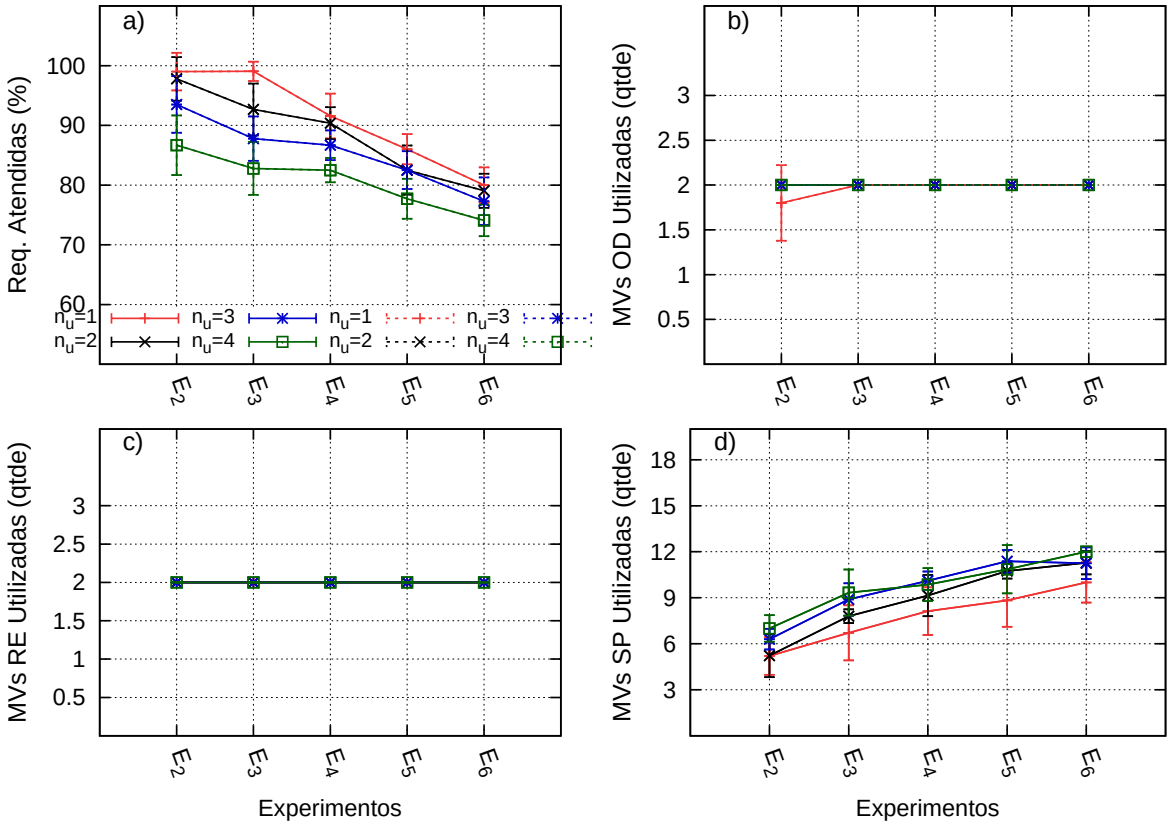


Figura 5.4: Quantidade de requisições atendidas e tipo de VMs utilizadas para k^1

As Figuras 5.4(b), 5.4(c) e 5.4(d) apresentam a quantidade de VMs utilizadas dos tipos, respectivamente, OD, RE e SP. Em quase todos os experimentos são utilizadas as duas VMs disponíveis dos tipos OD e RE, o que representa 100% da quantidade disponível para o escalonamento (observada na Tabela 5.1). Este comportamento é justificado pelo fato de que estamos utilizando o modelo de mapeamento conservador. Então, as requisições do tipo FTRx e FTRt concorrem e são escalonadas para VMs dos tipos OD e RE. Como mencionado anteriormente, nestes experimentos, temos 20% de requisições FTRx, 40% de requisições FTRt e 60% de requisições VTR. São escalonadas 60% das requisições para as VMs dos tipos OD e RE. Isso justifica a alta quantidade de VMs já no experimento E_2.

Observando a Figura 5.4(d) verificamos que a quantidade de VMs do tipo SP utilizadas é menor no Experimento E_2 e maior no Experimento E_6. Isso é justificado pela quantidade de requisições em cada experimento. Outra observação importante neste gráfico é que os escalonamentos para $n_u = 3$ e $n_u = 4$, utilizam mais VMs quando comparados aos escalonamentos para $n_u = 1$ e $n_u = 2$. Isso deve-se ao fato de que para um e dois usuários temos mais chances de alocação das requisições do que para três e quatro usuários. No Experimento E_6, todos atingem 100% de utilização das VMs do tipo SP.

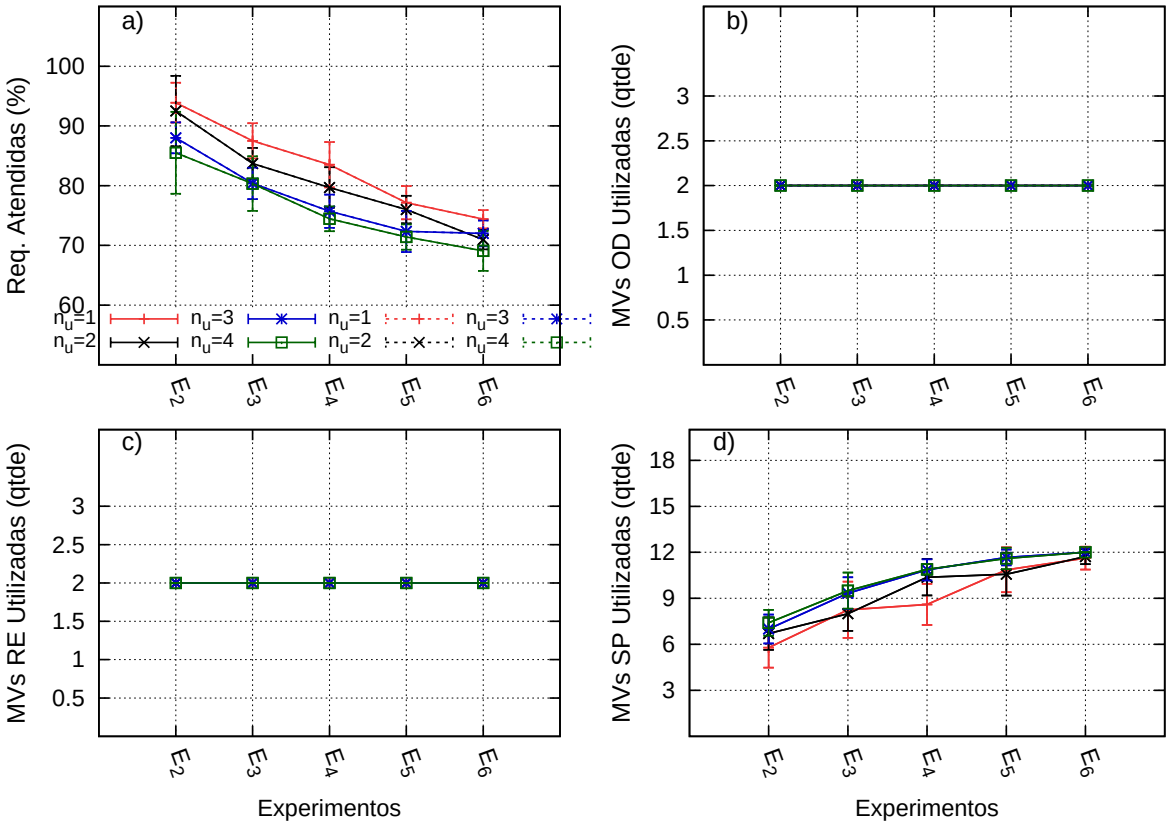


Figura 5.5: Quantidade de requisições atendidas e tipo de VMs utilizadas para k^2

A Figura 5.5 apresenta: (a) a quantidade de requisições atendidas; (b) a quantidade de VMs do tipo OD utilizadas; (c) a quantidade de VMs do tipo RE utilizadas; e (d) a quantidade de VMs do tipo SP utilizadas para k^2, similarmente aos dados apresentados na Figura 5.4.

Comparando as Figuras 5.4 e 5.5, podemos observar um comportamento semelhante nos gráficos. Contudo, os valores diferem. Os Experimentos E_2 e E_3 com $n_u = 1$ atingem aproximadamente 99% utilizando k^1, Figura 5.4(a). Utilizando k^2, os mesmos experimentos atingem respectivamente, 93% e 87%, Figura 5.5(a). Já para o Experimento E_6, o

valor é de aproximadamente 80% para k^1 e de 74% para k^2.

A utilização de uma PLI na solução do problema de escalonamento eleva a complexidade da solução a um fator exponencial. Como consequência direta, dependendo do tamanho das instâncias utilizadas nos experimentos, a execução da PLI pode necessitar de muito tempo para concluir. Para avaliarmos o tempo de execução, limitamos em 60 minutos a execução da PLI.

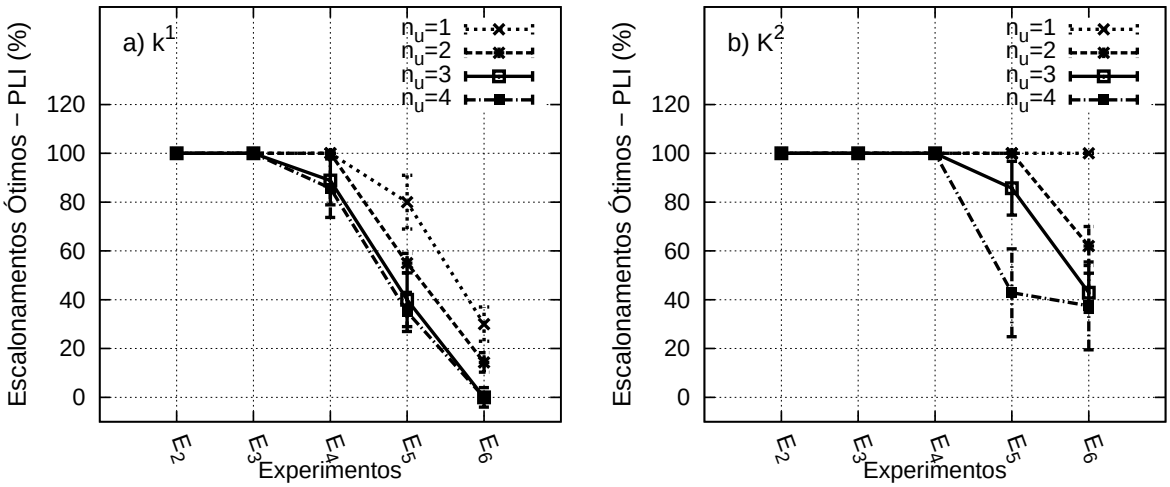


Figura 5.6: Porcentagem de execuções da PLI dentro do limite de tempo de 60 minutos.

A Figura 5.6 apresenta a porcentagem de experimentos que foram concluídos dentro do limite de tempo de execução para k^1, Figura 5.6(a), e para k^2, Figura 5.6(b).

Observando a Figura 5.6, verificamos que para k^1 apenas os Experimentos E_2 e E_3 concluíram a execução da PLI para $1 < n_u \leq 4$. Por outro lado, no Experimento E_6, para $n_u = 3$ e $n_u = 4$, a PLI não concluiu a execução dentro do limite de tempo. Isso é justificado pelo fato de que com mais usuários, a PLI a ser resolvida possui mais restrições (combinações entre possíveis alocações). Neste caso, a solução encontrada pela PLI não é uma solução ótima.

Comparando a porcentagem de execuções dentro do limite de tempo para k^1 e k^2,

respectivamente apresentados nas Figuras 5.6(a) e 5.6(b), observamos que para k^2, o escalonamento ótimo é computado também no Experimento E_4, além de E_2 e E_3, como observado para k^1. Além disso, para $n_u = 3$ e $n_u = 4$ no Experimento E_6, aproximadamente 40% dos escalonamentos computados são ótimos. Enquanto que nenhum escalonamento computado é ótimo em E_6 com $n_u = 3$ e $n_u = 4$ para k^1.

O processo de decomposição das requisições possibilita que mais requisições sejam atendidas, pois, gera novas requisições semelhantes com tempo de duração e necessidade de desempenho proporcionais à original. Contudo, este artifício aumenta a quantidade de requisições que são consideradas durante o escalonamento. Consequentemente, aumenta também o tempo de execução da PLI.

