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

Uma Arquitetura Resiliente Baseada em Agentes para Instâncias Transientes na Computação em Nuvem

José Pergentino de Araújo Neto

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Orientadora
Profa. Dra. Célia Ghedini Ralha

Brasília
2019

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

Uma Arquitetura Resiliente Baseada em Agentes para Instâncias Transientes na Computação em Nuvem

José Pergentino de Araújo Neto

Tese apresentada como requisito parcial para
conclusão do Doutorado em Informática

Profa. Dra. Célia Ghedini Ralha (Orientadora)
Universidade de Brasília (UnB)

Profa. Dra. Lúcia Maria de A. Drummond
Universidade Federal Fluminense (UFF)

Profa. Dra. Genaína Nunes Rodrigues
Universidade de Brasília (UnB)

Profa. Dra. Aletéia Patrícia Favacho de Araújo
Universidade de Brasília (UnB)

Profa. Dra. Genaína Nunes Rodrigues
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 12 de dezembro de 2019

Dedicatória

Dedico este trabalho a minha família, principalmente a minha esposa Catarina Gama Catão, pelo apoio, paciência e parceria, e a nossa filha Lis Catão Araújo, que na sua inocência, recarrega minhas energias através da sua alegria e carinho. Vocês representam tudo na minha vida. Amo vocês.

Agradecimentos

A Profa. Dra. Célia Ghedini Ralha, por ter acreditado e me apoiado incansavelmente nesta pesquisa. Sou e serei eternamente grato pela sua excelente dedicação, parceria e paciência em todos os momentos durante estes anos. Pela convivência e amizade que, ao longo destes anos, muito me ensinou. Foi um período de grande aprendizado pessoal, profissional e acadêmico. Muito obrigado por ter aceitado este desafio e ser quem você é. Não teria dado certo se fosse com outra pessoa.

Aos meus familiares, principalmente aos meus pais, que de forma espiritual estão me encorajando para lutar pelos meus objetivos. Agradeço a Deus pela família que tenho e pela educação que recebi dos meus pais, farei o possível para ser o que eles foram para mim. Aos meus amigos, que mesmo distante, lutam pelo meu sucesso. A todos os colegas que fiz durante esta minha passagem. Obrigado pelos momentos incansáveis de estudo, companhia, colaboração e solidariedade.

Aos membros da banca presentes na minha qualificação, Prof. Dr. Anderson Clayton Alves Nascimento, Profa. Dra. Aletéia Patrícia Favacho de Araujo, Profa. Dra. Alba Cristina Magalhães Alves de Melo, Prof. Dr. Donald Pianto, que me orientaram com maestria no direcionamento da minha pesquisa.

À Universidade de Brasília, por acolher e apoiar em vários aspectos que permitisse o desenvolvimento deste trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), por meio do Acesso ao Portal de Periódicos.

Resumo

Os provedores de nuvem computacional estão explorando seus recursos ociosos e oferecendo-os em forma de serviços não confiáveis. Conhecidos como instâncias transientes, estes serviços são sublocados a preços consideravelmente inferiores aos *on-demand* e podem ser revogados sem a intervenção do usuário. A Amazon AWS oferece este serviço, chamando de Instâncias Spot, por meio de um modelo de leilão, tendo o seu preço definido de acordo com a lei de oferta e procura. O usuário adquire uma Instância Spot enquanto o seu lance for superior ou igual ao valor da instância no mercado, sendo revogado caso contrário. Este serviço vem sendo utilizado para a execução de aplicações que necessitam de um consumo excessivo de CPU e memória. No entanto, para explorar de maneira eficiente estas instâncias, as aplicações precisam implementar técnicas de tolerância a falhas para evitar perda de dados. Neste cenário, esta pesquisa aborda o problema que envolve a utilização de instâncias transientes em um ambiente confiável, que garanta a execução da aplicação no menor tempo possível. Para uso adequado das instâncias transientes, em uma abordagem multi-estratégica, esta pesquisa apresenta uma arquitetura baseada em agentes inteligentes para prover um ambiente resiliente e tolerante a falhas, denominada BRA2Cloud (*A Brand new Agent Based Architecture for Cloud Computing*). BRA2Cloud combina a utilização de raciocínio baseado em casos com um modelo estatístico na predição do tempo de sobrevivência em instâncias transientes, refinando parâmetros de tolerância a falhas para reduzir o tempo total de execução. Experimentos demonstram que o modelo proposto é capaz de apresentar uma predição na garantia de tempo de revogação com níveis de acurácia de até 91%. A avaliação da proposta utilizou aproximadamente 21 milhões de registros de mudanças de preços reais, coletados a partir das Instâncias Spot, possibilitando a criação de mais de 357 milhões de registros na base de casos. Como resultado, a abordagem utilizada obteve uma redução no custo e uma diminuição no tempo total de execução de até 70,12% quando comparada a outras abordagens na literatura.

Palavras-chave: BRA2Cloud, Agentes Inteligentes, Máquinas Virtuais, Multi-estratégico, Tolerância a Falhas, Raciocínio Baseado em Casos

Abstract

Unused resources are being exploited by cloud computing providers, which are offering them as unreliable services. Known as transient instances, these services are sublet with low costs compared to on-demand, but without availability guarantee. Spot instances are transient instances offered by Amazon AWS, with rules that define prices according to supply and demand in a market model. These instances will run for as long as the current price is lower than the maximum bid price given by users. Spot instances have been increasingly used for executing computation and memory-intensive applications. By using dynamic fault-tolerant mechanisms and appropriate strategies, users can effectively use spot instances to run applications at a cheaper price and avoid losing processed data. In this scenario, this research addresses the problem involving the use of transient instances in a trusted environment that ensures application execution in the shortest possible time. This research presents a resilient agent-based architecture for transient instances in cloud computing, namely BRA2Cloud (*A Brand new Agent Based Architecture for Cloud Computing*). For appropriate usage of transient instances, using an multi-strategic approach, the architecture uses intelligent agents that combines machine learning and a statistical model to predict instance survival time, refine fault tolerance parameters and reduce total execution time. Experiments show that the proposed model is capable of presenting a prediction on revocation time guarantee with levels of accuracy up to 91%. The proposal evaluation used approximately 21 million records of actual price changes collected from Spot Instances, enabling the creation of over 357 million records in the case base. As a result, the approach used achieved a reduction in cost and a reduction in total execution time of up to 70.12% compared to other approaches in the literature.

Keywords: BRA2Cloud, Inteligent Agents, Virtual Machines, Multi-strategic, Fault Tolerance, Case Based Reasoning

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A computação em nuvem surgiu como uma proposta de ambiente computacional com alto nível de escalabilidade, flexibilidade e com preço accessível para atender as necessidades de computação emergente (Armbrust et al., 2010a; Mell et al., 2011). As infraestruturas computacionais na nuvem vem apresentando uma ótima alternativa para execução de aplicações com carga elevada em ambientes virtuais seguros e escaláveis (Khan et al., 2012).

Neste sentido, os provedores de computação em nuvem necessitam utilizar mecanismos que possam dinamicamente planejar e provisionar recursos para usuários que necessitam de ambientes robustos na execução de aplicações.

A computação em nuvem é um ambiente que oferece recursos de maneira totalmente distribuída, disponibilizados sob demanda por meio da Internet (Foster et al., 2008). Este ambiente provê software, serviço, meio de armazenamento e processamento em plataformas gerenciáveis, abstratas, virtualizadas e dinamicamente escaláveis.

Em Buyya et al. (2009), os autores mostram que estes serviços podem ser disponibilizados por meio de um ambiente paralelo, no qual deve existir um conjunto de recursos computacionais conectados, dinamicamente provisionados e disponibilizados de maneira transparente e unificada para o usuário. Além disso, a disponibilização de recursos deve respeitar as definições acordadas no nível de serviço estabelecido entre o consumidor e o provedor de serviço (*Service Level Agreement* - SLA).

As definições de computação em nuvem apresentadas tem em comum a ideia de uma plataforma que disponibilize recursos e serviços por meio de ambientes seguros, transparentes e escaláveis em um modelo *pay-per-use*. Assim, pode-se citar como provedores que se destacam no oferecimento de coleções de serviços de computação em nuvem: Amazon, Google e Microsoft, por meio dos respectivas provedores de serviços, *Amazon Web Services* (Amazon, 2019), *Google Cloud Platform* (Google, 2019), e *Microsoft Azure* (Microsoft, 2019).

Os provedores de computação em nuvem, cientes de que seus recursos de infraestrutura não estavam sendo explorados na sua potencialidade, começaram a adotar um modelo de negócio que oferece esses recursos ociosos por meio de Máquinas Virtuais (VMs) não confiáveis, i.e. sem garantia de disponibilidade. Conhecidas como Instâncias Transientes Shastri et al. (2016), estas VMs são revogadas do usuário de forma irreversível de acordo com regras específicas de cada provedor, sendo oferecidas por valores consideravelmente inferiores quando comparadas aos recursos com disponibilidade garantida.

No cenário atual, pode-se citar os quatro maiores provedores que oferecem VMs como instâncias transientes, adotando regras de negócio específicas: (1) IBM Cloud[1] - explora os recursos ociosos provisionando-os em VMs na presença de ociosidade em seus servidores físicos; (2) Google GCE[2] - explora os recursos ociosos oferecendo instâncias por um preço fixo e pré-estabelecido, porém com limitações de tempo; (3) Microsoft Azure[3] - estende o serviço oferecendo uma plataforma para submissão de tarefas; e (4) AWS EC2[4] - adota um modelo de mercado que se baseis em lances do usuário em um cenário de precificação dinâmico.

Em Iosup et al. (2011), os autores mostram que ambientes de computação em nuvem são vistos como um promissor instrumento para execução de aplicações distribuídas do tipo *bag-of-tasks* (BoT). Em Cirne et al. (2003), os autores declaram que BoT é formado por um conjunto de aplicações (tarefas) independentes, que podem ser executadas em um ambiente de execução paralela e distribuída.

Aplicações BoT vem sendo utilizadas em uma variedade de cenários, incluindo simulações, processamento de imagens e bioinformática. Apesar do *grid* computacional ser o ambiente predominante para execução de aplicações BoT, devido a sua alta disponibilidade e provimento de recursos computacionais, a computação em nuvem tornou-se um ambiente alternativo e interessante para ser explorado.

Além dos benefícios da computação em nuvem, o uso de instâncias transientes apresentam desafios relevantes para a computação científica, envolvendo aspectos relacionados a segurança, conectividade, confiabilidade e resiliência, sendo esses adequados à aplicação de técnicas de Tolerância a Falhas (TF) (Iosup et al., 2008; Oprescu and Kielmann, 2010; Tang et al., 2016; Yang et al., 2015).

[1] *Transient VM* - `https://cloud.ibm.com/docs/vsi?topic=virtual-servers-about-vs-transient`
[2] *Preemptible VM* - `https://cloud.google.com/preemptible-vms`
[3] *Low-priority VM* – `https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms`
[4] *Spot Instances* - `https://aws.amazon.com/ec2/spot`

1.1 Motivação

Na literatura há diversos trabalhos que apresentam soluções para otimizar o custo do usuário, adaptando aplicações por meio de mecanismos que permitam uma melhor estratégia de alocação e de utilização de recursos (Buyya et al., 2016; Gong et al., 2015; Guo et al., 2015; He et al., 2015; Irwin et al., 2017; Qu et al., 2016).