Apresentamos na Figura 5.7 a quantidade de requisições geradas pelo processo de decomposição durante o escalonamento nos cinco experimentos para k^1 e k^2.

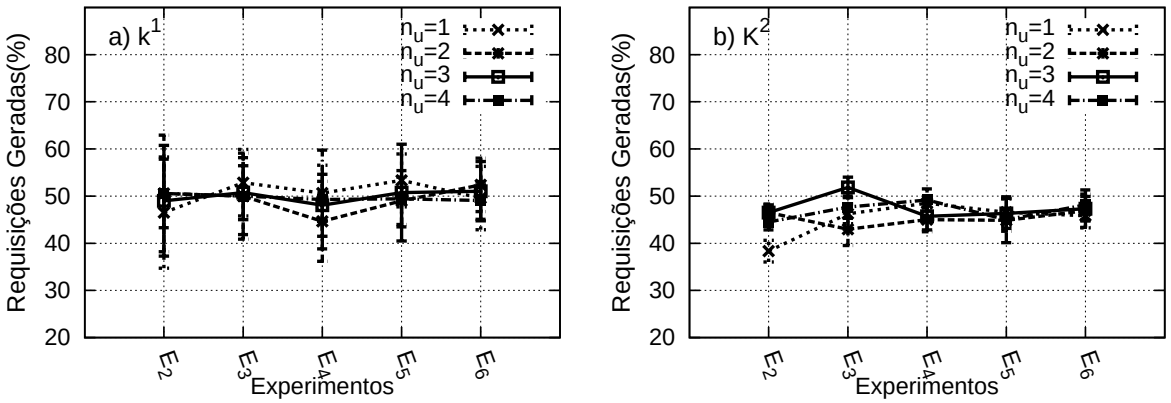


Figura 5.7: Quantidade de requisições geradas pelo processo de decomposição

Observamos na Figura 5.7(a), que para k^1, a quantidade de requisições geradas é próxima de 50% da quantidade de requisições originais de cada experimento. Por outro lado, para k^2, esta quantidade é de aproximadamente 45%.

Cada requisição possui um tempo de duração $r(d)$. O resultado do escalonamento computado pela PLI-1 pode não contemplar todas as requisições, pois, pode não existir VMs suficientes disponíveis para o escalonamento. Desta forma, caso uma requisição não seja atendida, não serão alocadas todas as unidades de tempo solicitadas em $\mathcal{R}^+$.

A Figura 5.8 apresenta a quantidade de unidades de tempo que foram escalonadas em VMs considerando a variação do número de usuários $0 \leq n_u \leq 4$ para k^1 (Figura 5.8-a) e k^2 (Figura 5.8-b). Podemos observar que as maiores quantidades de unidade de tempo atendidas estão nos experimentos com menos requisições (E_2 e E_3), tanto para k^1 como para k^2. Por outro lado, as menores quantidades de unidade de tempo atendidas estão nos experimentos com mais requisições (E_5 e E_6). Este comportamento é um reflexo da quantidade de requisições atendidas apresentada nas Figuras 5.4(a) e 5.5(a).

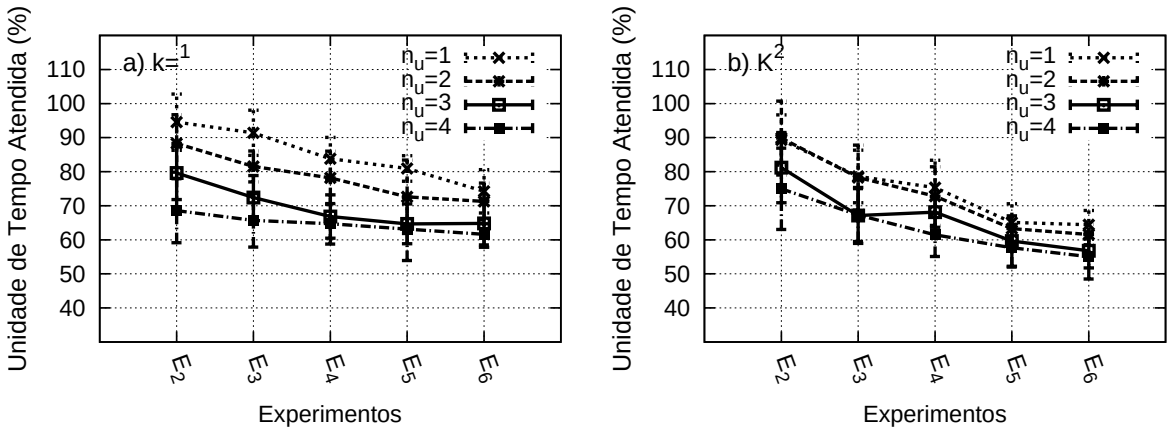


Figura 5.8: Quantidade de unidades de tempo atendida para k^1 e k^2

Os experimentos apresentados nesta seção contribuem para avaliarmos o comportamento do Algoritmo 5 na computação do escalonamento utilizando o modelo de mapeamento conservador. Variamos os três requisitos do usuário abordados nesta tese: Qualidade de Serviço, Desempenho e Segurança. Os resultados permitem avaliarmos a

interferência desta variação no custo e na qualidade do escalonamento computado.

5.4 Escalonamento Utilizando o Mapeamento Flexível

Como discutido anteriormente, o escalonamento computado utilizando o mapeamento conservador proposto na Figura 4.4 garante que os requisitos das requisições não são violados. Como uma alternativa, o modelo de escalonamento considera o mapeamento flexível, apresentado na Figura 4.5. O mapeamento flexível permite que uma requisição FTRt seja alocada sobre uma instância de VM do tipo SP. Em especial, esta possibilidade permite que o custo do escalonamento seja inferior, dado que uma instância SP possui um custo menor comparado ao de uma instância RE. Contudo, este mapeamento negligencia a QoS da requisição ao implantar uma requisição FTRt sobre uma VM SP, pois, os provedores podem interromper o serviço da VM SP. Como uma estratégia para evitar a violação da QoS da requisição e ainda manter a vantagem de utilizar instâncias SP para alocar requisições FTRt, propomos a execução redundante destas requisições.

O sistema de escalonamento gerencia um conjunto de m VMs alugadas (denotadas por $\Psi = \{\Psi_{od} \bigcup \Psi_{re} \bigcup \Psi_{sp}\}$), o qual é utilizado para executar as requisições dos usuários. Para garantir que a requisição FTRt escalonada sobre uma instância SP não tenha sua QoS violada, dividimos o conjunto de VMs alugadas para o escalonamento em duas regiões: Zona de Execução (ZE) e Zona de Redundância (ZR). Estas regiões são apresentadas na Figura 5.9 e definidas em seguida.

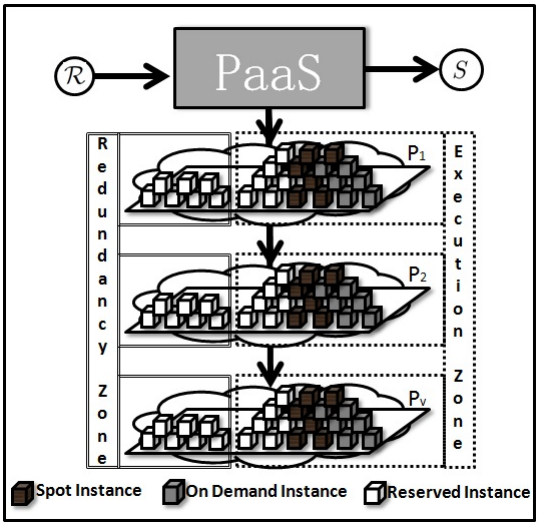


Figura 5.9: Conjunto de instâncias disponíveis para o escalonamento: Zona de Execução e Zona de Redundância

Definimos como Zona de Execução o conjunto Ψ composto por instâncias do tipo OD,

RE e SP de VMs. A Zona de Redundância é definida pelo conjunto Ψ' composto de m' VMs do tipo RE disponíveis para alocar requisições redundantes. Para cada requisição r, com $r(qos_d) = FTRt$, alocada em uma VM do tipo SP na zona de execução, é criada uma cópia de r denotada por r' e escalonada na zona de redundância para o caso da VM que executa r falhar e r não completar. Isto é, a requisição r é duplicada e executada redundantemente [55]. Com isto, a computação do escalonamento requer dois passos: (1) Computação do escalonamento S_{ze} da zona de execução; (2) Computação do escalonamento S_{zr} da zona de redundância. Deste modo, podemos utilizar diferentes estratégias para computar a zona de execução e a zona de redundância. Observe que, caso tenhamos alguma interrupção de VM, é necessário realizar um conjunto de operações para garantir a conclusão das requisições na zona de redundância.

Apresentamos o Algoritmo 6 para computar o escalonamento utilizando o modelo de mapeamento flexível. Os três primeiros procedimentos são similares aos utilizados no Algoritmo 5. Computamos a zona de execução na linha 4. Em seguida, a zona de redundância na linha 5.

Algoritmo 6 Escalonamento Utilizando Mapeamento Flexível

Entrada: $\mathcal{R}^+$, Ψ, n_u, m^g
Saída: S_F
1: $\mathcal{R}^+ \leftarrow$ *Mapeamento_Categorias_de_QoS*($\mathcal{R}^+$, Ω^1)
2: $\mathcal{R}^+ \leftarrow$ *Mapeamento_Desempenho*($\mathcal{R}^+$)
3: $\mathcal{R}^+ \leftarrow$ *Mapeamento_Seguranca*($\mathcal{R}^+$, m^g, n_u)
4: $S_{ze} \leftarrow$ Compute_ZE($\mathcal{R}^+$, Ψ, n_u, Ω^2)
5: $S_{zr} \leftarrow$ Compute_ZR(S_{ze}, σ^c, m^g)
6: return $S_F \leftarrow S_{ze} \bigcup S_{zr}$

A computação da zona de execução pode ser feita utilizando a formulação de PLI apresentada na Seção 5.3.1. Contudo, devemos alterar a função de mapeamento *MF()* utilizada na restrição (R-5.6) para que realize o mapeamento seguindo o modelo flexível.

É importante computar o escalonamento da zona de redundância a um custo baixo, pois, isso influencia no custo total do escalonamento e pode inviabilizar a utilização desta estratégia. Propomos dois algoritmos para computar o escalonamento da zona de redundância:

1. Baseado no Grau de Utilização - BGU

2. Baseado no Grau de Locação - BGL

O Algoritmo BGU utiliza uma heurística baseada na quantidade de unidades de tempo escalonadas em cada VM do tipo SP na zona de execução que recebeu requisições do tipo

RTFt e computa o escalonamento. Pelo fato de ser baseado em uma heurística, não garante a qualidade da solução. Contudo, em alguns cenários específicos, que serão descritos mais adiante, podem oferecer bons resultados em um tempo considerável. Face à falta de garantia na qualidade da solução computada pelo Algoritmo BGU, propomos um segundo algoritmo, denominado BGL, baseado em uma formulação de PLI que diminui a quantidade de unidades de tempo alocadas e consequentemente, o custo pago pela alocação, elevando, contudo, a complexidade assimptótica da solução para um fator exponencial. Nas próximas seções, apresentaremos cada um dos dois algoritmos.

Definimos o *grau de concorrência* (σ^c) na ZR como a quantidade de requisições r' executando concorrentemente, isto é, ao mesmo tempo na mesma VM ψ'. O grau de concorrência é um parâmetro ajustável que permite controlar a relação entre o custo e a disponibilidade. O grau de concorrência σ^c com valor alto pode reduzir o custo do escalonamento pelo fato de utilizar menos VMs, contudo, aumenta a probabilidade de falhas de VMs SP e pode comprometer a QoS. Assumimos que o tempo de execução da requisição r' será $\frac{100}{\sigma^c}\%$ do tempo de execução da requisição r na zona de execução como um resultado da execução concorrente.

5.4.1 Uma Heurística Baseada no Grau de Utilização das VMs

Após a conclusão da computação do escalonamento da zona de execução, o escalonamento computado pode ter gerado situações de risco comprometendo a garantia de QoS no momento em que requisições FTRt são escalonadas para VMs do tipo SP. Como descrito anteriormente, uma requisição FTRt pode sofrer um atraso inicial estabelecido por $r(d\alpha)$ e não pode ser interrompida. Contudo, VMs do tipo SP podem ser interrompida pelo provedor e deste modo, precisamos alocar estas requisições para VMs na zona de redundância. Então elas podem ser ativadas no caso de uma VM SP ser interrompida na zona de execução.