Especificamente, as instâncias transientes da Amazon AWS, conhecidas como Instância Spot, são servidores disponíveis a partir de um modelo de determinação de preços de mercado. Nesse modelo, os recursos ociosos são sublocados e disponibilizados utilizando-se um cenário com preço variável dinamicamente de acordo com a lei da oferta e procura.

A Figura 1.1 ilustra um histórico real de preços de uma instância na AWS no período de Dezembro de 2018 a Março de 2019. Nessa figura é possível perceber a existência de uma variabilidade nos preços no decorrer do tempo, tendenciado por um ambiente de leilão, o qual é por definição imprevisível.

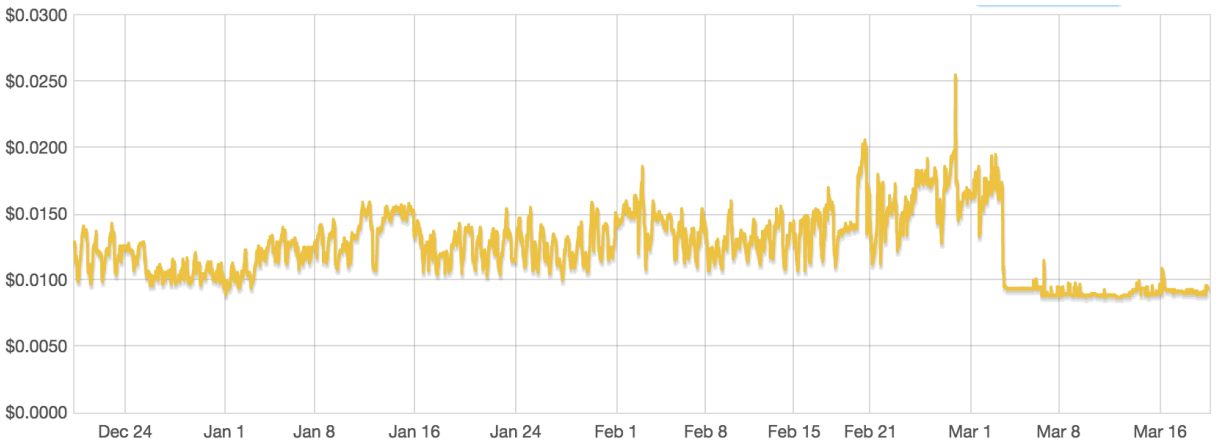


Figura 1.1: Histórico de preço de uma instância Spot do tipo *m3.medium* no período entre 20 de dezembro de 2018 e 20 de março de 2019.

Nesse cenário, a busca por garantias de execuções mais confiáveis é natural. Neste caso, as aplicações devem possuir a capacidade de prevenção e de adaptação em ambientes onde eventos imprevisíveis acontecem (e.g., falhas de revogação), sendo desejável o uso de técnicas de TF.

No entanto, a decisão dinâmica de estratégias de TF em ambientes de computação em nuvem não é um problema com solução trivial, pois deve levar em consideração diversos fatores, tais como: as características da aplicação, sua temporalidade, recursos utilizados e o histórico de execução da instância, além dos aspectos relacionados à probabilidade de falha da aplicação. Neste sentido, torna-se necessário um modelo que possibilite a tomada de decisão racional que escolha dinamicamente a estratégia de utilização de instâncias transientes com redução no tempo total de execução e no custo do usuário.

Desta forma, será possível reduzir a possibilidade de falhas nas revogações destes servidores e diminuir o tempo de execução comparado com as abordagens existentes na literatura.

1.2 Problema

A execução de tarefas BoT com *workload* elevado consomem tempo e recursos consideráveis, estando sujeitas a interrupções devido a falhas não esperadas, colocando em risco os dados processados. Na utilização de instâncias transientes, apesar do usuário estar ciente de que não há garantia de disponibilidade deste serviço, não há como prever o momento em que a revogação ocorrerá.

Em Irwin et al. (2017), os autores deixam claro que para utilizar de maneira eficiente as instâncias transientes em computação em nuvem, faz-se necessário a presença de três fatores chave:

1. Escolher o servidor mais adequado – as instâncias transientes são escolhidas sob medida para a execução da aplicação, evitando-se desperdícios de recurso e custo excessivo;

2. Escolher o mecanismo de TF apropriado – envolve a escolha da técnica adequada que atenda os requisitos da aplicação, evitando a perda de processamento com garantia de execução, reduzindo o tempo e os respectivos custos; e

3. Recuperar a falha – na presença de falhas de revogação faz-se necessário mecanismos de recuperação que garantam a continuidade da execução. Neste caso, a técnica de TF deve possibilitar a recuperação de um estado consistente de execução, além de garantir a sua finalização.

Considerando os trabalhos que exploram o uso de instâncias transientes encontrados na literatura, o problema abordado por esta pesquisa está no uso de agentes para a utilização dessas instâncias, com objetivo de escolher a técnica de TF mais apropriada para execução de aplicações, provendo um ambiente resiliente de computação em nuvem.

Durante o desenvolvimento da pesquisa definiu-se como questão de investigação central: Quais são as vantagens de utilizar uma arquitetura tolerante a falhas, baseada em agentes autônomos, para manter a resiliência e reduzir os custos do usuário em um ambiente de computação em nuvem com instâncias transientes?

1.3 Objetivos

O objetivo principal deste trabalho envolve o desenvolvimento de uma arquitetura baseada em agentes autônomos, para escolha dinâmica da técnica mais adequada de TF na execução de BoT em computação em nuvem com instâncias transientes, minimizando os impactos na presença de falhas de revogação e reduzindo os custos do usuário.

Assim sendo, apresenta-se como objetivos específicos deste trabalho:

(i) Avaliar a aplicabilidade dinâmica das técnicas de TF, e analisar as suas influências em diferentes cenários, incluindo ambiente simulado e real;

(ii) Desenvolver uma arquitetura transparente que não necessite de adaptação das tarefas; e

(iii) Validar o modelo proposto por meio de um estudo de caso, utilizando uma aplicação BoT existente para efeitos de comparação da proposta.

1.4 Metodologia

Para atingir os objetivos deste trabalho de pesquisa, algumas etapas metodológicas foram adotadas, conforme seguem:

- Estudar os principais conceitos de TF, computação em nuvem, instâncias transientes e arquiteturas baseadas em agentes inteligentes;
- Realizar o levantamento do estado da arte, por meio de uma revisão sistemática da literatura, com a finalidade de obter resultados dos trabalhos relacionados e situar a pesquisa no cenário mundial;
- Investigar a aplicabilidade de técnicas de TF em instâncias transientes na computação em nuvem;
- Definir, implementar e validar o modelo arquitetural proposto neste trabalho;
- Analisar, de maneira quantitativa e qualitativa, os resultados apresentados após o desenvolvimento do modelo em ambiente simulado e real;
- Publicar os resultados alcançados em eventos científicos e periódicos da área de sistemas distribuídos e sistemas baseados em agentes; e
- Escrever a tese, relatando o trabalho de pesquisa realizado, incluindo as dificuldades encontradas e os resultados alcançados.

Os demais capítulos desta tese estão organizados da seguinte forma: no Capítulo 2 apresenta-se a fundamentação teórica incluindo os principais conceitos envolvidos na pesquisa; no Capítulo 3 descreve-se os trabalhos correlatos, apresentando uma revisão do estado da arte com as principais pesquisas que envolvem o uso de computação em nuvem com instâncias transientes, as técnicas de TF adotadas e a utilização de agentes para aplicação dessas técnicas; a solução proposta é apresentada no Capítulo 4, incluindo a descrição do modelo conceitual e os elementos arquiteturais da proposta; os experimentos realizados em ambiente simulado e real são apresentados no Capítulo 5; por fim, no Capítulo 6 são apresentadas as conclusões e as considerações relacionadas aos trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste Capítulo será apresentada uma visão geral dos principais conceitos envolvidos na pesquisa, incluindo elementos relacionados a computação em nuvem, técnicas de TF, inteligência artificial e análise de sobrevivência.

2.1 Computação em Nuvem

A computação em nuvem vem alterando o paradigma na execução e no provimento de serviços, mudando o escopo e o ambiente de execução tradicional em máquina local para um ambiente distribuído. A sua disponibilização é feita sob demanda de serviços de forma transparente, ubíqua, em ambientes computacionais seguros, compartilháveis e escaláveis, provisionados de forma rápida, muitas vezes em uma localização física desconhecida (Mell et al., 2011).

Com o objetivo de prover serviços e recursos de uma maneira fácil, eficiente, transparente, flexível e escalável, é composta por uma arquitetura de computação distribuída que permite a disponibilização de recursos, tais como processamento, memória, armazenamento e software (Foster et al., 2008). Originalmente, esta arquitetura que utiliza recursos de forma distribuída, deve ser oferecida para os usuários por meio da Internet.

Esta arquitetura é conhecida como computação em nuvem, a qual oferece diversos tipos de serviços e recursos gerenciados em plataformas distintas, sendo organizada por regiões e zonas. Na literatura encontrou-se várias referências que definem a computação em nuvem:

- Em Foster et al. (2008), os autores definem como um ambiente cujos recursos estão disponíveis de maneira distribuída, disponibilizado sob demanda para usuários por meio da Internet, provendo serviços, meios de armazenamento e processamento em plataformas gerenciáveis, abstratas, virtualizadas e dinamicamente escaláveis;

- Em Buyya et al. (2009), os autores complementam a definição de computação em nuvem como sendo um tipo de sistema que além de distribuído, também é paralelo em um conjunto de recursos computacionais conectados, dinamicamente provisionados e disponibilizados de maneira transparente e unificada para o usuário, respeitando as definições acordadas entre o consumidor e o provedor de serviços no SLA;

- Os autores em Fox et al. (2009) e Armbrust et al. (2010a) caracterizam como computação em nuvem o ambiente disponível na Internet que possibilita o acesso a um conjunto de aplicações que, unidas, oferecem serviços de software e hardware de maneira descomplicada e transparente para o usuário.

A ideia fundamental nas definições apresentadas considera a existência do modelo *pay-per-use* para oferecer benefícios tanto para os consumidores como para os provedores de serviços na nuvem. Desta forma, os consumidores não precisam contratar profissionais para desenvolver soluções ou investir em equipamentos de hardware e recursos de infraestrutura, já que poderiam utilizar soluções de serviços e recursos disponíveis na computação em nuvem.

A Figura 2.1 apresenta uma representação do acesso a provedores de computação em nuvem e seus pacotes de serviços.

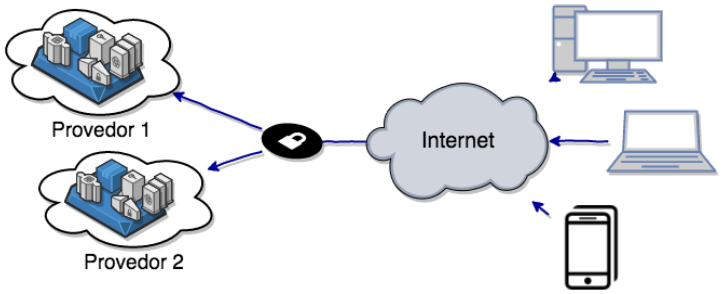


Figura 2.1: Arquitetura de provedores de computação em nuvem.