Para computar o escalonamento da zona de redundância, propomos o Algoritmo 7 baseado no grau de utilização da VM resultante do escalonamento da zona de execução. Definimos o *grau de utilização* σ^u_ψ como a quantia de unidades de tempo que uma VM ψ executa uma requisição de acordo com o escalonamento S_{ze} da zona de execução. Por exemplo, suponha que somente as requisições r_1 e r_2 do tipo FTRt com $r_1(d) = 25$ e $r_2(d) = 40$ foram escalonadas, r_1 antes de r_2, para executar na VM ψ_i em S_{ez}. Neste caso, $\sigma^u_{\psi_i} = 65$.

Denotamos como Ψ° o conjunto de VMs do tipo SP para as quais alguma requisição do tipo FTRt foi escalonada na zona de execução. Estas são as VMs que devem ser consideradas durante a computação da zona de redundância. O conjunto Ψ° é computado na primeira linha do Algoritmo 7 utilizando o procedimento *SP_VMC(S_{ze})*. Em seguida,

Algoritmo 7 Escalonamento Baseado no Grau de Utilização - BGU

Entrada: S_{ze}, σ^c, m^g
Saída: Escalonamento S_{rz}

1: $\Psi^\circ \leftarrow$ SP_VMC(S_{ez});{Compute o conjunto de VMs SP que devem ser escalonadas na ZR}
2: $\Psi' \leftarrow$ RE_VMC(Ψ°);{Compute o conjunto de VMs para o escalonamento}
3: **Para todo** $g \in \mathcal{G}$ **faça**
4: $\quad \Psi_g^\circ \leftarrow$ IGG(Ψ°, m^g);{Compute os grupos de isolamento}
5: **fim Para**
6: **Para todo** $g \in \mathcal{G}$ **faça**
7: $\quad$ **Para todo** $\psi \in \Psi_g^\circ$ **faça**
8: $\quad\quad$ Compute σ_ψ^u;{Compute o grau de utilização em S_{ez}}
9: $\quad$ **fim Para**
10: **fim Para**
11: **Para todo** $g \in \mathcal{G}$ **faça**
12: $\quad \Psi_g^\circ \leftarrow$ Sort Ψ_g° by σ^u
13: $\quad \bar{m}_g \leftarrow |\Psi_g^\circ|$
14: $\quad$ **Para** $i = 1$ até $\bar{m}_g$ **faça**
15: $\quad\quad \psi'_{\lceil \frac{i}{\sigma^c} \rceil} \leftarrow \psi'_{\lceil \frac{i}{\sigma^c} \rceil} \bigcup \psi_g^i$ {Compute o escalonamento}
16: $\quad$ **fim Para**
17: **fim Para**
18: **Para todo** $\psi' \in \Psi'$ **faça**
19: $\quad$ **Para** $t = 1$ até T **faça**
20: $\quad\quad$ **Se** $\psi'.t = 1$ **então**
21: $\quad\quad\quad S_{ez}.c = S_{ez}.c + RE_c$;{Compute o custo do escalonamento}
22: $\quad\quad$ **fim Se**
23: $\quad$ **fim Para**
24: **fim Para**

na linha 2, utilizamos o procedimento *RE_VMC(Ψ°)* para computar o conjunto de VMs do tipo RE necessário para escalonar as requisições de Ψ° na zona de redundância. Quanto maior for o grau de concorrência σ^c, menor será a quantidade de VMs no conjunto Ψ', pois, sendo $m^\circ = |\Psi^\circ|$, então, $m' = |\Psi'| \leq \frac{m^\circ}{\sigma^c}$. Isto é, a maior quantidade de VMs necessárias para alocar as requisições com redundância é dada por $\frac{m^\circ}{\sigma^c}$. O valor de m' depende do modelo de geração de grupos de isolamento utilizado e também das características das requisições escalonadas na zona de execução.

Na linha 3 computamos Ψ_g°, $g \in \mathcal{G}$, os grupos de isolamento de acordo com o modelo de grupos definido em m^g.

O grau de utilização σ^u para cada grupo de isolamento é computado na linha 6. Então, ordenamos o conjunto Ψ_g° considerando o grau de utilização σ^u para cada grupo na linha

12 e computamos o escalonamento utilizando o conjunto ordenado na linha 15. O custo monetário do escalonamento é dado pelo somatório das unidades de tempo nas quais alguma VM ψ' recebeu uma alocação de requisição, multiplicado pelo custo de alugar cada unidade de tempo. Então, na linha 17 computamos o custo do escalonamento $S_{rz}.c$.

Denotamos $\psi'.t$ como uma unidade de tempo de uma VM tal que:

$$\psi.t = \begin{cases} 1, & \text{If } \psi_i \text{ está executando uma requisição no tempo } t; \\ 0, & \text{Caso contrário.} \end{cases}$$

Deste modo, o valor de $S_{rz}.c$ é computado como:

$$S_{rz}.c = \sum \psi'.t \times RE_c; \quad \forall \psi' \in \Psi', \quad 0 < t \leq T \tag{5.11}$$

Para explorarmos as características do Algoritmo 7, considere um exemplo de escalonamento denotado por S_{ze} computado na primeira fase apresentado na Figura 5.10. Suponha que temos $\{\psi_1, \psi_2, \psi_3 \text{ e } \psi_4\} \in \Psi^\circ$ (na zona de execução). Utilizamos o modelo de geração de grupos completamente compartilhado (Figura 4.7-d).

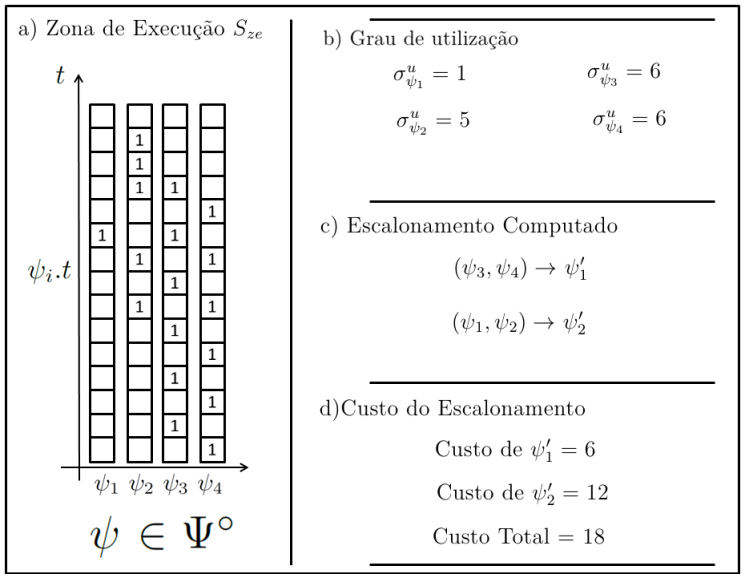


Figura 5.10: Um exemplo ilustrativo da computação do escalonamento utilizando o Algoritmo BGU

A Figura 5.10(b) apresenta o grau de utilização σ^u para as VMs ψ, $\psi \in \Psi^\circ$. O escalonamento computado utilizando o grau de utilização é apresentado na 5.10(c). O custo do escalonamento da zona de redundância utilizando o Algoritmo BGU, apresentado na 5.10(d), é dado por: $\psi'_1 + \psi'_2 = 6 + 12 = 18$.

A complexidade assintótica do Algoritmo BGU é dominada pela computação do grau de utilização σ^u e pela ordenação do conjunto Ψ°, resultando em $O(mT + m \log m)$,

onde T é o tempo total considerado para a disponibilidade das VMs. Embora tenha uma complexidade polinomial, utilizando uma heurística, ele não garante a qualidade da solução. Isso acontece pelo fato de que ele considera apenas a quantidade de unidades de tempo utilizadas em uma VM, e não o instante de utilização.

Mostraremos na próxima seção que o escalonamento computado na Figura 5.10(c) não possui o menor custo.

5.4.2 Configuração dos Experimentos

Implementamos o Algoritmo 6 para computar o escalonamento considerando o modelo de mapeamento flexível. Utilizamos a PLI-1 para computar a Zona de Execução e o Algoritmo 7 para computar a zona de redundância, respectivamente, nas linhas 4 e 5 do Algoritmo 6. Como descrito anteriormente, alteramos o procedimento *MF()* da PLI-1 para considerar o mapeamento flexível durante a computação. As métricas observadas são:

- o custo do escalonamento;
- a quantidade de requisições atendidas; e
- a quantidade de VMs utilizadas.

Realizamos 10 experimentos, E_i, $1 \leq i \leq 10$, cada um com 30 conjuntos de requisições, $E_i = \{R_1, R_2, ..., R_{30}\}$. Em cada experimento, o número de requisições em cada conjunto é E_i: $|R_j| = i \times 10$, $1 \leq j \leq 30$. Além disso, configuramos o tempo de execução $r(d) \leftarrow$ *random*(1,20) e o tempo de relaxamento $r(d\alpha) \leftarrow$ *random*(1, 80)+$r(d)$ de cada requisição em cada conjunto $R_j \in E_i$.

Como apresentado na Tabela 5.2, combinamos dois provedores em dois cenários de custo denominados CP_1 e CP_2. O custo de cada VM é proporcional ao custo atualmente praticado pela Amazon EC2 e idêntico ao apresentado na Tabela 5.1.

Tabela 5.2: Configuração de VMs disponíveis para o escalonamento.

CP	**Provedor**	**Desempenho**	**OD**	$	**RE**	$	**SP**	$	**TOTAL**
1	1	1ρ	0	0	0	0	6	8.00	12
	2	2ρ	0	0	0	0	6	16.00	
2	1	1ρ	1	70.00	1	40.00	6	8.00	16
	2	2ρ	1	140.00	1	100.00	6	16.00	

Variamos a composição do conjunto $\mathcal{R}^+$ executando os experimentos em dois cenários denominados CR_1 e CR_2. O experimentos foram executados considerando requisições de até quatro usuários distintos sobre o modelo de geração de grupos completamente isolados (Figura 4.7-a).

Tabela 5.3: Composição do conjunto de requisições.

CR	FTRx (%)	FTRt (%)	VTR (%)
1	0	100	0
2	20	60	20

Avaliação dos Resultados Obtidos Considerando Apenas Requisições Flutuantes Sobre VMs Spot

Realizamos a execução dos 10 experimentos considerando o cenário de provedores disponíveis CP_1 e a configuração de requisições CR_1. Os resultados evidenciam o potencial da utilização das VMs do tipo SP no escalonamento. A Figura 5.11 apresenta o custo do escalonamento considerando requisições pertencentes a 1, 2, 3 e 4 usuários: $a) n_u = 1$, $b) n_u = 2$, $c) n_u = 3$ e $d) n_u = 4$.

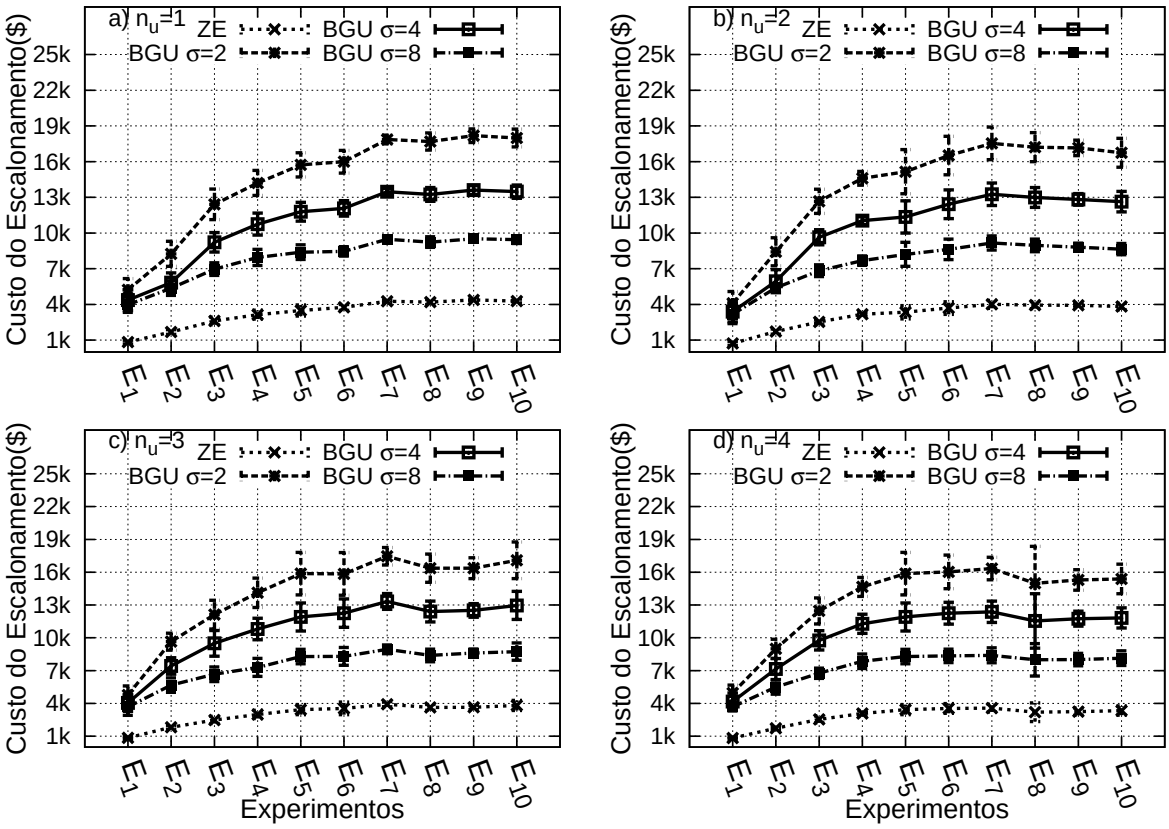


Figura 5.11: CP_1 e CR_1: Custo do escalonamento com grupos de requisições pertencentes a 1, 2, 3 e 4 usuários distintos

Podemos observar que o custo do escalonamento da zona de execução é baixo. Já o

custo total do escalonamento, considerando a computação das zonas de execução e de redundância, é maior quando utilizamos o grau de concorrência $\sigma^c = 2$ e menor com $\sigma^c = 8$. Seja c_{n_x}, $0 < x \leq 4$, o custo do escalonamento de requisições pertencentes a x usuários. De maneira geral, podemos observar na Figura 5.11 que o custo do escalonamento obedece a seguinte relação: $c_{n_1} < c_{n_2} < c_{n_3} < c_{n_4}$. Isso é justificado pelo fato de que quando temos requisições de diferentes usuários aumentamos as restrições de alocação de uma requisição sobre uma VM e diminuímos as chances das requisições serem alocadas nas VMs com custo menor, pois as requisições de um usuário não podem ser executadas com as de outros usuários (seguindo o modelo de geração de grupos isolamento completo, Figura 4.7-a). Consequentemente, são utilizadas também as VMs com maior custo de alocação. Por outro lado, caso utilizássemos o modelo de geração de grupos compartilhamento completo Figura 4.7-d) a diferença seria menos expressiva pelo fato de que muitas requisições concorreriam na mesma VM.

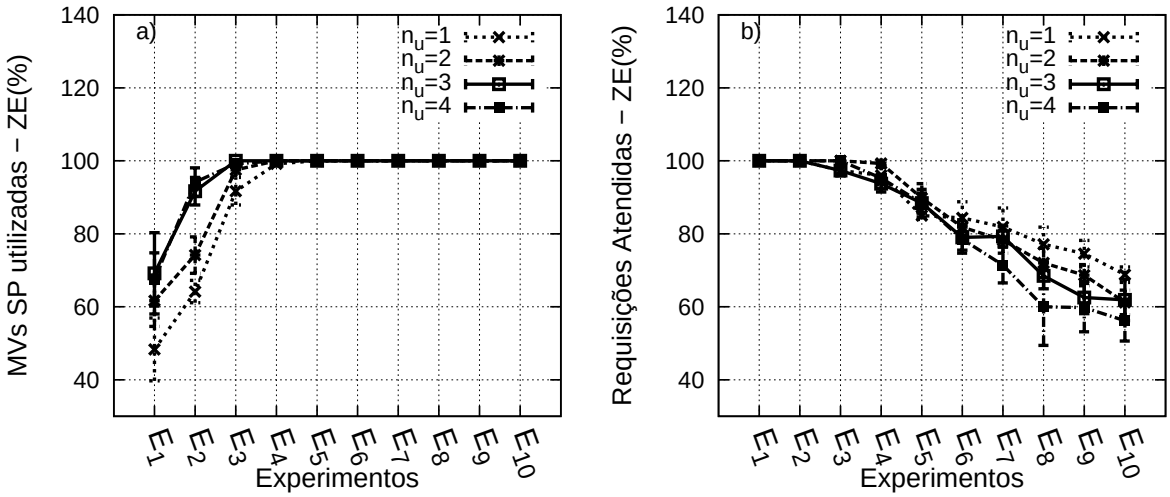


Figura 5.12: CP_1: Quantidade de requisições atendidas e VMs utilizadas

A Figura 5.12 (b) apresenta a quantidade de requisições atendidas. Podemos observar que, embora o escalonamento para apenas um usuário (n_1) tenha um custo maior, são

atendidas mais requisições, quando comparado aos resultados do escalonamento para n_2, n_3 e n_4. Isso justifica o custo mais alto. Para $n_u = 4$, temos o menor custo. Contudo, a quantidade de requisições atendidas é menor. A Figura 5.12 (a) apresenta a quantidade de VMs do tipo *spot* que foram utilizadas durante a computação do escalonamento da zona de execução, as quais devem ser escalonadas redundantemente na zona de redundância.

Podemos observar que nos experimentos E_1 e E_2 são utilizados menos de 100% da quantidade de VMs *spot* disponíveis. Desta forma, a quantidade de requisições atendidas, apresentada na Figura 5.12 (b), atinge 100% para os experimentos E_1 e E_2. Quando são utilizadas 100% das VMs *spot* na zona de execução (Figura 5.12-a), não são atendidas 100% das requisições, isso acontece nos experimentos de E_3 a E_{10}, pois a quantidade de requisições é maior e a quantidade de VMs disponíveis não é suficiente.

Avaliação dos Resultados Obtidos Considerando Requisições com Diferentes QoS sobre VMs de Diferentes Tipos

Nesta seção, apresentamos os resultados obtidos na computação do escalonamento utilizando diferentes categorias de QoS e diferentes tipos de VMs dos provedores. Consideramos a configuração das requisições denotada por CR_2 descrita na Tabela 5.3, onde temos os três tipos de categorias de QoS. Estas requisições são escalonadas em um conjunto de VMs dos provedores pertencentes ao CP_2, Tabela 5.2.

Validamos a composição da zona de redundância observando diferentes graus de concorrência como $\sigma^c = \{2, 4, 8\}$. Os 10 experimentos foram executados para $1 \leq n_u \leq 4$ e k^1, isto é, todas as requisições necessitam de apenas uma unidade de processamento (1ρ).

A Figura 5.13 apresenta um comparativo entre os custos dos escalonamentos computados utilizando o Algoritmo 5 e o Algoritmo 6. Utilizamos a PLI-1 e o Algoritmo 7 para computar, respectivamente, a zona de execução e a zona de redundância. O comportamento dos quatro gráficos apresentados na Figura 5.13 é semelhante. Podemos observar que para os experimentos com menos requisições temos um custo menor e para experimentos com mais requisições, temos um custo maior. Em linhas gerais, o escalonamento para apenas um usuário $n_u = 1$ possui os menores valores. Já com o quatro usuários $n_u = 4$ possui o maior custo. Este comportamento é similar ao apresentado no cenário utilizando CP_1 e CR_1.

A curva denotada por MC apresenta os resultados obtidos pela execução dos experimentos utilizando o mapeamento conservador (Algoritmo 5). A curva denotada por ZE apresenta os resultados do escalonamento da zona de execução utilizando o mapeamento flexível (Algoritmo 6) sem o custo da zona de redundância. Os resultados obtidos na computação da zona de redundância utilizando o Algoritmo BGU são apresentados pelas curvas $BGU\sigma^c = 2$, $BGU\sigma^c = 4$ e $BGU\sigma^c = 8$ para diferentes valores para o grau de concorrência σ^c. Analisando estes dados, observamos que para $n_u = 1$ (Figura 5.13-a), a

estratégia de utilizar o mapeamento flexível apresenta um custo mais baixo comparado ao custo obtido pelo mapeamento conservador, exceto no Experimento E_{10} utilizando $\sigma^c = 2$, onde temos muitas requisições e o custo de computar a zona de redundância é alto. Um comportamento semelhante pode ser observado para $n_u = 2$, (Figura 5.13-b). Contudo, não é vantagem utilizar o mapeamento flexível com $\sigma^c = 2$ nos experimentos E_6, E_7, E_8, E_9 e E_{10}

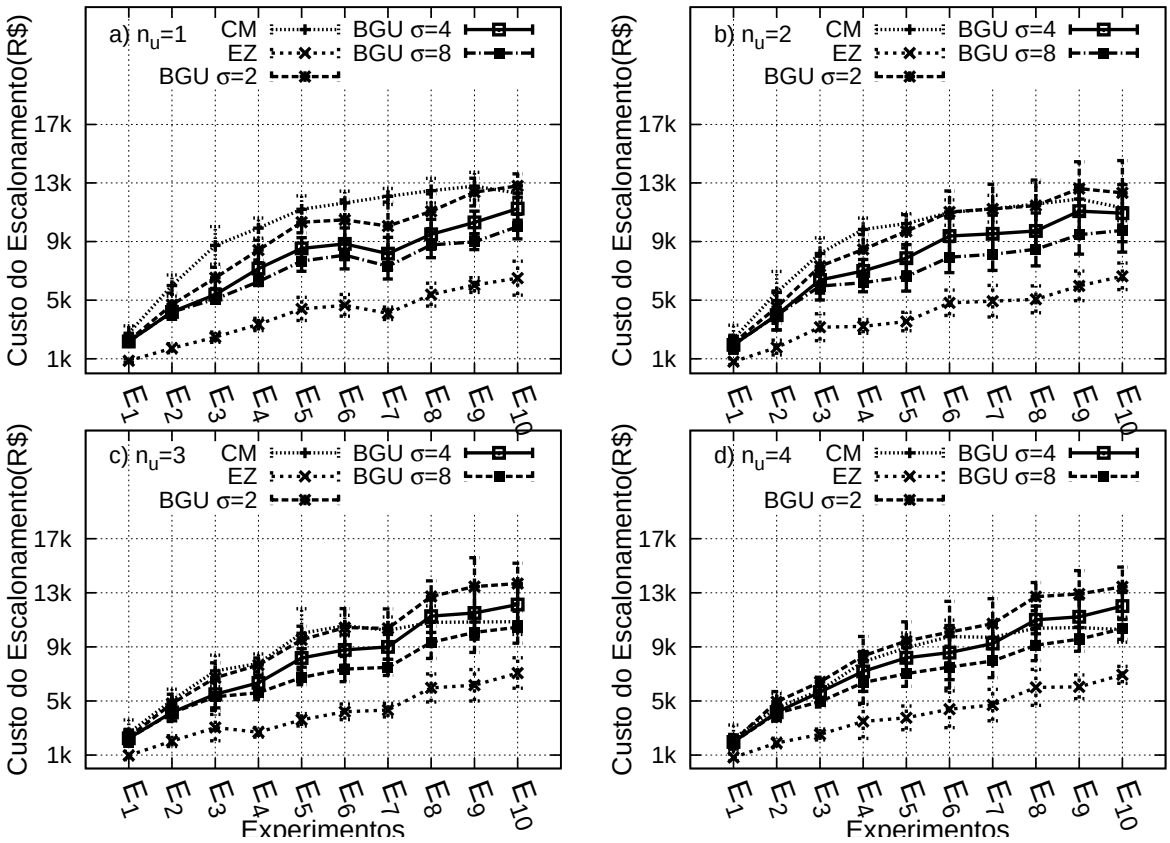


Figura 5.13: Comparativo entre o custo do escalonamento: Mapeamento Conservador e Mapeamento Flexível

No cenário com quatro usuários (Figura 5.13-d), todos os experimentos com $\sigma^c = 4$ e $\sigma^c = 8$ obtiveram um custo menor comparado ao obtido pelo escalonamento utilizando o mapeamento conservador. Além isso, 100% dos experimentos com $\sigma^c = 2$ têm resultados melhores utilizando o mapeamento conservador.