Como resultado, os consumidores alcançam uma redução de custos baseado em suas necessidades de recurso e tempo. Além disso, o termo sob demanda (*on-demand*) permite que os consumidores tenham acesso a uma variedade de recursos disponíveis, podendo contratá-los de maneira confiável, segura, e com garantida de disponibilidade, para que possam adaptar às suas necessidades.

Neste cenário, a computação em nuvem apresenta-se como um ambiente atraente para serviços específicos com garantia de disponibilidade, sem a necessidade de adquirir equipamentos de alto custo, que poderiam ficar com uma parcela de recursos ociosa. Então, a computação em nuvem reserva recursos para a necessidade do usuário, que utiliza e paga apenas pelo que consome (Buyya et al., 2009).

Arquitetura de Computação em Nuvem

Os serviços de computação em nuvem são categorizados em três classes, de acordo com o nível de abstração, capacidade e modelo de provisionamento de serviço. Os autores em Youseff et al. (2008a) descrevem esses três níveis de abstração como camadas arquiteturais, em que os serviços de uma camada superior pode ser composta pelo agrupamento de vários serviços das camadas inferiores, como apresentado na Figura 2.2.

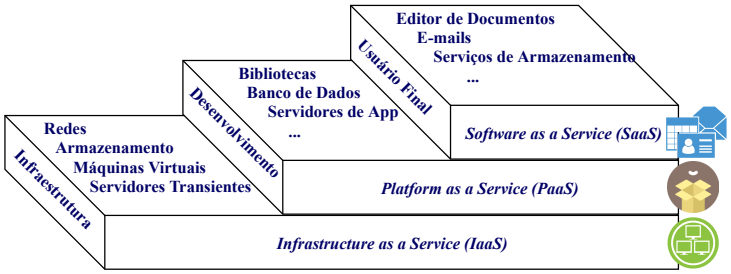


Figura 2.2: Classes e camadas da computação em nuvem.

Em Lenk et al. (2009), os autores apresentam as três classes presentes em um ambiente de computação em nuvem:

1. Software como Serviço (*Software as a Service - SaaS*) – serviços em formato de software são disponibilizados para usuários finais e podem ser acessados por meio dos navegadores de Internet. Desta forma, a necessidade de acesso à Internet está presente para uso dos recursos do software. Aplicações *desktop* tradicionais, tais como processadores de texto e imagens, podem ser acessadas como serviços disponíveis na nuvem, sem a necessidade de ter o software instalado na máquina do usuário (Youseff et al., 2008b).

2. Plataforma como Serviço (*Platform as a Service - PaaS*) – caracterizada pela disponibilização de um ambiente que provê interfaces públicas, permitindo ao usuário o acesso para programar, testar e executar aplicações de forma transparente, utilizando processadores e memória sob demanda, conforme políticas previstas pelo provedor. Assim sendo, autenticação, processamento de imagens, e aprendizado de máquinas são exemplos de serviços providos, estando disponíveis para utilização sem a necessidade de uma infraestrutura no lado do cliente.

3. Infraestrutura como Serviço (*Infrastructure as a Service - IaaS*) – funciona como uma camada base, como uma classe de serviço que oferece acesso a recursos físicos de maneira virtualizada, tais como armazenamento, processamento e comunicação. Esta classe possibilita, por exemplo, o provisionamento de servidores, viabilizando a escolha de diversos sistemas operacionais e permitindo personalização de software

e serviço que estarão executando nas VMs. Os autores em Nurmi et al. (2009); Sotomayor et al. (2009) reforçam que IaaS é a classe que está presente na camada base em sistemas de computação na nuvem, sendo necessária para o funcionamento das camadas superiores.

Conforme apresentada na Figura 2.2, a arquitetura de nuvem está categorizada em três camadas: SaaS - a que mais se aproxima do usuário devido a natureza dos seus serviços; PaaS - a mais utilizada por desenvolvedores que buscam reutilizar serviços já implementados; e IaaS - serve como apoio para todas as demais camadas, permitindo que serviços possam ser executados em uma plataforma robusta, segura e escalável.

Na camada IaaS estão localizados os recursos físicos de infraestrutura que contém *hardwares* com seus servidores e datacenters, disponibilizando-os de maneira virtualizada para o cliente como uma alternativa a execução de aplicações em recursos locais (Vaquero et al., 2008). Provedores que se destacam ao oferecer serviços nesta classe de nuvem é o Google App Engine e Amazon AWS, por meio dos seus serviços de *Google Compute Engine* (GCE)[1] e *Elastic Compute Cloud* (EC2)[2].

Elementos que diferem o ambiente de computação em nuvem dos demais são apresentados em (Buyya et al., 2009; Fox et al., 2009; Mell et al., 2011). Os autores destacam como propriedades essenciais que caracterizam o modelo de computação em nuvem:

1. *Self-service* – os consumidores dos serviços oferecidos pelos provedores de computação em nuvem esperam acessos aos recursos instantâneos e sob demanda de maneira descomplicada. Para atender esta expectativa, provedores de computação em nuvem devem prover meios de acessos para que os clientes possam customizar o ambiente de acordo com suas necessidades e pagar apenas pelos recursos utilizados sem que haja intervenção humana no processo.

2. Medição e cobrança de uso – os usuários podem realizar requisições e utilizar os recursos que são necessários para atender às suas necessidades, sem compromisso inicial. Os serviços devem ser mensurados por meio de uma medida básica e pequena, e.g., a cobrança por intervalos de hora na utilização de recurso. Desta forma, usuários podem adquirir recursos, utilizá-los e liberá-los de acordo com as suas necessidades, pagando apenas pelos recursos consumidos. O acesso a esses recursos devem ser mensurados e os provedores desses serviços devem ter meios eficientes de medição para diferentes naturezas de recurso, e.g., serviços de armazenamento, processamento e utilização de rede. Esses serviços possuem unidades de medição

[1]Disponível em `https://cloud.google.com/compute`
[2]Disponível em `https://aws.amazon.com/ec2`

diferente: o armazenamento deve ser mensurado pelo espaço utilizado; o processamento é medido pelo tempo de propriedade do(s) processador(es); e a medição de utilização de rede é definido pelo tráfego realizado na interface de rede. Por este motivo, provedores de nuvem devem prover meios para negociação de contratos que definam preços e faturamento, permitindo que o usuário possa ter acesso a relatórios de maneira transparente para observar dados sobre a utilização dos recursos disponíveis na nuvem.

3. Elasticidade – a transparência na utilização dos recursos cria uma ilusão de que os recursos computacionais disponibilizados pelos provedores de computação em nuvem são infinitos. Portanto, usuários esperam que estes recursos possam ser fornecidos em qualquer quantidade e em qualquer momento, possibilitando aumentar a utilização destes recursos de maneira dinâmica e provisionada, caracterizando a elasticidade. Desta forma, a aplicação que necessita de mais recursos pode automaticamente utilizar e liberar esses recursos, aumentar ou diminuir a sua escala.

4. Customização – em um ambiente que possui inúmeros recursos disponíveis, deve-se permitir a personalização dos recursos requeridos pelo usuário. Em se tratando de serviços de infraestrutura, que disponibiliza o acesso as VMs com sistemas operacionais pré-configurados, esta personalização trata da possibilidade do usuário modificar o acesso, privilégios e serviços do sistema operacional vigente naquela máquina virtual. Isto só é permitido se o perfil de acesso possuir o privilégio de administrador.

5. Amplo acesso – a capacidade de acesso aos recursos da nuvem devem estar presentes nos provedores de computação em nuvem. Os recursos devem ser disponibilizados em um meio de comunicação na Internet e os mecanismos de acesso devem atender a padrões que promovem a heterogeneidade de plataformas.

6. Agrupamento de recursos – os recursos disponíveis em um provedor de nuvem são organizados em forma de *pooling* de recursos, que organizam de maneira agrupada e controlam a disponibilização de seu acesso, podendo ser físico ou virtual. O objetivo deste agrupamento é facilitar o acesso concorrente aos recursos por vários usuários e facilitar os ajustes necessários.

Modelos de Computação em Nuvem

A computação em nuvem surgiu com o propósito de permitir a utilização de seus serviços, disponibilizando-os de maneira pública com o objetivo de estreitar a relação com os seus

usuários na implantação de seus serviços. Com o passar do tempo, outros modelos de implantação foram adotados, oferecendo variações de localização física e distribuída.

Em Mell et al. (2011), os autores apresentam modelos de nuvem que independem da classe de serviço, sendo classificados em: pública, privada, comunitária ou híbrida. Os autores em Armbrust et al. (2010b) propõem a definição de nuvem pública como sendo aquela que está disponível para todos, sem restrições de acesso, desde que a sua utilização seja cobrada de acordo com os recursos e serviços utilizados.

Inserida em um contexto empresarial e inacessível externamente, o modelo de nuvem privada disponibiliza recursos e serviços a usuários restritos, podendo ser os funcionários da empresa provedora ou parceiros que compartilham recursos. O acesso por meio da Internet é restrito, o que reforça a vantagem de segurança e políticas de acesso aos seus recursos. Em Fox et al. (2009), os autores definem o modelo de nuvem privada como sendo um datacenter interno e restrito de uma organização que não está disponível para o público externo, permitindo um controle maior no gerenciamento dos recursos.

Em Voorsluys and Buyya (2012), os autores complementam que na maioria dos casos, uma nuvem privada significa a realização de uma reestruturação da infraestrutura existente, adicionando virtualização e interfaces de acesso aos seus serviços. Desta forma, o usuário interno pode interagir com os datacenters locais, apresentando as mesmas vantagens das nuvens públicas, principalmente pelo seu acesso privilegiado aos servidores virtuais.

Uma nuvem comunitária é aquela que compartilha sua infraestrutura por organizações que possuem interesses comuns, podendo ser utilizada e gerenciada por uma ou várias organizações de maneira transparente. Em Mell et al. (2011), os autores declaram estes interesses comuns como sendo requisitos de segurança, políticas de acesso e jurisdição.

Por fim, o modelo de nuvem híbrida é caracterizado por meio de um ambiente que envolve mais de um dos modelos citados. Uma nuvem híbrida se forma quando uma nuvem privada é complementada com utilização de outros recursos, normalmente públicos, tornado-a com capacidade de uma nuvem pública (Sotomayor et al., 2009). Quando uma nuvem privada está sobrecarregada, ocorre uma alocação semelhante a um empréstimo, mesmo que temporária, de recursos externos para aumentar a capacidade e suportar picos de execução. Em Mell et al. (2011), os autores definem esta sobrecarga em nuvem privada e alocação temporária de recursos de nuvens públicas como explosão de nuvem (*cloud-bursting*).

Recursos Transientes

Computação em nuvem tem uma crescente demanda, forçando provedores a desenvolver modelos variados de disponibilização de serviços e recursos, principalmente na classe de

IaaS, o qual teve um aumento estimado de 30.6% em 2016 (Columbus, 2016). Como principais provedores destes serviços, que vêm investindo massivamente seus recursos de infraestrutura tem-se: Amazon AWS *Elastic Compute Cloud* (EC2), *Google Compute Engine* (GCE) e Microsoft Azure.

Com o crescimento na procura dos serviços de IaaS, cresceu também a quantidade de VMs alocadas pelos usuários na categoria *on-demand*, possuindo garantia de alta disponibilidade pelos seus recursos. Porém, devido a natureza da sua utilização, aplicações que estariam executando nestas VMs podem não utilizar todos os recursos alocados, permitindo a criação de um modelo de negócio para disponibilizar, de forma transparente, os recursos ociosos para que sejam usados por outros usuários por preços mais acessíveis.

A utilização desses recursos pode variar de acordo com o tempo e, para o provedor de computação em nuvem, a imprevisibilidade na utilização destes recursos não permite a garantia na realocação para outros usuários, porém, permite a utilização sem garantia de disponibilidade provida na categoria *on-demand*. Em Shastri et al. (2016), os autores apontam que a quantidade de recursos e a capacidade de ociosidade provida pelas alocações *on-demand* pode ser significante.

Nesse contexto, os provedores de computação em nuvem adotaram um modelo de negócio que oferece a capacidade ociosa dos seus recursos em forma de instâncias transientes (Shastri et al., 2016). Esta modalidade oferece a possibilidade de alocação de recursos semelhantes a servidores *on-demand* por um preço que pode chegar até 10% do seu valor original, porém sem a garantia de disponibilidade, revogando o acesso à VM.

Instâncias transientes possibilitam que usuários possam reduzir os custos para execução de aplicações que possuem características de uso excessivo de CPU e memória, em grande escala, desde que estas aplicações estejam preparadas para cenários em que haja uma degradação de performance e perda do servidor de execução.

Provedores de computação em nuvem utilizam recursos não utilizados sob a ótica de que, quando necessário, o servidor transiente pode ser revogado de acordo com as regras estabelecidas por cada provedor. A Google GCP e Microsoft Azure funcionam como uma caixa preta, que aparentemente revogam o acesso quando o recurso ocioso começa a ser utilizado pelo usuário proprietário, mas nada é publicado sobre os seus funcionamentos. Em contrapartida, a Amazon AWS adota um modelo de mercado baseado em leilão, disponibilizando publicamente os registros das variações em seus valores dos últimos três meses.

Em Shastri et al. (2016), os autores apresentam as instâncias transientes como um conceito relativamente novo, e por esse motivo, não há um padrão elaborado para definição de preços praticados e termos de uso. Provedores que se destacam no mercado de máquinas transientes diferem na definição de seus modelos. A Amazon AWS disponibiliza os serviços

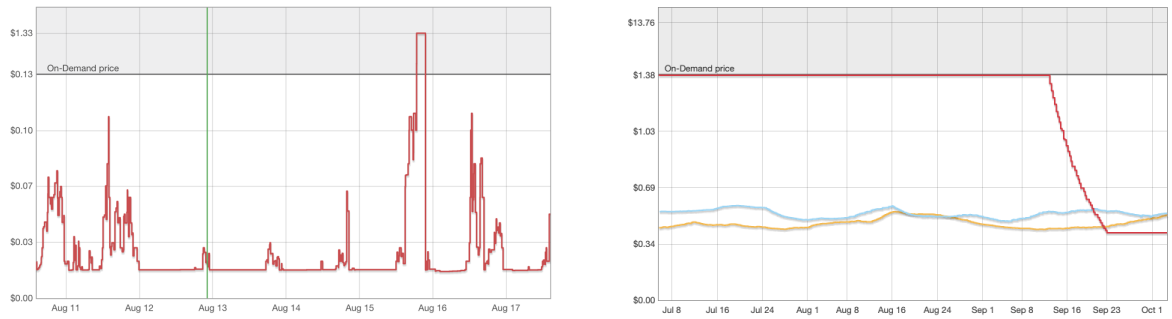
de instâncias transientes por meio de suas instâncias spots em formas de VMs, que possui um mecanismo de mercado para medição de cobrança dos seus recursos.

Neste modelo de mercado, semelhante a um leilão, os usuários informam qual é o valor máximo (um lance) que está disposto a pagar, e adquire uma instância spot enquanto seu lance for superior ou igual ao valor corrente da instância, de acordo com a sua disponibilidade em uma unidade de tempo pré-estabelecida.

A Amazon AWS define valores por estas unidades baseado em um modelo de mercado. No entanto, em Ben-Yehuda et al. (2011) foram feitas análises no histórico de preços definidos pela Amazon para cobrança de instâncias transientes, e os autores apresentam resultados que apontam que os valores definidos neste leilão não eram orientados pelo mercado. Posteriormente, o modelo foi modificado pela Amazon para adequação em um modelo que represente um cenário real de mercado, que varia os seus preços baseado na lei da oferta e procura.

Pela natureza de funcionamento de um leilão com esquema de mercado, o valor da instância é imprevisível, sendo que o seu tratamento é dado da seguinte maneira: o usuário define um lance com o valor máximo que está disposto a pagar e adquire a instância spot se o valor de leilão for inferior ou igual ao seu lance.

Quando o seu lance é superior ao valor corrente da instância transiente, a Amazon disponibiliza ou mantém a propriedade do servidor transiente previamente disponibilizado. A Figura 2.3 apresenta um exemplo de histórico de preço em instâncias spots, recuperadas entre julho e agosto de 2019.



(a) Instância c5.9xlarge. (b) Instância r3.8xlarge.

Figura 2.3: Exemplos de históricos nos preços de instâncias spots.

Observa-se que diferentes tipos de instâncias possuem comportamentos diferentes na variação do seu preço. Se o valor corrente da instância sobe a ponto de ultrapassar o valor estipulado no lance do usuário, a VM é revogada de maneira irreversível, ou seja, sem a intervenção do usuário para evitar a revogação.

Quando há a revogação, o usuário possui o período de dois minutos para que possa agir e evitar a perda de dados processados. Caso o usuário queira usar o recurso novamente, deverá submeter um novo valor para o seu lance e aguardar a sua disponibilização.

Diferentemente do modelo econômico praticado pela AWS, o GCE, IBM Cloud e Azure definem um valor fixo para as suas instâncias transientes, fixando em uma cobrança de aproximadamente 20% do valor comparado a uma VM *on-demand*.

Estas instâncias transientes, chamados de instâncias preemptáveis (GCE), servidor virtual transiente (IBM), e VMs de baixa prioridade (Azure), também podem ser revogadas do usuário de forma imprevisível, respeitando a regra de que o máximo de tempo de propriedade sobre estas instâncias não pode ultrapassar 24 horas. Para instâncias mais robustas o limite de tempo é de seis horas. Quando há revogação, o GCE e Azure disponibilizam 30 segundos para que o usuário possa agir e evitar a perda de dados processados, enquanto a IBM realiza a revogação de forma imediata.

Pela natureza de servidores virtuais transientes, o acesso aos seus recursos pode ser revogado do usuário a qualquer momento. Na AWS, este acesso pode ser revogado baseando-se no valor de leilão, que recebe mudanças em tempo real, baseado em variáveis imprevisíveis da Lei da Oferta e Procura. Sobretudo, o ambiente AWS disponibiliza, publicamente, informações históricas da variação em seus valores dos últimos três meses, que podem ser utilizados como insumos para modelos estatísticos e técnicas de predições definirem melhores estratégias de locação dessas instâncias transientes (Gong et al., 2015; Moreno-Vozmediano et al., 2013).

Em contraste, os provedores GCR, IBM e Azure não disponibilizam informações públicas do histórico das revogações em suas instâncias transientes. A não publicação desses dados dificulta pesquisas a utilizarem dados para criar estratégias de utilização dos recursos, como a projeção de disponibilidade do recurso e criação de estratégias e mecanismos para lidarem com a revogação abrupta das instâncias transientes.

É importante observar que instâncias transientes podem ser revogados a qualquer momento, sem permitir a intervenção desta ação. Por este motivo, é interessante que estratégias para se adequar a este cenário sejam previstas na execução das aplicações. Assim sendo, aplicações devem prever interrupções abruptas em suas execuções e adotar mecanismos que permitem a continuação e finalização de sua execução.

2.2 Tolerância a Falhas

Quando um sistema, em sua execução, não atende às suas especificações de execução e disponibilidade, considera-se que o sistema está falho (Tanenbaum and Van Steen, 2007a). Falhas, erros e defeitos são circunstâncias indesejadas e tem suas definições apresentadas

na literatura de TF (Isermann, 2006; Laprie, 1985; Lee and Anderson, 1990; Tanenbaum and Van Steen, 2007a).

Uma falha pode ser definida como um comportamento que se encontra em um estado que não está em conformidade com a sua especificação. Considera-se falha de um sistema o evento que ocorre quando há um desvio na execução de um determinado serviço, sendo possível observar uma alteração no seu resultado em decorrência da inconsistência entre a especificação e a implementação de sua função (Lee and Anderson, 1990).

Os escopos que representam a relação entre falha, erro e defeito são apresentados na Figura 2.4 (Isermann, 2006).

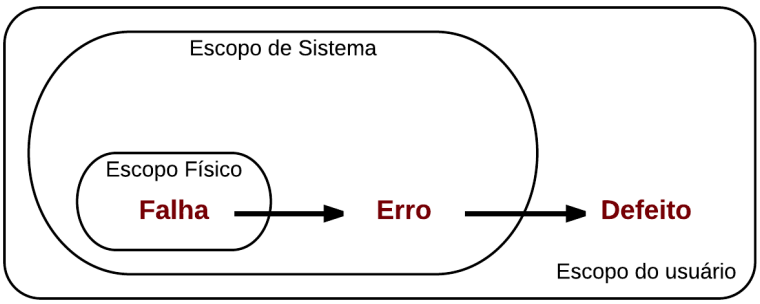


Figura 2.4: Escopos de falha, erro e defeito.

.

O estado de um sistema é considerado errôneo caso possa resultar em uma falha e após a sua identificação, é comum que o erro forneça informações que permite o conhecimento da sua causa (Isermann, 2006). Um erro está associado a uma falha ou parte de um sistema que pode causar um defeito. Desta forma, um erro pode estar presente dentro de um sistema e não ser identificado devido a não captura, identificação e tratamento do mesmo.

Por fim, um defeito é a hipótese causada na presença de um erro. Esse defeito, que pode ser classificado como físico ou humano é definido como a materialização do erro por meio de percepção do usuário (Avizienis, 1978; Laprie, 1985). Em ambiente de software, o defeito é um estado causado por um erro ou condição que provoca a execução incorreta do algoritmo.

Em Lee and Anderson (1990), os autores citam que as falhas são geralmente classificadas por medição, valorização e duração, apresentando as seguintes definições:

- Medição – classificação que se aplica a erros causados por falhas que foram identificados e sua extensão foi limitada apenas a um estado local ou distribuído;
- Valorização – indicador que permite identificar se a falha gera valores errôneos fixos ou variáveis; e

- Duração – classificação responsável por fornecer métricas que permite identificar se a falha é temporária ou permanente. Os autores definem como temporária (ou transiente) uma falha que está presente em um software durante um período limitado de tempo, desaparecendo espontaneamente do sistema. Quando a falha temporária é recorrente pode-se classificar como intermitente. Falhas que insistem em permanecer no software são classificadas como permanentes.