A Figura 5.14(a) apresenta a quantidade de requisições atendidas e a quantidade de VMs utilizadas para os tipos OD (Figura 5.14-b), RE (Figura 5.14-c) e SP (Figura 5.14-d) obtidas utilizando o mapeamento conservador. A Figura 5.15 apresenta as mesmas informações, contudo, os dados foram obtidos utilizando o mapeamento flexível.

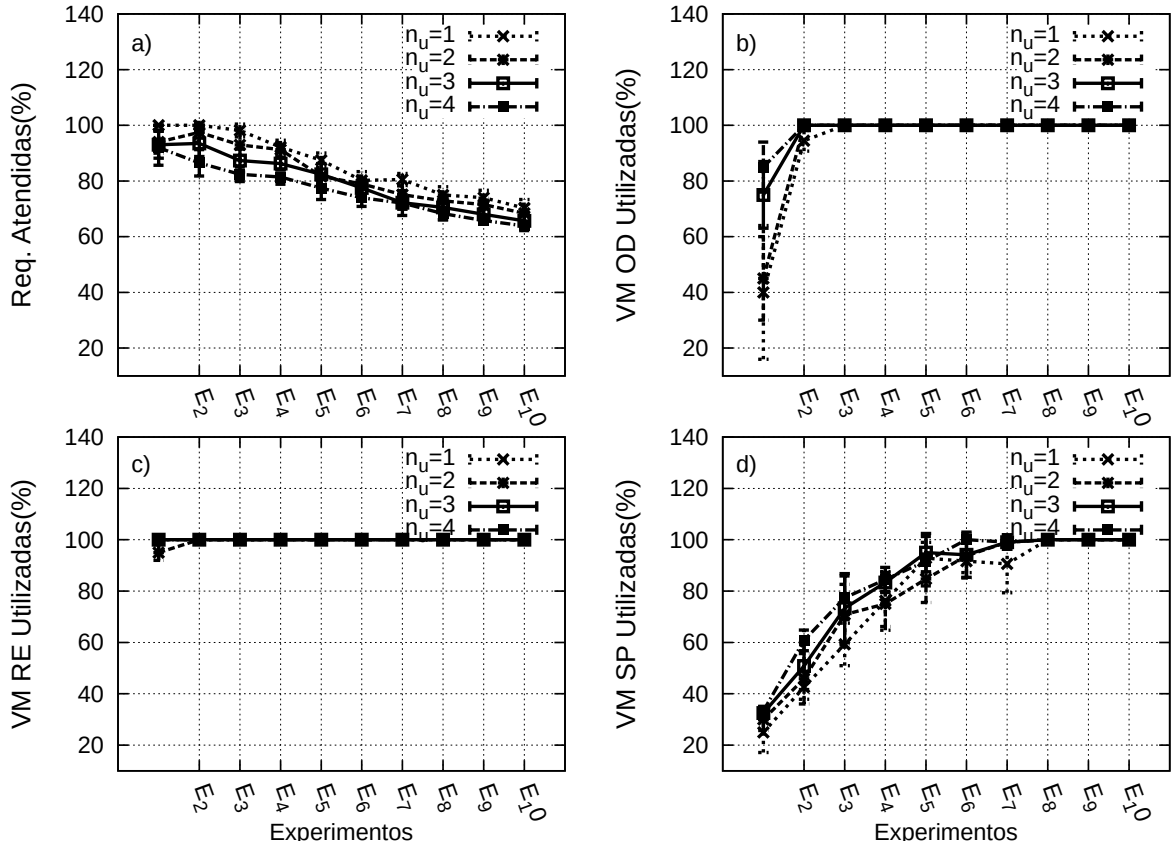


Figura 5.14: Algoritmo 5: Quantidade de requisições atendidas e VM utilizadas

Comparando os resultados apresentados nas Figura 5.14(a) e 5.15(a), observamos que o escalonamento utilizando o mapeamento flexível 5.15(b) atende mais requisições que o mapeamento conservador 5.14(a). A quantidade de requisições e VMs disponíveis é a mesma, visto que estamos utilizando o cenário onde temos CR_2 e CP_2. Desta forma, o que influencia nesta diferença é exatamente o fato de que o mapeamento flexível oferece mais chances de alocação do que o conservador, pois o mapeamento flexível permite que requisições RTFt sejam alocadas em VMs do tipo SP. Em geral, temos que no cenário com apenas um usuário mais requisições são atendidas, pois temos menos restrição de isolamento das requisições quando comparado ao cenário com quatro usuários.

A quantidade de VMs do tipo OD utilizadas nos dois cenários, Figura 5.14(b) e 5.15(b), é de 100% a partir do Experimento E_4. Já nos Experimentos E_1, E_2, E_3 e E_4, verificamos uma expressiva diferença quando comparamos a quantidade utilizada no mapeamento conservador e no mapeamento flexível. O mapeamento flexível utiliza menos requisições OD. Para o Experimento E_1, foram utilizadas aproximadamente 10% das VMs no mapeamento flexível e 41% no mapeamento conservador para $n_u = 1$. Esta dife-

rença tem um impacto no custo, pois, as VMs OD têm o maior custo. Por outro lado, a quantidade de VMs do tipo SP utilizada no mapeamento conservador, Figura 5.14(d), é inferior à utilizada no mapeamento flexível, Figura 5.15(d). Desta forma, observamos que o mapeamento conservador utiliza mais VMs do tipo OD do que o mapeamento flexível; e menos VMs do tipo SP do que o mapeamento flexível. Este comportamento contribui para que o escalonamento computado utilizando o mapeamento flexível tenha um custo menor.

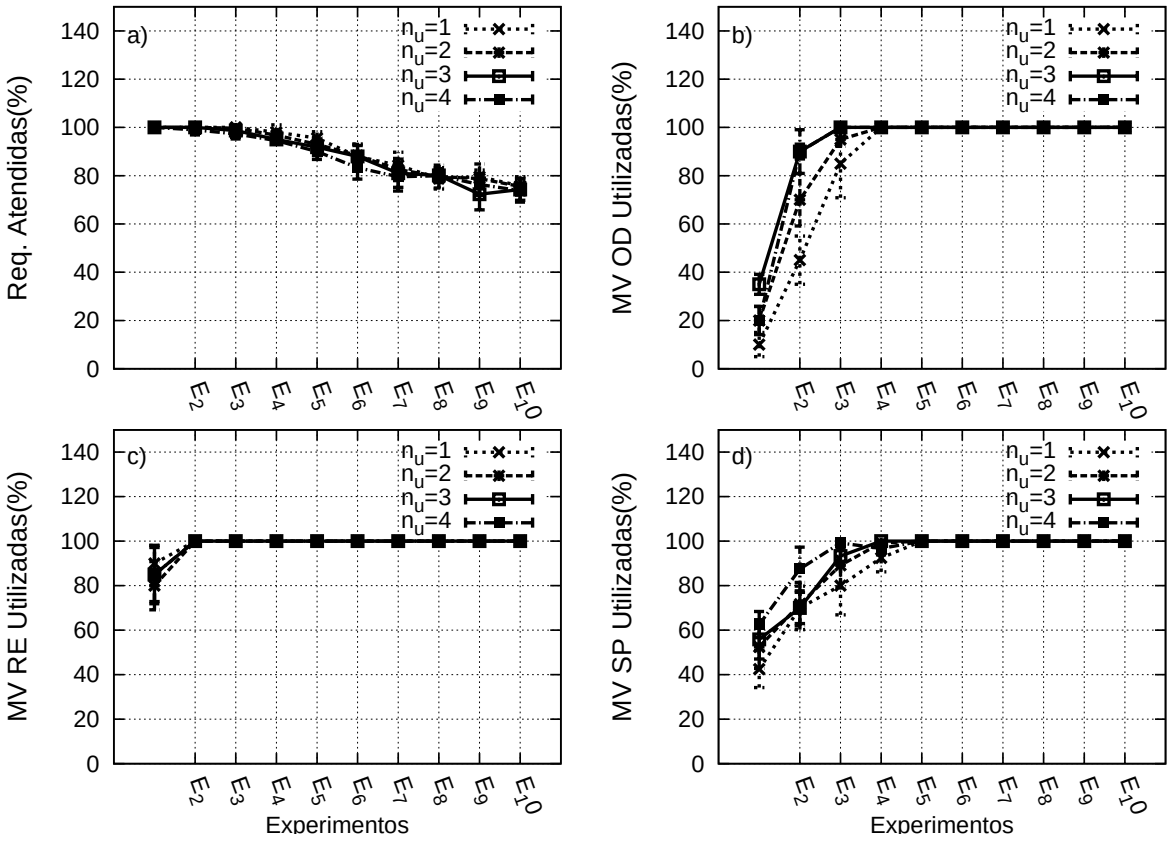


Figura 5.15: Algoritmo 6: Requisições Atendidas e quantidade de VMs

A quantidade de VMs do tipo RE é semelhante nos dois cenários, Figura 5.14(c) e 5.15(c), atingindo 100% de utilização do Experimento E_2 em diante.

5.4.3 Uma Abordagem Baseada no Grau de Locações das VMs

A estratégia descrita anteriormente para computar o escalonamento na zona de redundância baseada no grau de utilização é uma abordagem que produz bons resultados, mas não garante a qualidade da solução.

Como uma opção ao Algoritmo BGU, apresentamos o Algoritmo 8, denominado Baseado no Grau de Locação (BGL) [56], para computar quais VMs devem ser alocadas concorrentemente com a finalidade de reduzir o custo do escalonamento na zona de redundância.

Em contraste com o Algoritmo BGU, o algoritmo utiliza uma métrica diferente, denominada grau de locação, para agrupar as requisições r' na zona de redundância. Esta métrica permite a priorização da consolidação da VM que hospedará as requisições e, consequentemente, reduz o custo do escalonamento.

A principal ideia do Algoritmo BGL é combinar (ou consolidar) as VMs $\psi_i \in \Psi^\circ$ em grupos de tamanho σ^c tal que, exista uma maior intersecção nas unidades de tempos utilizadas. Isto aumenta a consolidação da zona de redundância, consequentemente, diminui o custo pelo fato de que, globalmente, o tempo total de utilização das VMs na zona de redundância é minimizado.

Definimos $\mathcal{C}^{\bar{m}}_{\sigma^c}$ como a combinação de VMs $\psi_i \in \Psi^\circ$, $1 < i < \bar{m}$ em grupos com σ^c elementos cada. A partir disto, a quantidade de combinações possíveis para serem consideradas para a consolidação pelo Algoritmo 8 é dada por:

$$\gamma = \frac{\bar{m}!}{(\bar{m} - \sigma^c)! \times \sigma^c!}$$

Seja $c_i = \{c_{(1)}, c_{(2)}, c_{(3)}, ...c_{(\sigma^c)}\} \in \mathcal{C}^{\bar{m}}_{\sigma^c}, 1 \leq i \leq \gamma$. Definimos o grau de ocupação $\sigma^o_t(c_i)$ de uma combinação de VMs $c_i \in \mathcal{C}^{\bar{m}}_{\sigma^c}$ no tempo t como

$$\sigma^o_t(c_i) = \begin{cases} 1, & \text{Se } \exists\ \psi.t = 1,\ \psi \in c_i \\ 0, & \text{Caso contrário.} \end{cases}$$

Se $\sigma^o_t(c_i) = 1$, então existe um custo por alugar as máquinas na zona de redundância. Deste modo, definimos o grau de locação (σ^t_c) de uma combinação c como

$$\sigma^t_c = \sum_{t \leftarrow 1}^{T} \sigma^o_t(c)$$

Isto é, computamos o grau de ocupação de um conjunto de VMs em uma unidade de tempo. O grau de locação é o somatório do grau de ocupação considerando todas as unidades de tempo.

O Algoritmo 8, apresentado a seguir, computa o escalonamento da zona de redundância considerando as combinações possíveis de VMs e o grau de locação em cada combinação.

Inicialmente, na linha 1, computamos o conjunto Ψ° utilizando o procedimento *SP_VMC()* e o tamanho deste conjunto na linha 2. Em seguida, na linha 3, computamos todas as combinações do conjunto Ψ° utilizando o procedimento *Computa_Combinações()*. Cada combinação tem um tamanho máximo de σ^c. A quantidade de combinações é computada na linha 4.