Em Lee and Anderson (1990) e Stanković et al. (2017), os autores declaram que o sistema pode apresentar diversos estados de falha, permitindo uma caracterização para serem analisados individualmente e definir quais técnicas de TF devem ser aplicadas para o tratamento e a execução da aplicação. Os autores em Tanenbaum and Van Steen (2007b) classificam falhas como:

- Transiente – consiste em uma falha que acontece em um instante durante a execução, normalmente de maneira imprevisível, podendo não ocorrer em execuções futuras idênticas, e.g., causada por meio de um elemento externo, como falta de comunicação ou energia durante a execução;
- Intermitente – o surgimento da falha ocorre sem uma razão aparente e de maneira esporádica. Comumente apresentada quando há uma falha de hardware ou componente defeituoso. Estas falhas ocorrem de maneira aleatória em intervalos de tempo não-determinísticos, dificultando o seu diagnóstico; e
- Permanente – consiste em uma falha presente de maneira permanente em unidades de software ou hardware que foram incluídas no momento da sua criação. Para resolver este tipo de falha deve-se realizar um processo de manutenção na unidade, para garantir que a falha foi resolvida e não volte a ocorrer.

Por meio de um processo de monitoramento e de observação de um sistema, é difícil determinar se uma falha que ocorreu é transiente ou intermitente, pois elas podem possuir comportamentos semelhantes. Os autores em Barborak et al. (1993) mostram a importância na distinção destas falhas ao mostrar que uma falha transiente não implica necessariamente que o sistema deve ser declarado defeituoso, pois um ambiente instável pode justificar um defeito temporário.

Em Lee and Anderson (1990) e Stanković et al. (2017) os autores categorizam as etapas da TF e as respectivas técnicas elencando quatro fases que provêm uma generalização de cenários. As técnicas de TF podem ser aplicadas para prevenir falhas de sistema considerando essas quatro fases.

Fase 1: Detecção do Erro

Para aplicação de TF, faz-se necessária a detecção de uma falha ocorrida em um sistema. Quando esta falha ocorre, ela é replicada entre as camadas do sistema e manifestada por meio de um erro no sistema, apresentando exceções ou eventos que permitem a identificação da sua natureza.

Como mencionado, é comum que um erro detectado forneça informações, permitindo o mapeamento e a identificação da sua causa. Desta forma, o passo inicial para aplicação de técnicas de TF é o monitoramento e a detecção do estado da aplicação na ocorrência de uma falha.

Existe a possibilidade de ocorrer uma falha no sistema e esta permanecer durante toda a execução, sem que um erro possa ser manifestado ou capturado. Isso possibilita a execução e permanência da falha no sistema, assumindo estados em que o erro nunca possa ser explicitamente detectado.

Para mitigar estas situações de falhas, medidas são incorporadas aos sistemas para prover a detecção de erros. Em Lee and Anderson (1990), os autores apresentam sete tipos de abordagem para auxiliar na identificação de um erro:

1. Verificação em replicação (*replication check*) – essa abordagem consiste na presença de n execuções simultâneas do sistema, sendo necessário que todas elas sejam monitoradas. Dessa forma, por meio de uma abordagem de monitoramento, a detecção de um erro ocorre quando há a presença destas execuções em, no máximo $n - 1$ execuções simultâneas, caracterizando a falha em algum recurso que não está mais disponível para verificação;

2. Verificação cronometrada (*timing check*) – nessa abordagem são realizadas comunicações entre os componentes de um sistema utilizando monitores para observar os seus recursos de forma temporal. A verificação permite a troca de mensagens entre o monitor e o sistema, identificando quando há um tempo de retorno acima do aceitável para um recurso monitorado. Dessa forma, o monitor sinaliza, por meio de exceções ou sinais, que um erro foi detectado;

3. Verificação reversa (*reversal check*) – essa verificação é aplicada em sistemas mais complexos que existem relacionamento entre recursos de sistema, representado em módulos ou distribuídos. Tendo *out* e *in* como, respectivamente, as representações dos dados de saída e de entrada de um sistema, essa verificação analisa *out* e realiza cálculos para produzir quais seriam os dados de entrada in'. Essa verificação faz a comparação entre os dados reais de entrada *in* e in', identificando o erro e sinalizando o sistema quando $in \neq in'$;

4. Verificação de codificação (*coding check*) – utilizando verificação em duas etapas, essa abordagem faz uso de redundância dos recursos para identificação de erros. A sua técnica consiste na verificação de execução de códigos no recurso redundante, na qual possui relação com o recurso original e realiza verificações em suas estruturas de dados que representam um valor para o recurso. É considerado erro quando não há uma paridade nessas verificações;

5. Verificação de estrutura (*structural check*) – aplicável a sistemas que possuem representação abstrata e complexa de dados, com verificação utilizando duas técnicas para detecção do erro: integridade semântica, que se preocupa com a consistência da informação contida na estrutura de dados; e integridade estrutural, que avalia a consistência da estrutura dos objetos inseridos nas estruturas de dados. Um erro é identificado quando as suas representações de dados (listas, filas, pilhas e etc.) estão com suas estruturas comprometidas. Essa diferença é validada por meio da comparação com estruturas que estão redundantes; e

6. Verificação diagnosticada (*diagnostic check*) – diferente das abordagens citadas, as quais se preocupam com a verificação do erro, por meio de mudanças nos comportamentos dos componentes do sistema, a verificação diagnosticada tem foco na mudança comportamental do próprio sistema. Ela tem conhecimento da relação entre valores de entrada e saída do sistema, fazendo uma comparação entre todas as saídas e confrontando com as entradas realizadas. É considerado erro quando não há conformidade entre os dados de entrada e saída processados.

A fase de detecção do erro deve garantir a identificação de uma operação de falha que pode levar a um defeito do sistema. A identificação correta do erro reflete no sucesso de um sistema tolerante a falhas, pois as próximas fases necessitam desta identificação para que o processo de recuperação possa ser adequadamente realizado.

Fase 2: Isolamento

Quando ocorre a suspeita de que um erro esteja presente e existe a confirmação por meio da sua detecção, o ambiente deve ser isolado. Devido ao tempo de resposta entre a manifestação e a detecção do erro, dados considerados válidos podem ter sido manipulados, gerando informações inválidas, comprometendo a execução do sistema, inclusive prospectando novos erros ainda não detectados. Para evitar esse tipo de cenário, o ambiente de execução deve ser isolado para permitir o seu tratamento.

Avaliar qual técnica de TF será aplicada depende das decisões feitas pelo projetista ao desenhar a solução para limitar a propagação da falha. Nessa fase, os recursos em que

ocorreram as falhas no sistema são isolados e, em tempo de execução, o sistema utiliza mecanismos para mensurar e identificar os danos causados.

Para garantir esse isolamento, algumas medidas são apresentadas e sua aplicabilidade depende do cenário em que está inserida. Em Lee and Anderson (1990), os autores apresentam a divisão desse processo em duas etapas:

1. Isolamento – consiste em um processo em que deve ocorrer o isolamento do recurso que ocasionou o erro e avaliar a sua projeção, preparando o ambiente para a próxima fase. As medidas aplicadas nessa etapa são mais eficazes quando o cenário envolve hardware ao invés de software. Quando a falha ocorre no hardware, o tratamento envolve ações físicas, como a retirada do componente que apresentou falha. Em se tratando de software, medidas que emulam um mecanismo de proteção devem ser aplicadas;

2. Avaliação – quando há a detecção da falha, seja por meio das verificações apresentadas ou pelos mecanismos providos pelo sistema, as suas informações são disponibilizadas por meio das exceções ou eventos. Semelhante ao item de isolamento, o tratamento do hardware também envolve ações físicas para avaliar o erro. Quando ocorre uma falha em um software são utilizas regras definidas pelo projetista para identificar o erro e aplicar o seu respectivo tratamento.

No entanto, é comum que o sistema não possibilite meios para o isolamento. Neste caso, faz-se necessário especificar limites para a propagação do dano, o que deve ser realizado ainda no projeto do sistema. Nele devem ser previstas as regras na troca de mensagens para evitar trocas acidentais e impossibilitar a verificação para detecção de erros. Assim como na detecção, as ações realizadas nessa fase também refletem na qualidade do sistema.

Fase 3: Recuperação do Erro

No processo de falha, as Fases 1 e 2 (detecção e isolamento do erro) estão em uma situação passiva, no sentido que não afetam nenhuma ação que possa realizar alterações no sistema. No entanto, na fase de recuperação do erro técnicas devem ser aplicadas para alcançar o menor índice de perda de processamento e de informação.

Utilizando uma ilustração de linha do tempo, na Figura 2.5 são apresentados os estados α e β representando pontos livres de falha, complementando a aplicação das abordagens de *backward* e *forward recovery* na presença de uma falha.

Em Lee and Anderson (1990), os autores declaram que os danos causados por um erro podem ser previstos, permitindo que algum tratamento possa ser aplicado de maneira

antecipada. Quando não é possível prever o dano, considera-se que o caso não pode ser antecipado. Então, quando existem casos previstos, as abordagens de *forward recovery* podem ser utilizadas, caso contrário, as abordagens de *backward recovery* são aplicadas.

Na abordagem *backward recovery*, técnicas são aplicadas de maneira preventiva, permitindo a restauração do sistema para um estado anterior à falha (estado α), na intenção de que esse estado não contenha erros. As recuperações desse tipo possuem um alto custo de implementação, porém permite o uso de técnicas em situações genéricas. Exemplo de implementação aplicada a essa abordagem é o uso de pontos de verificação (*checkpoint*), que permite salvar estados válidos de execução de sistema, sendo apresentado no Capítulo 3.

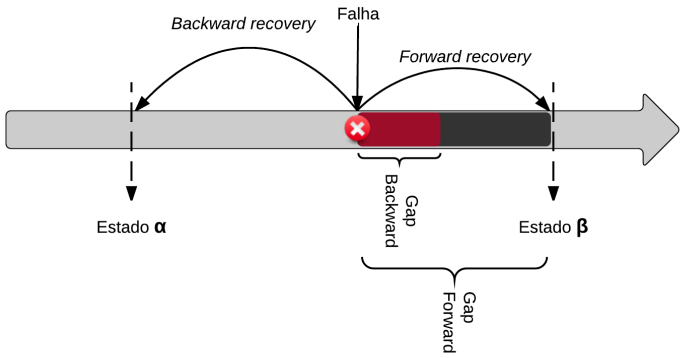


Figura 2.5: Abordagens de *Backward* e *Forward Recovery*.

Em Elnozahy et al. (2002), os autores apresentam estratégias que utilizam *backward recovery* e trazem o sistema para um estado de execução consistente, previamente armazenado, utilizando a técnica de TF denominada *checkpoint/restore*.

Já a abordagem de *forward recovery* age sobre o estado do sistema na previsão de quando ocorre o erro, aplicando técnicas que conduzem a um novo estado em que o erro não está mais presente (estado β), permitindo a continuação na execução da aplicação. Por ser uma abordagem em que a sua implementação é específica para cada sistema, necessitando de um conhecimento sobre a aplicação, suas técnicas são mais eficientes. Os sistemas que necessitam de processamento em tempo real são indicados para aplicação dessa abordagem.