Algoritmo 8 Escalonamento Baseado no Grau de Locação

Entrada: S_{ez}, σ^c
Saída: S_{rz}
1: $\Psi^\circ \leftarrow$ SP_VMC(S_{ez});{Compute o conjunto de VMs SP que devem ser escalonadas na ZR}
2: $m^\circ \leftarrow |\Psi^\circ|$;
3: $\mathcal{C}_{\sigma^c}^{m^\circ} \leftarrow$ Computa_Combinações($\Psi^\circ, m^\circ, \sigma^c$);
4: $\gamma \leftarrow |\mathcal{C}_{\sigma^c}^{m^\circ}|$;
5: $V_{1,2,...,\gamma} \leftarrow$ Vetor;
6: **Para** $i = 1$ até γ **faça**
7: $\quad V[i] \leftarrow 0$;
8: **fim Para**
9: **Para** $t = 1$ até T **faça**
10: $\quad$ **Para each** $c \in \mathcal{C}_{\sigma^c}^{m^\circ}$ **faça**
11: $\quad\quad V[c] \leftarrow V[c] + \sigma_t^o(c)$;
12: $\quad$ **fim Para**
13: **fim Para**
14: **Para each** $c \in \mathcal{C}_{\sigma^c}^{m^\circ}$ **faça**
15: $\quad \mathcal{D}_c \leftarrow$ Elimina_Duplicações($c, \mathcal{C}_{\sigma^c}^{m^\circ}$);
16: **fim Para**
17: $S_{rz} \leftarrow$ PLI-ZR($\mathcal{C}_{\sigma^c}^{m^\circ}, V, \mathcal{D}_c$)
18: return S_{rz};

Na linha 5, o algoritmo reserva um vetor V com γ posições para armazenar o grau de locação σ^t de cada combinação. O grau de ocupação em cada unidade de tempo é computado na linha 11. O Somatório de todas as unidades de tempo, computado entre as linhas 9 e 13, corresponde ao grau de locação.

Um total de $\frac{m^\circ}{\sigma^c}$ combinações serão utilizadas no escalonamento, uma para cada VM $\psi' \in \Psi'$. Uma VM contida em uma dada combinação c não pode estar contida em outra combinação no escalonamento, isto duplicaria a execução dela na zona de redundância. A função *Elimina_Duplicações* (linhas 14-16), descrita no Algoritmo 9, computa todas as combinações que não devem estar na solução se a combinação c estiver, e armazena o resultado em uma variável que é passada como um parâmetro para a programação linear de inteiros (denominada PLI-ZR) que computa o escalonamento na linha 17 do Algoritmo 8. Esta PLI computa as $\frac{\bar{m}}{\sigma^c}$ combinações que juntas produzem o menor valor global do grau de locação.

É importante observar que escolher as combinações segundo o grau de locação local para compor o escalonamento S_{rz}, não necessariamente produzirá o menor grau de locação global.

O Algoritmo 9 gera um conjunto de combinações $\mathcal{D}_c$ que devem ser excluídas da solução

Algoritmo 9 Elimina_Duplicações

Entrada: c, $\mathcal{C}_{\sigma^c}^{m^\circ}$
Saída: $\mathcal{D}_c$

```
1: Para each ψ ∈ c faça
2:   Para each c_i ∈ C_{σ^c}^{m°} faça
3:     Se ψ ⊂ c_i então
4:       D_c ← D_c ⋃ c_i
5:     fim Se
6:   fim Para
7: fim Para
8: return D_c;
```

caso a combinação c esteja na solução. Para cada VM ψ pertencente à combinação c, todas as outras combinações que contenham ψ são incluídas no conjunto $\mathcal{D}_c$.

A PLI-ZR, descrita a seguir, resolve o problema de minimização do grau de locação global utilizando as variáveis binárias x e y e as constantes V_i e U_j como a seguir:

- x_i: Variável binária que assume o valor 1 se a combinação i é selecionada. Caso contrário, assume o valor 0;
- V_i: Constante que assume o grau de locação $\sigma_{c_i}^t$ da combinação c_i;
- y_j: Variável binária que assume o valor 1 se a VM j é selecionada; Caso contrário, assume o valor 0;
- U_j: Constante que assume o valor do grau de utilização $\sigma_{\psi_j}^u$ da VM ψ_j;

A função objetivo $F_{zr} = \sum_{i=1}^{\gamma}(x_i \times V_i \times C_{re}) + (\sum_{i=j}^{m^\circ}(1-y_j) \times U_j \times C_{re})$ computa as possíveis combinações e o respectivo grau de locação. Desta modo, desejamos minimizar a Função F_{zr} sujeito às seguintes restrições:

$$\sum_{i=1}^{\gamma} x_i = \frac{m^\circ}{\sigma^c}; \tag{5.12}$$

$$\mathcal{D}(x_i) \leq M \times (1 - x_i); \forall i \in \left[1, \gamma\right] \tag{5.13}$$

$$\sum_{j=1}^{\sigma^c} y_j = x_i \times \sigma^c; \forall i \in \left[1, \gamma\right] \tag{5.14}$$

A PLI-ZR utiliza as variáveis binárias x e y, e as constantes V_i e U_j para computar o menor grau de locação global. A constante C_{re} assume o valor do custo de aluguel de

uma instância de VM RE por uma unidade de tempo. A restrição (R-5.12) especifica que somente $\frac{m^\circ}{\sigma^c}$ combinações devem ser selecionadas. A restrição (R-5.13) especifica que se a combinação x_i é selecionada, todas as combinações do conjunto de combinações eliminadas de x_i também devem ser. A função $\mathcal{D}(x)$ contém o conjunto de combinações eliminadas de x_i computado pelo Algoritmo *Elimina_Duplicações()*. A constante M é utilizada como uma técnica para considerar a restrição quando x_i é selecionada. Caso x_i seja selecionada, $x_i = 1$, então $M \times (1 - x_i) = 0$ e todas as restrições em $\mathcal{D}(x_i)$ devem ser falsas. A restrição (R-5.14), auxilia na segunda parte da função objetivo para computar o conjunto de VMs que não estão contidas em alguma combinação, e deste modo, não farão parte da zona de redundância. Isto é especialmente importante quando o valor de m° não é um múltiplo de σ^c. Neste caso, algumas VMs estarão fora do escalonamento na redundância. Quando o Algoritmo 8 é concluído, as VMs contidas em uma combinação selecionada na solução da PLI-ZR devem ser escalonadas em uma mesma VM na zona de redundância.

Para ilustrar as características do Algoritmo 8, apresentamos na Figura 5.16 um exemplo da computação da zona de redundância após a conclusão da fase 2.

Como no exemplo apresentado na Figura 5.10, suponha que tenhamos $\{\psi_1, \psi_2, \psi_3$ e $\psi_4\} \in \Psi^\circ$ (na zona de execução). Utilizando $\sigma^c = 2$ seriam necessárias ψ'_1 e ψ'_2 para receber as 4 VMs de Ψ°. Além disso, suponha que cada unidade de tempo tenha um valor monetário de R$1,00 para sua utilização. Utilizando a estratégia de grau de locação, apresentada nesta seção, são computadas as seis combinações das VMs contidas no conjunto Ψ°, Figura 5.16(b).

Para cada combinação c_i, $0 < i \leq 6$, computamos o grau de locação $\sigma^t_{c_i}$, apresentado na Figura 5.10(c). Com isso, podemos escalonar as combinações com o menor σ^t sobre VMs ψ' com o menor custo utilizando menos unidades de tempo. Contudo, somente escolher as combinações com menor grau de locação não é suficiente para produzir o escalonamento com o menor custo global. Observe na Figura 5.10(d) que se utilizarmos a estratégia ineficiente de escolhermos as combinações com menor σ^t, primeiramente selecionaríamos a combinação c_1 com $\sigma^t_{c_1} = 6$. Como a combinação c_1 contém as VMs ψ_1 e ψ_2, $c_1 = \{\psi_1, \psi_2\}$, as combinações c_2, c_3, c_4, c_5 seriam excluídas da solução pelo Algoritmo *Elimina_Duplicações*. A combinação c_6 seria selecionada para a solução. O escalonamento computado, utilizando a estratégia ineficiente de selecionar as combinações com os menores σ^t, seria composta pelas combinações c_1 e c_6 com um custo de, $12 + 6 = 18$. Este não é o menor custo possível para o escalonamento. Por este motivo, utilizamos uma PLI para escolher as combinações que juntas compõem o escalonamento com o menor custo.

Como apresentado na Figura 5.16(e), as combinações selecionadas utilizando a PLI são c_2 e c_5 produzindo um custo de $6 + 9 = 15$ unidades de tempo utilizadas. Este valor é inferior quando comparado aos valores computados pelo Algoritmo 7 e utilizando o grau de locação local.

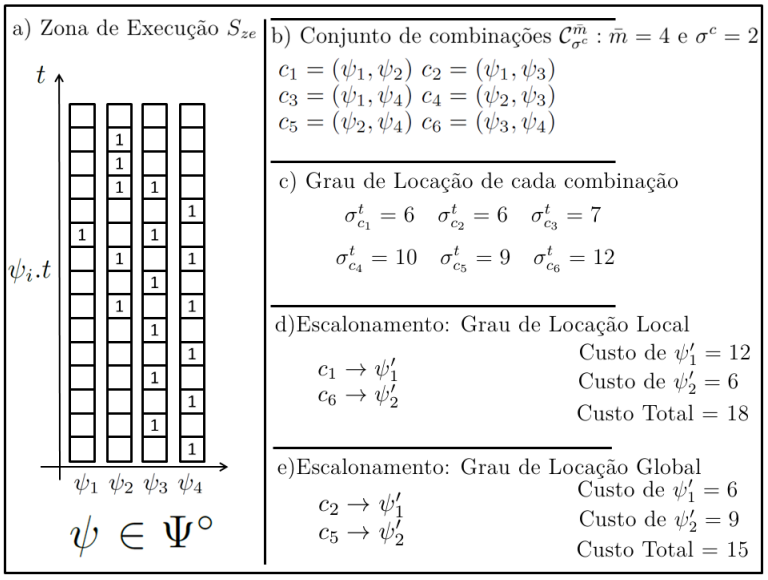


Figura 5.16: Um exemplo ilustrativo da computação do escalonamento utilizando o Algoritmo BGL

Resultados Experimentais

Implementamos o Algoritmo 6 para computar o escalonamento considerando o modelo de mapeamento flexível. Utilizamos a PLI-1 para computar a Zona de Execução e o Algoritmo 8 para computar a zona de redundância, respectivamente, nas linhas 4 e 5 do Algoritmo 6. Como descrito anteriormente, alteramos o procedimento *MF()* da PLI-1 para considerar o mapeamento flexível durante a computação. A métrica observada foi o custo do escalonamento.

Realizamos 10 experimentos, E_i, $1 \leq i \leq 10$, cada um com 30 conjuntos de requisições, $E_i = \{R_1, R_2, ..., R_{30}\}$. Em cada experimento, o número de requisições em cada conjunto é E_i: $|R_j| = i \times 10$, $1 \leq j \leq 30$. Além disso, configuramos o tempo de execução $r(d) \leftarrow$ *random*(1,20) e o tempo de relaxamento $r(d\alpha) \leftarrow$ *random*(1, 80)+$r(d)$ de cada requisição em cada conjunto $R_j \in E_i$.

Como apresentado na Tabela 5.4, recursos de três provedores estão disponíveis para o escalonamento neste experimento.

Estamos considerando VMs com apenas um tipo de desempenho ($\psi(k) = 1\rho$). Todas as requisições a serem escalonadas possuem $r(qos_d) = 1\rho$ e são do tipo FTRt. Além disso, não consideramos os grupos de isolamento. Isto é, todas as requisições pertencem a um único usuário e podem compartilhar recursos.

Tabela 5.4: Conjunto de VMs disponíveis para o escalonamento.

Provedor	**Desempenho**	**OD**	$	**RE**	$	**SP**	$	**TOTAL**
1	1ρ	1	70.00	0	0	0	0	2
2	1ρ	0	0	1	40.00	0	0	4
3	1ρ	0	0	0	0	6	8.00	8
							Total	**14**

Avaliação Comparativa entre os Algoritmos BGU e BGL

Apresentamos na Figura 5.17 uma comparação entre o custo dos escalonamentos obtidos pela execução dos Algoritmos BGU e BGL utilizados para computar a zona de redundância utilizando diferentes graus de concorrência $\sigma^c = \{2, 4, 8\}$.