Em Elnozahy et al. (2002) e Taesoon Park et al. (2002), os autores mostram que a técnica de *checkpoint/restore* insere *overhead* na execução de um sistema devido a sua necessidade de armazenar os estados da execução. As características da aplicação refletem no tempo de *overhead*, e.g., aplicações de uso excessivo de memória requer um tempo superior a aplicações menores, pois o estado de CPU e de memória devem ser armazenados. Além disso, a velocidade de transmissão de dados entre o dispositivo em

que a aplicação está sendo executada e o dispositivo de armazenamento também reflete no tempo total de *overhead*.

Como a abordagem de *forward recovery* realiza o tratamento da falha sem estados anteriores, é necessário realizar um mapeamento do que foi afetado e o que pode ser recuperado. Desta forma, o processo de recuperação da abordagem de *backward recovery* se mostra mais eficiente comparada a de *forward recovery*.

Em um sistema que utiliza apenas um único processo em sua execução, o tratamento, apesar de complexo, é mais simples do que em outros cenários. Em Coulouris et al. (2011), os autores definem um sistema distribuído como um conjunto de componentes independentes de hardware e/ou software, interconectados em uma rede de computadores, que comunicam e coordenam suas ações utilizando mensagens como meio de comunicação.

Estes componentes independentes, que são conectados em uma rede utilizando um *middleware* distribuído, podem coordenar as suas atividades de maneira centralizada ou distribuída, compartilhando os seus recursos e oferecendo um meio transparente para o usuário. As principais diferenças entre os sistemas centralizados e distribuídos são apresentadas na Tabela 2.1 (Elnozahy et al., 2002).

Tabela 2.1: Classificação dos ambientes centralizados e distribuídos.

	Centralizado	**Distribuído**
Gerenciamento	Pelo menos um componente para gerenciamento de recursos.	Múltiplos componentes.
Recursos	Todos são acessíveis; compartilhados com todos os usuários ao mesmo tempo.	Nem sempre acessível; não compartilhados com todos os usuários.
Processos	Execução em apenas um único processo.	Execução paralela de processos em diferentes processadores independentes.
Controle	Único ponto de gerenciamento	Múltiplos pontos distribuídos e independentes.
Falhas	Único ponto de falha	Múltiplos pontos de falhas independentes.

Como observada na Tabela 2.1 os sistemas que estão inseridos em um ambiente distribuído possuem propriedades que os tornam mais complexos quanto a sua execução quando comparado a abordagens centralizadas. Portanto, os sistemas com características distribuídas, pela sua natureza, possuem uma complexidade maior no tratamento de falhas, envolvendo diversos processos com vários recursos, muitas vezes distintos e independentes (Jalote, 1994; Tanenbaum and Van Steen, 2007a).

Em ambientes distribuídos é comum que processos distintos possuam relações de dependência e realizem trocas de mensagens durante a execução. Na presença de uma falha no sistema, o tratamento deve buscar um estado consistente, ou seja, um estado em que o erro não está mais presente ou que não tenha como consequência o mesmo erro. Muitas vezes é necessário realizar análises da propagação da falha em diversos recursos e processos distintos para encontrar um estado consistente.

Em Marzullo (1993), os autores definem o estado consistente como aquele estado que reflete um ponto de execução de uma aplicação que é livre de falhas em um cenário de computação distribuída que realiza trocas de mensagens.

Em Elnozahy et al. (2002), os autores complementam que este estado consistente é aquele em que, se um determinado processo destino reflete o estado e após uma mensagem m recebida, então o estado r do remetente reflete o envio desta mensagem m. Algoritmos e técnicas para detectar esses estados globais consistentes são apresentados em (Chandy and Lamport, 1985).

A Figura 2.6 apresenta dois exemplos de estados de execução, sendo representados por meio de linhas de recuperação, que simboliza o conjunto de estados locais dos processos distintos, agrupados em um ponto de execução do sistema.

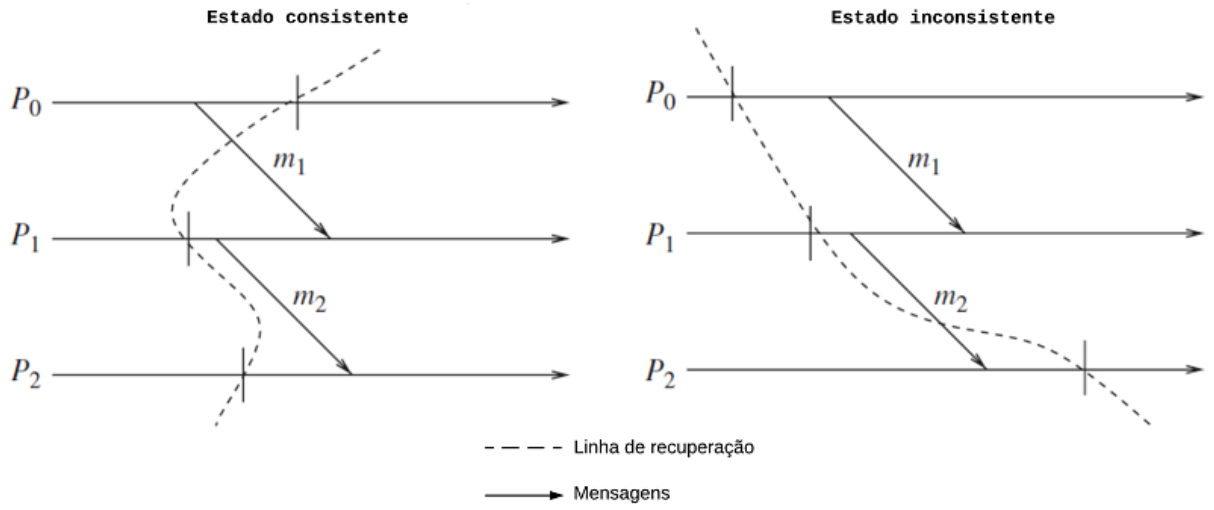


Figura 2.6: Linhas de execução representando cenários consistentes e inconsistentes.

Observando a Figura 2.6, o exemplo de estado global consistente demonstra um cenário em que o processo P_0 salva o seu estado local P_{0e0} após o envio da mensagem m_1, porém o processo P_2 salva o seu estado de execução local P_{1e0} antes de receber a mensagem m_1. Caso semelhante ocorre na transmissão da mensagem M_2 entre P_1 e P_2, em que os estados locais de ambos os processos (P_{1e0} e P_{2e0}) são armazenados antes do seu envio. Desta forma, este cenário representa um estado global consistente (E_G), sendo composto pelo conjunto de estados $E_G =< P_{0e0}, P_{1e0}, P_{0e0} >$.

Este exemplo mostra um estado consistente global porque ele apresenta um cenário em que o conjunto de estados salvos, em todos os processos, representam um instante de execução que não haverá perda de mensagens. A mensagem m_1 foi enviada do remetente (P_0) e ainda está sendo transmitida pelo canal de comunicação, i.e., não foi recebida pelo processo destinatário (P_1) e a mensagem m_2, em ambos os estados P_{1e0} e P_{0e0} não foi, respectivamente, enviada e recebida.

O exemplo de estado global inconsistente apresenta a mensagem m_2 enviada pelo processo P_1 após o estado local P_{1e0} ser salvo e, no processo P_2, estado local P_{2e0} é salvo após o recebimento da mensagem m_2. Desta forma, o estado global representado neste exemplo não é consistente, i.e. inconsistente, considerando que uma possível recuperação desta linha, composta com o conjunto de estados, faz com que a mensagem m_1 seja enviada novamente pelo processo P_0, mesmo que já tenha sido recebida pelo processo P_2 em seu estado P_{2e0}.

O protocolo de recuperação de falha para um estado consistente é chamado de *Rollback Recovery Protocol* (Elnozahy et al., 2002). Este protocolo trata uma aplicação em um sistema distribuído como uma coleção de processos que se comunicam e trocam mensagens em uma rede conectada, buscando a recuperação de um sistema a partir de um estado global consistente após identificada a falha.

É necessário considerar todas as mensagens que estão sendo enviadas e recebidas em um sistema completamente distribuído, para que possa obter estados globais e analisar a sua consistência. Como os estados de cada processo são independentes e se modificam durante a sua execução, aumenta a complexidade para que o sistema possa alcançar um estado global consistente.

A dependência na troca de mensagens obriga a análise de diversos estados locais nos seus processos, inclusive os que não apresentaram falhas. Após a identificação da falha em um ou mais processos de um sistema distribuído, a dependência das mensagens muitas vezes obriga processos em que a falha não está presente, reverterem para um estado que possui alguma falha, gerando uma propagação desta reversão (*rollback propagation*) (Elnozahy et al., 2002).

A busca por um estado global consistente, no pior caso, pode levar ao início da execução do sistema, sendo necessária a recuperação e análise de cada processo distinto e seus estados locais até que o sistema se encontre em um estado inicial de execução.

Em Randell (1975), os autores apresentam um cenário em que ocorre *rollback propagation* para o início da execução da aplicação. Isto é conhecido como efeito dominó (*domino effect*) e este cenário ocorre normalmente quando não há a presença de coordenação nos instantes em que os processos salvam os seus estados na tentativa de garantir um estado global consistente.

Observando a Figura 2.7, adaptada de Elnozahy et al. (2002), pode-se elencar vários estados globais por meio da composição dos seus estados locais, mas nenhum deles possui um estado consistente. Uma falha presente no processo P_2 após o envio da mensagem m_6 causa um *rollback propagation* devido a não consistência destes estados globais. Neste exemplo, a linha de recuperação válida (consistente) está presente no início da execução,

invalidando todos os *checkpoints* e forçando a aplicação a reiniciar sua execução caracterizando o efeito dominó.

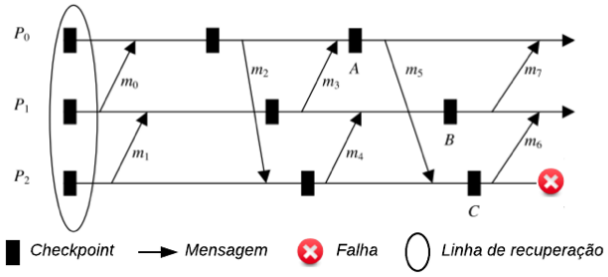


Figura 2.7: Ilustração do efeito dominó.

.

Um exemplo de técnica distribuída que permite a construção de estados globais sem a necessidade de interromper a execução de um sistema é a utilização do protocolo de *snapshots*. Por meio dos estados locais, esse protocolo consiste no envio de uma mensagem do tipo *marker* para todos os processos relacionados. Após o recebimento desta mensagem, os processos também criam seus *snapshots* locais e enviam *markers* na rede. Assim, após todos os processos receberem os marcadores e criarem seus respectivos *snapshots*, o estado global do sistema é montado pelo processo que originalmente iniciou.

Fase 4: Tratamento da Falha e Continuação do Serviço

A fase de tratamento da falha é a última fase do processo de TF (Lee and Anderson, 1990). Após a fase de recuperação, permitir a transformação do sistema em um estado em que o erro não está mais presente, é necessário. Para isso, aplicar técnicas de tratamento da falha para que o sistema possa continuar a sua execução, provendo os dados e serviços sem mudança em suas instruções definidas na especificação é fundamental.