Na Figura 5.17(a) apresentamos o custo em R$. Na Figura 5.17(b) apresentamos o percentual do custo do escalonamento obtido pelo Algoritmo BGL em relação ao obtido pelo Algoritmo BGU. Podemos observar que o Algoritmo BGL computa o escalonamento com um custo menor. Isso é mais evidente quando consideramos o grau de concorrência $\sigma^c = 2$. Já para $\sigma^c = 8$ os custos são idênticos. Outra observação importante é que quanto menos requisições são consideradas no experimento, mais próximos são os resultados. No Experimento E_1 utilizando $\sigma^c = 4$ os custos são idênticos. Como observado na Figura 5.17(b). Para $\sigma^c = 2$ o escalonamento obtido pelo Algoritmo BGU é aproximadamente 97% do valor obtido pelo Algoritmo BGU. Já para o Experimento E_{10} essa porcentagem diminui para 92%.

5.4.4 Computação do Custo do Escalonamento

Se uma preempção ocorrer, a concorrência na zona de redundância é imediatamente interrompida, deixando a cópia da requisição que foi interrompida na zona de execução concluir sua execução na zona de redundância sozinha permitindo que ela termine dentro do prazo do relaxamento $r(d\alpha)$ sem violar a QoS. Observe que a preempção de duas ou mais requisições na zona de execução que compartilham a mesma VM na zona de redundância causará falha à segunda (ou às posteriores) requisição interrompida, pelo fato de já ter sido excluída da zona de redundância.

Denotamos C_1 e C_2, respectivamente, como os custos dos escalonamentos computados na primeira e na segunda fases. Além disso, denotamos C_3 como o custo para tratar a preempção das VMs durante a execução da requisição. O custo C_3 é computado como a seguir:

$$C_3 = \tau_{eer} \times C_{re} - \tau_{dp} \times C_{sp}, \tag{5.15}$$

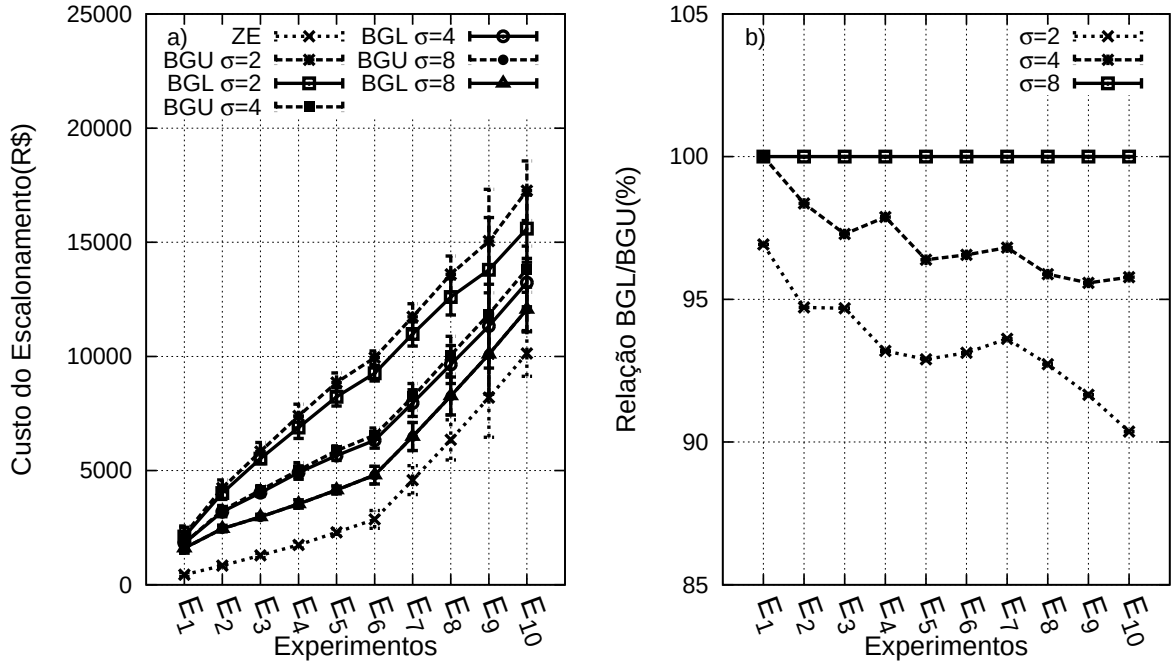


Figura 5.17: Comparação entre o custo dos escalonamentos obtidos pelos Algoritmos 7 e 8.

onde, C_{re} e C_{sp} denotam o custo de utilização por unidade de tempo de uma instância de VM, respectivamente, do tipo RE na zona de redundância e do tipo SP na zona de execução por unidade de tempo.

Se uma preempção ocorrer em uma VM SP, é necessário computar um tempo extra τ_{eer} e um desconto τ_{dp} pela VM interrompida na zona de execução, pois o custo já havia sido computado durante a primeira fase.

Denotamos o custo total do escalonamento por C_t como:

$$C_t = C_1 + C_2 + C_3 \tag{5.16}$$

Se não ocorrem preempções sobre VMs SP, então temos $C_3 = 0$. Caso contrário, a probabilidade de uma requisição não ser atendida por conta de uma preempção de VM é

dada por:

$$P_f = 1 - \frac{m' - \mathcal{F} \times \sigma^c}{m' - \mathcal{F}} \tag{5.17}$$

onde, $\mathcal{F}$ define a quantia de preempções que já ocorreram anteriormente. Por outro lado, quando uma preempção ocorre, é necessário eliminar as requisições concorrentes e manter apenas uma executando na VM. Desta maneira, a probabilidade de um conjunto de requisições alocadas para a mesma VM aumenta com o número de preempções, e esta probabilidade alcança 100% quando o número de VMs interrompidas alcança $(\frac{m'}{\sigma^c} + 1)$.

Capítulo 6

Conclusão

O amadurecimento das tecnologias de virtualização, bem como o crescimento da taxa de transmissão de dados e a popularização da Internet contribuíram para o fortalecimento da computação em nuvem. Este paradigma atrai cada vez mais usuários a consumirem e provedores a oferecerem serviços em nuvem por meio do modelo *pay-as-you-go*. Especialmente, considerando serviços de infraestrutura, usuários alugam capacidade de processamento a partir de provedores de IaaS sob a forma de máquina virtual onde os usuários podem implantar suas aplicações. Com isso, o estudo de escalonar a demanda dos usuários sobre os serviços oferecidos pelos provedores é importante para auxiliar os usuários na seleção dos recursos objetivando o menor custo sem violar a qualidade de serviço de suas demandas. Um escalonamento simples pode resultar em um custo financeiro alto e/ou violações na qualidade de serviço do que for solicitado.

Nesta tese, apresentamos uma PaaS que oferece o serviço de escalonamento de requisições de múltiplos usuários e seleciona os melhores recursos a partir de múltiplos provedores públicos de IaaS. Uma boa caracterização da demanda do usuário, antes de realizar o escalonamento, influencia diretamente na qualidade do escalonamento. Para atingir um resultado eficiente que, além de ter o menor custo, corresponda às expectativas, abordamos três características relacionadas às necessidades do usuário:

1. Confiabilidade: Está relacionada à confiabilidade do serviço considerando se o serviço pode sofrer atraso ou ser interrompido.

2. Desempenho: Está relacionada ao requisito de desempenho das requisições do usuário.

3. Segurança: Está relacionada à execução de VMs ou serviços de maneira compartilhada ou isolada.

Introduzimos na PaaS um novo modelo de cobrança denominado *time-slotted reservation*(TS) que melhora a utilização de instâncias de VMs do tipo RE realizando o agen-

damento da requisição em unidades de tempo consecutivas o mais adiante possível, sem violar a QoS da requisição. Isso possibilita escalonar a requisição e ainda assim, manter unidades de tempo consecutivas livres para que outras requisições sejam escalonadas. A PaaS possui um modelo conceitual de três níveis de mapeamento das necessidades do usuário que permite mapearmos as necessidades dos usuários sobre os serviços oferecidos pelos provedores de IaaS. Com isso, é possível introduzir novos modelos de cobrança na PaaS e também absorver os modelos utilizados pelos provedores. Propomos duas possibilidades de mapeamento do modelo conceitual:

- Mapeamento conservador das necessidades do usuário: Neste mapeamento a QoS das requisições do usuário não é violada. Apresentamos uma formulação de PLI para computar o escalonamento utilizando o mapeamento conservador.

- Mapeamento flexível das necessidades do usuário; É uma outra possibilidade de mapeamento a qual flexibiliza a utilização de VMs do tipo RE e SP para diminuir o custo do escalonamento. Contudo, pode produzir situações com violação da QoS das requisições dos usuários por utilizar instâncias de VMs do tipo SP.

Apresentamos uma estratégia baseada em redundância para diminuir as situações onde ocorrem violações de QoS do mapeamento flexível em que cada requisição com risco de violação é escalonada redundantemente em uma zona de redundância em outra VM do tipo RE. A estratégia baseada em redundância permite evitar as situações de violação de QoS duplicando as requisições e executando-as concomitantemente. Para computar o escalonamento da zona de redundância, propomos dois algoritmos:

- Baseado no grau de utilização: Corresponde a uma heurística em que verifica a quantidade de requisições com possibilidade de violação de QoS e utiliza este parâmetro para computar a zona de redundância.

- Baseado no grau de locação: Corresponde a uma formulação de PLI que objetiva diminuir o custo da locação das VMs que serão utilizadas na zona de redundância.

O algoritmo baseado no grau de utilização possui uma complexidade polinomial. Contudo, não garante a qualidade da solução. Já o algoritmo baseado no grau de locação computa o menor custo. Contudo, possui uma complexidade exponencial.

A PaaS proposta nesta tese auxilia os usuários na seleção de recursos de múltiplos provedores de IaaS. A caracterização das necessidades do usuário (confiabilidade, desempenho e segurança) apresentadas nesta tese, assim como os modelos conservador e flexível de mapeamento das necessidades do usuário, permitem ajustar a qualidade da solução do escalonamento aproximando as necessidades do usuário aos serviços oferecidos pelos

provedores. Sempre objetivando o menor custo sem violar a QoS das requisições. A utilização do modelo conceitual de mapeamento permite que a PaaS seja capaz de absorver as alterações/atualizações dos serviços oferecidos pelos provedores.

O escalonamento de requisições de usuários sobre múltiplos provedores de IaaS é um tema que ainda pode ser abordado de diferentes maneiras, o que abre espaço para uma diversidade de trabalhos futuros. Nesta tese consideramos que o conjunto de recursos oferecidos pelos provedores já está disponível para a PaaS. Mensurar a quantidade de recursos de cada provedor a ser alugada é importante e pode influenciar no custo do escalonamento.

Outro tópico importante a ser abordado é a busca por novas estratégias para computar o escalonamento da zona de redundância com o objetivo de diminuir a possibilidade de violação de QoS da requisição.

Como mencionado anteriormente, o algoritmo baseado no grau de locação possui uma complexidade exponencial. É necessário realizar outros estudos para verificar se ao fixarmos o grau de concorrência, seja possível propor outras soluções com uma complexidade menor. Com o objetivo de aumentar a eficiência das soluções propostas pela PaaS, podemos incluir outras características no SLA utilizado pela PaaS, tais como, localização do recurso, custo de armazenamento.

Como outro trabalho futuro possível, é importante estender a PaaS para considerar o escalonamento *on-line* das requisições dos usuários para melhor atender às necessidades do usuário. Por fim, é necessário introduzir novas funcionalidades, tais como, implantar, monitorar, terminar VMs e oferecer uma interface para múltiplos provedores que traduz estas operações para a API específica de cada provedor no sentido de expandir a PaaS e torná-la um *broker*.

Referências Bibliográficas

[1] Globus toolkit. `http://www.globus.org/`. Acessado: 06/2016.

[2] Projeto aeolus. `http://www.aeolus-project.org/`. Acessado: 06/2016.

[3] Projeto opennebula. `http://opennebula.org/`. Acessado: 06/2016.

[4] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon EC2 spot instance pricing. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 304–311, Nov 2011.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53:50–58, April 2010.

[6] Marcos Dias Assunção, Alexandre Costanzo, and Rajkumar Buyya. A cost-benefit analysis of using cloud computing to extend the capacity of clusters. *Cluster Computing*, 13(3):335–347, September 2010.

[7] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. da Fonseca. Scheduling in hybrid clouds. *Communications Magazine, IEEE*, 50(9):42–47, September 2012.

[8] L. F. Bittencourt, C. R. Senna, and E. R. M. Madeira. Scheduling service workflows for cost optimization in hybrid clouds. In *Proceedings of the International Conference on Network and Service Management (CNSM), 2010*, pages 394 –397, oct. 2010.