Nessa fase, quando não há a presença de erros transientes, o sistema deve ser reconfigurado para evitar a presença de novos erros, e garantir a continuação da execução. O tratamento da falha é dividido em duas fases:

- Localização da falha – consiste na identificação do componente ou recurso em que houve a falha. Normalmente a falha é identificada quando uma exceção é provocada, porém nem sempre ela vem acompanhada de informações suficientes para que possa ser interpretada e identificada a origem da falha. Nesses casos, a reparação automatizada da falha só é possível quando a exceção provê meios para que ela possa ser identificada; e

- Reparação do sistema – mesmo na hipótese de que a falha é tratada individualmente, um ou mais recursos do sistema podem estar defeituosos após a localização da falha. As técnicas aplicadas na reparação se baseiam na reconfiguração do sistema sem afetar o seu propósito. Estas técnicas são classificadas como: (i) manual – quando há a necessidade de intervenção humana na reparação; (ii) dinâmica – quando a aplicação da técnica de reparação é executada por meio de um sistema interno ou externo, limitando-se a atingir apenas o escopo da aplicação; e (iii) espontânea – técnica em que as ações de recuperação são incorporadas no sistema.

Após a localização e reparação da falha, é necessário que o sistema retorne ao seu estado normal de execução, realizando as operações posteriores ao momento da falha sem ser afetado por ela. Em cenários em que o tratamento está relacionado ao hardware, a reinicialização do sistema é a abordagem mais aplicada devido a sua natureza de execução.

Devido à relação complexa entre falha e erro, a detecção de um erro não necessariamente serve para identificar uma falha. Um exemplo disso ocorre quando há um erro na leitura de um endereço de memória ocasionado por um problema no barramento de transmissão, no fornecimento de energia ou uma falha na unidade de memória. Por isso, a identificação do local em que houve uma falha é importante para que o erro possa encapsular a informação.

A definição de qual técnica deve ser aplicada tem forte relação com a natureza da falha ocorrida. Em Fedoruk and Deters (2002), os autores apresentam os seguintes tipos de falhas:

- *Bugs* no programa – consiste em erros que não são detectados no processo de testes e normalmente estão associados a não concordância com a especificação do sistema;

- Estados desconhecidos – além dos erros não detectados no processo de testes, os erros são omitidos no projeto do sistema, em que determinados estados de falhas não são manipulados pela programação;

- Falhas de processamento – abrange falhas de sistema operacional ou interrupção no uso de recursos de um sistema, em que o processador não está disponível para a execução do serviço;

- Falhas de comunicação – envolve problemas na comunicação, incluindo lentidão, interrupções ou qualquer outro problema que impeça a troca de mensagens entre recursos de um sistema, criando cenários em que os componentes do sistema não possuam garantias no envio ou recebimento de mensagens.

Com foco inicial no tratamento de erros em operações de hardware, técnicas de TFs vem sendo utilizadas desde 1950, com o objetivo de aumentar o nível de confiança no funcionamento operacional de seus recursos (Elnozahy et al., 2002; Lee and Anderson, 1990).

Assim, consideráveis esforços vem sendo dedicados em pesquisas para prover métodos de tratamento da falhas (Bian et al., 2008; Elnozahy et al., 2002; Gokhale et al., 1998; Guimaraes et al., 2014; Hwang and Kesselman, 2003; Lee and Anderson, 1990). A aplicação de técnicas de TFs permite que um sistema reduza os impactos negativos ao incorporar a habilidade de realizar tratamentos quando falhas ocorrem dentro de um sistema.

As técnicas de TFs podem ser aplicadas em sistemas computacionais de diversas naturezas. Aplicações tolerantes a falhas reduzem os impactos negativos ao incorporar a habilidade de aplicar técnicas que permitem o tratamento na identificação de uma falha e recuperá-la.

Além da técnica de *checkpoint/restore*, apresentada na Seção 2.2, outras técnicas existentes na literatura também buscam reduzir os impactos negativos na presença de falhas. Em Elnozahy et al. (2002) e Guimaraes et al. (2014), os autores mostram que replicação e *retry* estão entre as principais técnicas de TFs utilizadas.

Utilizando técnicas de redundância, a replicação provê recursos extras que executam a mesma aplicação de forma paralela. Em Guerraoui and Schiper (1996), os autores apresentam duas principais classes de técnicas de replicação: ativa e *backup* primário (passiva).

Considerando um processo principal pm e $P = \{p_1, ..., p_n\}$ como um conjunto de processos em máquinas distintas, tendo $pm \not\subset P$, tem-se:

1. Replicação Ativa – sem controle centralizado, esta técnica consiste em execuções simultâneas do processo sem dependência entre elas. Considerando pm replicado em n máquinas distintas, todas as réplicas irão refletir a sua execução de maneira independente. Em um caso de falha no processo pm, por meio de regras definidas pelo projetista, um processos $p' \in P$ assume a sua execução sem interromper a sua execução.

2. Replicação em Backup Primário (passiva) – diferente da abordagem ativa, esta técnica consiste em um processo primário e centralizador $pc = pm$ que se comunica com outras execuções para sinalizar pontos de marcação de sua execução. Em intervalos de tempo, pc invoca $p' \in P$ para que seja realizado uma marcação da execução. Em caso de falha no processo pc, um processo p' assume a execução a partir da última marcação sinalizada por pc.

Em Guerraoui and Schiper (1996), os autores apresentam as seguintes relações entre as replicações ativas e passivas: (i) replicação ativa requer que a execução em suas réplicas sejam determinísticas para que seja possível ter efetividade nesta técnica, o que não acontece na abordagem passiva; (ii) com a replicação ativa, a falha do processo *pm* não sofre perda de processamento considerável, já que uma replica a substitui.

Na abordagem passiva, o estado latente pode ter um crescimento considerável, o que pode ser inaceitável para sistemas em tempo real; e (iii) por necessitar de n execuções simultâneas, a replicação ativa normalmente consome mais recursos do que a passiva, que realiza o processamento apenas quando são invocadas pelo processo *pm*.

Para o provimento de TF utilizando replicação, a redundância é fundamental e ela pode ocorrer de duas maneiras: espaço e tempo (Gärtner, 1999). A redundância por espaço é o mecanismo mais utilizado, em que provê a duplicação ou replicação de recursos por meio da replicação de tarefas e dados.

A replicação de tarefas consiste em criar múltiplas execuções paralelas (réplicas) de uma única tarefa, sendo executadas simultaneamente (Cirne et al., 2007). Naturalmente, quando identificada que a execução de uma tarefa replicada foi finalizada, o resultado da sua execução é o único considerado e as outras execuções são finalizadas, evitando a utilização improdutiva de recursos.

Conhecida também como re-submissão de tarefa, a técnica de *retry* consiste na reinicialização da execução de um processo quando uma falha ocorre. Esta técnica de TF realiza uma nova tentativa de execução, normalmente em uma máquina diferente da que originou a falha, desconsiderando os seus estados de execução e reexecutando os procedimentos desde o seu início. Dentre as técnicas de TF apresentadas, esta é a abordagem menos complexa de tratamento de falhas, que pode ser representada por meio da técnica de *checkpoint/restore* sem intervalos de *checkpoint*.

2.3 Inteligência Artificial

Inteligência Artificial (IA) pode ser definida como uma sub-área da Ciência da Computação que se preocupa com a automação do comportamento inteligente. Essa área de estudo envolve processos computacionais que reproduzam o comportamento humano inteligente envolvendo raciocínio lógico, aprendizagem, compreensão de linguagem natural e resolução de problemas (Barr and Feigenbaum, 1981; Luger and Stubblefield, 1993).

Em Russell and Norvig (2010) encontra-se quatro categorias de definições de IA que representam uma visão evolutiva da área, incluindo deste sua criação (década de 50) até os dias atuais: pensar como os humanos, agir como os humanos, pensar racionalmente e

agir racionalmente. Neste sentido, a visão moderna de IA dá ênfase ao desenvolvimento de entidades que possam agir de forma racional.

A visão de máquinas que pensam como os humanos está baseada no Teste de Turing apresentado em seu famoso artigo *Computing Machinery and intelligence*, conhecido como o jogo da imitação (Turing, 2009). Turing previu que em 2000 a máquina teria 30% de chance de enganar uma pessoa por cinco minutos, e desta forma antecipou os maiores argumentos contra IA nos 50 anos subsequentes. A Figura 2.8 ilustra o teste de Turing, que propõe a impossibilidade de distinguir entre um computador e um ser humano.

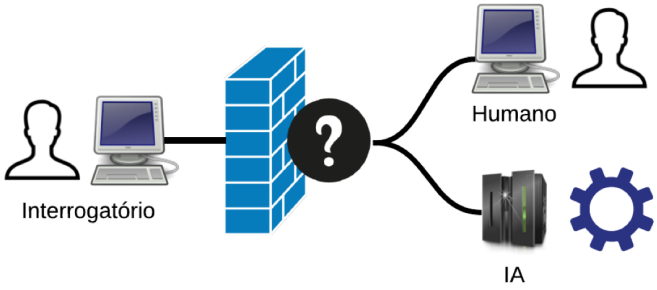


Figura 2.8: Ilustração do teste de Turing.

No cenário apresentado na Figura 2.8, o computador passará no teste se um interrogador humano não conseguir diferenciar, ao final de uma série de perguntas, se as respostas foram originadas por um ser humano ou por um sistema computadorizado. Essa é considerada uma das primeiras definições operacionais para a representação e validação de IA (Shieber, 1994; Turing, 1950).

O teste de Turing, apesar de aparentemente simples, envolve desafios relacionados a integração de diversas sub-áreas de IA, tais como processamento de linguagem natural, tratamento de conhecimento, raciocínio automatizado e aprendizado de máquina. Considerando esses aspectos, ainda não há solução racionalmente equivalente a capacidade cognitiva do ser humano.

Neste sentido, a entidade computacional inteligente é caracterizada pelo agente que pensa e age de forma racional, tomando a decisão correta para maximizar o alcance de objetivos, conforme as informações disponíveis no momento da decisão.

Aprendizado de Máquina

A IA estuda o desenvolvimento de métodos e técnicas computacionais para aquisição automatizada do conhecimento, permitindo a construção e a representação do seu aprendizado. Em Simon (1983), o autor apresenta o conceito de aprendizado de máquina como um processo que envolve modificações adaptativas e evolutivas nos sistemas, que oferece

a capacidade de realizar tarefas programadas ou provenientes da mesma população. Além disso, permite uma avaliação dos resultados produzidos para promover um acréscimo na eficiência e utilidade em suas próximas execuções.

Um sistema com aprendizado de máquina pode ser entendido como o treinamento de uma função utilizada para tomada de decisão baseada em experiências prévias com sucesso (Kulikowski and Weiss, 1991; Russell and Norvig, 2010). Neste sentido, o aprendizado não está ligado apenas a aquisição de novos conhecimentos, mas também a manutenibilidade da base de conhecimento adquirido, a qual será aplicada como raciocínio para a tomada de decisão na resolução de problemas.

Apesar do aprendizado de máquina oferecer uma ferramenta poderosa para aquisição e manutenção automatizada de conhecimento, a construção de um algoritmo genérico que alcance a mesma eficiência na resolução de problemas de forma geral é muito complexa (Sutton and Barto, 2018).