[9] Rajkumar Buyya. Market-oriented cloud computing: Vision, hype, and reality of delivering computing as the 5th utility. In *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Xi Chen, Haopeng Chen, Qing Zheng, Wenting Wang, and Guodong Liu. Characterizing web application performance for maximizing service providers' profits in clouds. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*, CSC '11, pages 191–198, Washington, DC, USA, 2011. IEEE Computer Society.

[11] Amit Kumar Das, Tamal Adhikary, Md. Abdur Razzaque, Eung Jun Cho, and Choong Seon Hong. A qos and profit aware cloud confederation model for iaas service providers. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '14, pages 42:1–42:7, New York, NY, USA, 2014. ACM.

[12] W. Dawoud, I. Takouna, and C. Meinel. Increasing spot instances reliability using dynamic scalability. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 959–961, June 2012.

[13] Sheng Di, Yves Robert, Frédéric Vivien, Derrick Kondo, Cho-Li Wang, and Franck Cappello. Optimization of cloud task processing with checkpoint-restart mechanism. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 64:1–64:12, New York, NY, USA, 2013. ACM.

[14] V. Di Valerio, V. Cardellini, and F. Lo Presti. Optimal pricing and service provisioning strategies in cloud systems: A stackelberg game approach. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 115–122, June 2013.

[15] H.M. Fard, R. Prodan, and T. Fahringer. A truthful dynamic workflow scheduling mechanism for commercial multicloud environments. *Parallel and Distributed Systems, IEEE Transactions on*, 24(6):1203–1212, June 2013.

[16] I. Foster, Yong Zhao, I. Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov 2008.

[17] Alexei A. Gaivoronski, Darijus Strasunskas, Per J. Nesse, Stein Svaet, and Xiaomeng Su. Modeling and economic analysis of the cloud brokering platform under uncertainty: Choosing a risk/profit trade-off. *Serv. Sci.*, 5(2):137–162, June 2013.

[18] S. Genaud and J. Gossa. Cost-wait trade-offs in client-side resource provisioning with elastic clouds. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 1–8, July 2011.

[19] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira. Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 906 –912, april 2012.

[20] Thiago A. L. Genez, Luiz F. Bittencourt, and Edmundo R. M. Madeira. On the performance-cost tradeoff for workflow scheduling in hybrid clouds. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, UCC '13, pages 411–416, Washington, DC, USA, 2013. IEEE Computer Society.

[21] Tian Guo, Upendra Sharma, Timothy Wood, Sambit Sahu, and Prashant Shenoy. Seagull: Intelligent cloud bursting for enterprise applications. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 33–33, Berkeley, CA, USA, 2012. USENIX Association.

[22] Ines Houidi, Marouen Mechtri, Wajdi Louati, and Djamal Zeghlache. Cloud service delivery across multiple cloud platforms. In *Proceedings of the 2011 IEEE International Conference on Services Computing*, SCC '11, pages 741–742, Washington, DC, USA, 2011. IEEE Computer Society.

[23] He Huang, Liqiang Wang, Byung Chul Tak, Long Wang, and Chunqiang Tang. Cap3: A cloud auto-provisioning framework for parallel processing using on-demand and spot instances. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 228–235, June 2013.

[24] Bahman Javadi, Ruppa K. Thulasiram, and Rajkumar Buyya. Characterizing spot price dynamics in public cloud environments. *Fut. Gen. Comput. Syst.*, 29(4):988–999, June 2013.

[25] Daeyong Jung, SungHo Chin, KwangSik Chung, and HeonChang Yu. VM migration for fault tolerance in spot instance based cloud computing. In JamesJ.(JongHyuk) Park, H. R Arabnia, Cheonshik Kim, Weisong Shi, and Joon-Min Gil, editors, *Grid and Pervasive Computing*, volume 7861 of *LNCS*, pages 142–151. Springer, 2013.

[26] Bogumil Kaminski and Przemyslaw Szufel. On optimization of simulation execution on amazon {EC2} spot market. *Simulation Modelling Practice and Theory*, pages –, 2015.

[27] Ali Khajeh-Hosseini, David Greenwood, and Ian Sommerville. Cloud migration: A case study of migrating an enterprise it system to IaaS. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 450–457, Washington, DC, USA, 2010. IEEE Computer Society.

[28] Sunirmal Khatua and Nandini Mukherjee. Application-centric resource provisioning for amazon ec2 spot instances. In *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, pages 267–278, Berlin, Heidelberg, 2013. Springer-Verlag.

[29] Ioannis Konstantinou, Evangelos Floros, and Nectarios Koziris. Public vs private cloud usage costs: the stratuslab case. In *Proceedings of the 2nd International Workshop on Cloud Computing Platforms*, CloudCP '12, pages 3:1–3:6, New York, NY, USA, 2012. ACM.

[30] Yu-Kwong Kwok and Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, December 1999.

[31] P. Leitner, W. Hummer, B. Satzger, C. Inzinger, and S. Dustdar. Cost-efficient and application sla-aware client side request scheduling in an infrastructure-as-a-service cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 213–220, June 2012.

[32] L.M. Leslie, Young Choon Lee, Peng Lu, and A.Y. Zomaya. Exploiting performance and cost diversity in the cloud. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 107–114, June 2013.

[33] Wubin Li, P. Svard, J. Tordsson, and E. Elmroth. Cost-optimal cloud service placement under dynamic pricing schemes. In *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, pages 187–194, Dec 2013.

[34] Richard K. Lomotey and Ralph Deters. Csb-ucc: Cloud services brokerage for ubiquitous cloud computing. In *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*, MEDES '13, pages 100–107, New York, NY, USA, 2013. ACM.

[35] Sifei Lu, Xiaorong Li, Long Wang, H. Kasim, H. Palit, T. Hung, E.F.T. Legara, and G. Lee. A dynamic hybrid resource provisioning approach for running large-scale computational applications on cloud spot and on-demand instances. In *Parallel and Distributed Systems (ICPADS), 2013 International Conference on*, pages 657–662, Dec 2013.

[36] Jose Luis Lucas-Simarro, Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Scheduling strategies for optimal service deployment across multiple clouds. *Fut. Gen. Comput. Syst.*, 29(6):1431–1441, August 2013.

[37] Maciej Malawski, Kamil Figiela, and Jarek Nabrzyski. Cost minimization for computational applications on hybrid cloud infrastructures. *Future Gener. Comput. Syst.*, 29(7):1786–1794, September 2013.

[38] Aniruddha Marathe, Rachel Harris, David Lowenthal, Bronis R. de Supinski, Barry Rountree, and Martin Schulz. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '14, pages 279–290, New York, NY, USA, 2014. ACM.

[39] Peter Mell and Timothy Grance. The nist definition of cloud computing. Technical report, National Institute of Standards and Technology, Sept 2011.

[40] Ishai Menache, Ohad Shamir, and Navendu Jain. On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud. In *11th International Conference on Autonomic Computing (ICAC 14)*, pages 177–187, Philadelphia, PA, June 2014. USENIX Association.

[41] Ana-Maria Oprescu and Thilo Kielmann. Bag-of-tasks scheduling under budget constraints. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 351–359, Washington, DC, USA, 2010. IEEE Computer Society.

[42] B. Palanisamy, A. Singh, and B. Langston. Cura: A cost-optimized model for mapreduce in a cloud. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*, pages 1275–1286, May 2013.

[43] Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 2008.

[44] N. Sadashiv, S.M.D. Kumar, and R.S. Goudar. Hybrid spot instance based resource provisioning strategy in dynamic cloud environment. In *High Performance Computing and Applications (ICHPCA), 2014 International Conference on*, pages 1–6, Dec 2014.

[45] Prateek Sharma, Stephen Lee, Tian Guo, David Irwin, and Prashant Shenoy. Spotcheck: Designing a derivative iaas cloud on the spot market. In *Proceedings of the Tenth European Conference on Computer Systems*, EuroSys '15, pages 16:1–16:15, New York, NY, USA, 2015. ACM.

[46] Siqi Shen, Kefeng Deng, Alexandru Iosup, and Dick Epema. Scheduling jobs in the cloud using on-demand and reserved instances. In *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, pages 242–254, Berlin, Heidelberg, 2013. Springer-Verlag.

[47] ThamaraiSelvi Somasundaram, Kannan Govindarajan, Usha Kiruthika, and Rajkumar Buyya. Semantic-enabled care resource broker (secrb) for managing grid and cloud environment. *The Journal of Supercomputing*, 68(2):509–556, 2014.

[48] Kai Song, Yuan Yao, and L. Golubchik. Exploring the profit-reliability trade-off in amazon's spot instance market: A better pricing mechanism. In *Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on*, pages 1–10, June 2013.

[49] D. Stefani Marcon, R. Ruas Oliveira, M. Cardoso Neves, L. Salete Buriol, L.P. Gaspary, and M. Pilla Barcellos. Trust-based grouping for cloud datacenters: Improving security in shared infrastructures. In *IFIP Networking Conference, 2013*, pages 1–9, May 2013.

[50] Johan Tordsson, Rubén S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

[51] M. Unuvar, Y. Doganata, M. Steinder, A. Tantawi, and S. Tosi. A predictive method for identifying optimum cloud availability zones. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 72–79, June 2014.

[52] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.

[53] Christian Vecchiola, Rodrigo N. Calheiros, Dileban Karunamoorthy, and Rajkumar Buyya. Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka. *Fut. Gen. Comput. Syst.*, 28(1):58–65, January 2012.

[54] C. C A Vieira, L. F Bittencourt, and E. R M Madeira. Towards a PaaS architecture for resource allocation in IaaS providers considering different charging models. In *Economics of Grids, Clouds, Systems, and Services*, volume 8193 of *LNCS*, pages 185–196. 2013.

[55] C.C.A. Vieira, L.F. Bittencourt, and E.R.M. Madeira. Reducing costs in cloud application execution using redundancy-based scheduling. In *Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on*, pages 117–126, Dec 2014.

[56] C.C.A. Vieira, L.F. Bittencourt, and E.R.M. Madeira. A scheduling strategy based on redundancy of service requests on iaas providers. In *Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on*, pages 497–504, March 2015.

[57] C.C.A. Vieira, L.F. Bittencourt, and E.R.M. Madeira. A two-dimensional sla for services scheduling in multiple iaas cloud providers. *Int. J. Distrib. Syst. Technol.*, 6(4):45–64, October 2015.

[58] C.C.A. Vieira, L.F. Bittencourt, and E.R.M. Madeira. Um sla 3d para o escalonamento multiusuário sobre múltiplos provedores de iaas. In *de Redes de Computadores e Sistemas Distribuídos (SBRC), 2016 Simpósio Brasileiro*, page Aceito para publicação, Maio 2016.

[59] W. Voorsluys and R. Buyya. Reliable provisioning of spot instances for compute-intensive applications. In *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*, pages 542–549, March 2012.

[60] Wei Wang, Di Niu, Baochun Li, and Ben Liang. Dynamic cloud resource reservation via cloud brokerage. In *Distributed Computing Systems (ICDCS), 2013 IEEE 33rd International Conference on*, pages 400–409, July 2013.

[61] Alexander Wieder, Parmod Bhatotia, Ansley Post, and Rodrigo Rodrigues. Orchestrating the deployment of computations in the cloud with conductor. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 367–381, San Jose, CA, 2012. USENIX.

[62] Min Yao, Peng Zhang, Yin Li, Jie Hu, Chuang Lin, and Xiang Yang Li. Cutting your cloud computing cost for deadline-constrained batch jobs. In *Web Services (ICWS), 2014 IEEE International Conference on*, pages 337–344, June 2014.

[63] Sangho Yi, Artur Andrzejak, and Derrick Kondo. Monetary cost-aware checkpointing and migration on amazon cloud spot instances. *IEEE Trans. Serv. Comput.*, 5(4):512–524, January 2012.

[64] Zeng Zeng and Bharadwaj Veeravalli. Do more replicas of object data improve the performance of cloud data centers? In *Proceedings of the 2012 IEEE/ACM Fifth*

International Conference on Utility and Cloud Computing, UCC '12, pages 39–46, Washington, DC, USA, 2012. IEEE Computer Society.

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

[66] Qi Zhang, Eren Gürses, Raouf Boutaba, and Jin Xiao. Dynamic resource allocation for spot markets in clouds. In *Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services*, Hot-ICE'11, Berkeley, CA, USA, 2011. USENIX Association.