Os sistemas inteligentes devem considerar os estilos de aprendizagem automática que deverão ser adotados, bem como os tipos de algoritmo disponíveis para uso (Kotsiantis et al., 2007). Uma classificação dos tipos de aprendizado indutivo baseado em inferência lógica, que permite obter conclusões a partir de um conjunto de exemplos particulares de entrada (*particular* $\rightarrow$ *geral*) são: supervisionado, não-supervisionado e semi-supervisionado.

A indução é uma forma de inferência lógica relacionada a IA, que utiliza exemplos externos ao sistema de aprendizado para permitir que o raciocínio e decisão sejam baseadas em experiências anteriores. Esta abordagem permite obter conclusões a partir de um conjunto materializado de exemplos. No aprendizado indutivo, casos são absorvidos utilizando um inferência indutiva sobre os exemplos apresentados, efetuado a partir de uma massa de dados coletadas externamente, sempre havendo uma cautela nas escolhas e relevâncias.

No aprendizado supervisionado, o classificador (indutor) é construído para que seja determinada a classe de novos exemplos a partir de classes de exemplos previamente rotulados. Existem duas abordagens: a classificação de rótulos com uso de valores discretos e a regressão, que utiliza valores contínuos para rotular os exemplos.

Desta forma, o aprendizado supervisionado está associado a classificação de problemas com uso de exemplos ou dados de entrada classificados previamente, viabilizando a fase de treinamento. Durante o processo de aprendizado um conjunto de dados de entrada são manipulados e categorizados para indicar a qual classe eles pertencem. Como exemplos de algoritmos pode-se citar *Support Vector Machine (SVM), naive bayes, decision trees, random forest, gradient boosting, k-nearest-neighbors, neural network, multi-layer perceptron.*

O processo de treinamento para aquisição do conhecimento é contínuo e permite a sua correção quando houver previsões consideradas errôneas. É importante reter e manter o conhecimento, e esta continuidade é necessária para que o modelo alcance um nível aceitável de acurácia nos dados adquiridos durante o treinamento. A riqueza no detalhamento do modelo de aprendizagem se dá por meio da qualidade dos dados adquiridos, possibilitando a criação de um modelo rico de informações que pode ser utilizado, por exemplo, para produzir predições.

No processo de aprendizado não-supervisionado o algoritmo determina como os dados de entrada estão organizados por meio de treinamento de exemplos de entrada não categorizados ou rotulados. Neste caso, a produção de predições torna-se mais desafiadora, pois o modelo se baseia em exemplos não-rotulados que precisam ser inseridos em uma estrutura semi-categorizada para que o algoritmo possa aprender, organizar os dados e produzir previsões mais confiáveis. O modelo é produzido pela dedução na estrutura presente nos dados de entrada, que por natureza, não possui organização na sua representação (Dougherty et al., 1995; Luger and Stubblefield, 1993).

No processo de aprendizado semi-supervisionado são utilizados dados de entrada rotulados e não rotulados para o treinamento, sendo uma abordagem interessante quando não se tem a possibilidade de rotular muitos dados de entrada (Chapelle et al., 2009). Existe ainda o aprendizado por reforço, o qual não utiliza dados de entrada, mas aprende a partir de descoberta de eventos diretamente por meio da interação com o ambiente, aplicando uma função de máxima recompensa possível (Sutton and Barto, 2018; Sutton et al., 1998).

O resultado destes processos é um classificador (indutor), criado também com o auxílio de um especialista que materializa o conhecimento adquirido durante o aprendizado de máquina. Este classificador, que organiza a estrutura dos dados que representa o aprendizado de máquina, é um algoritmo de aprendizagem que visa a criação e a manutenção de um bom classificador a partir de um conjunto de exemplos, permitindo, quando necessário, que seus dados possam ser modificados para adaptar a novas realidades.

Segundo Rezende (2003), o aprendizado de máquina possui cinco paradigmas:

1. Simbólico – o processo de aprendizado tem como resultado representações lógicas, organizadas como estruturas simbólicas com a utilização de exemplos e contraexemplos. Exemplos de técnicas que utilizam este paradigma são: agentes inteligentes e árvores de decisão (Russell and Norvig, 2010).

2. Estatístico – neste paradigma são utilizados métodos estatísticos que possibilitam a criação de hipóteses para produzir resultados aproximados ao esperado. O modelo Bayesiano é um exemplo de modelo estatístico (Korb and Nicholson, 2010).

3. Conexionista – utiliza uma metáfora biológica com as conexões neurais do sistema nervoso humano, simulando Redes Neurais Artificiais (RNA). Este paradigma possui como principal característica o aprendizado por meio de poder de generalização, envolvendo unidades que representam os neurônios e sinapses altamente interconectadas. RNA são indicadas na resolução de problemas que envolvem a necessidade de processamento sensorial humano, tais como visão, reconhecimento de imagens, audição, reprodução e reconhecimento de voz (Demuth et al., 2014; Rowley et al., 1998).

4. Evolutivo – inspirado na teoria de Charles Darwin, que trata da evolução das espécies e sua preservação por meio da seleção natural. Este paradigma trata da derivação do modelo evolucionário de aprendizado utilizando indivíduos de uma população como uma possível solução. Inseridos em um cenário em que os indivíduos competem entre si em uma população, aqueles que representam um menor valor para o domínio são descartados, permanecendo apenas os indivíduos (soluções) que oferecem melhores resultados. Neste paradigma, estes indivíduos possuem a capacidade de reprodução, gerando novos indivíduos que podem sofrer mutação evolutiva ou regressiva. Algoritmo genético é um exemplo de técnica que está inserida neste paradigma (Von Neumann et al., 1966).

5. Baseado em exemplo – também chamado de memorização ou *instance-based*, em que são utilizados exemplos de casos classificados anteriormente para resolver problemas para classificar exemplos novos por meio de exemplos similares conhecidos. Desta forma, padrões são identificados baseados em exemplos semelhantes anteriormente utilizados com sucesso. O raciocínio baseado em casos (*Case Based Reasoning* - CBR) é um exemplo de técnica que utiliza este paradigma (Aamodt and Plaza, 1994).

A seguir será detalhado o modelo de CBR por ser o modelo utilizado neste trabalho, o qual oferece mecanismos de raciocínio e aprendizagem baseado em experiências anteriores para dar suporte ao processo de tomada de decisão.

Raciocínio Baseado em Casos

O modelo de CBR oferece meios para que inferências possam ser realizadas no processo de resolução de problemas, considerando um conhecimento anterior baseado em casos previamente conhecidos (Aamodt and Plaza, 1994; Kolodner, 2014).

Em Aamodt and Plaza (1994), os autores definem um fluxograma que representa o ciclo básico de funcionamento do CBR como um conjunto de quatro etapas sequenciais, apresentado na Figura 2.9, adaptada dos mesmos autores:

1. Consiste na recuperação de casos anteriores em que, dado um problema, um ou mais casos de sucesso são recuperados na sua base de casos;

2. Casos recuperados servem de referência para adaptações do novo caso para a resolução do problema;

3. É realizada a avaliação dos resultados apresentados após a aplicação do novo caso;

4. Retenção do novo caso, incrementando a base de casos (conhecimento).

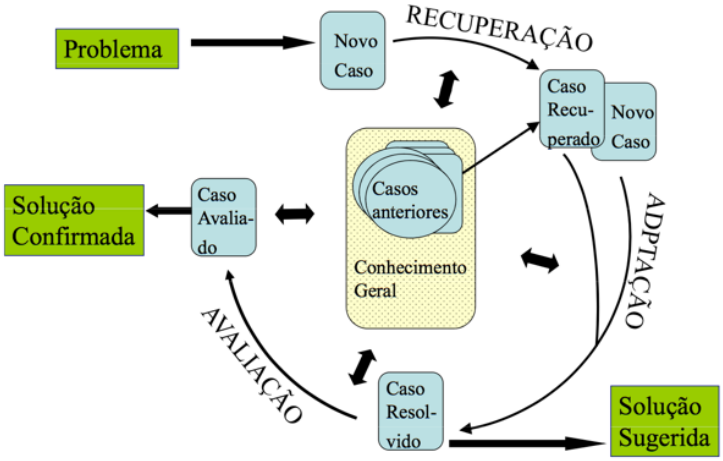


Figura 2.9: Funcionamento do CBR.

Conforme exposto, sistemas que utilizam o CBR possuem a capacidade de aprendizado contínuo por meio de experiências anteriores e adaptações nas avaliações de novas experiências. Pela capacidade de equivalência e a adaptação própria do modelo CBR, foi adotado neste trabalho como mecanismo adequado de aprendizado de máquina.

Além de envolver o entendimento de casos anteriores para auxiliar nas decisões semelhantes, o modelo trabalha com os casos de insucesso, permitindo a sua adaptação na base de casos (Kolodner, 2014). Com a inclusão de um novo problema, casos semelhantes são recuperados em uma base de casos previamente conhecidos e adaptados para adotar soluções de sucesso.

Diferente de outros métodos de aprendizado de máquina, o modelo CBR não necessita de uma quantidade considerável de dados de entrada. Desta forma, é possível construir um protótipo antes de obter a completa estruturação do domínio, possibilitando a aprendizagem de novos casos de forma incremental.

Após a finalização do caso, uma etapa de avaliação torna-se necessária para a sua aprendizagem. Para este aprendizado, a base de casos pode ser incrementada ou atualizada após a observação dos impactos gerados, e após a definição da solução sugerida previamente.

Um sistema inteligente deve ser capaz de adaptar-se a novas situações, raciocinar, entender relações entre fatos, descobrir significados, reconhecer a verdade e aprender com base em sua experiência. Diante deste cenário, o uso de agentes inteligentes se torna interessante e adequado (Wooldridge, 2009).

O esforço em entender o funcionamento interno de agentes incorporados a ambientes reais ampliou o uso, sendo atualmente utilizados em diversos domínios de aplicação, tais como jogos, autonomia em transações monetárias, investigações forenses, diagnósticos médicos e robótica (Brenner et al., 2012; Kendrick et al., 2018; Pendharkar and Cusatis, 2018; Rao et al., 1995).

Agentes Inteligentes

Em Russell and Norvig (2010), os autores definem um agente como uma entidade de software que possui a capacidade de perceber o ambiente em que está inserido por meio de sensores, e agir de maneira racional sobre esse ambiente por meio de seus atuadores.

Essas entidades autônomas, também conhecidas como agentes inteligentes, são recursos computacionais que possuem capacidade de resolução de problemas. A representação básica de um agente é apresentada na Figura 2.10 (Russell and Norvig, 2010).

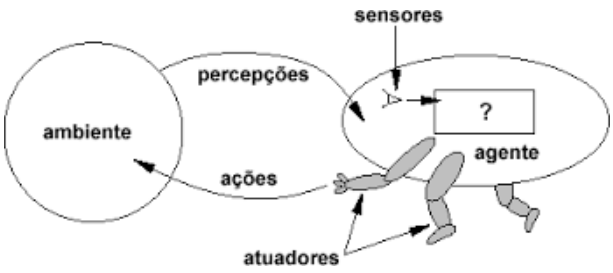


Figura 2.10: Representação de um agente.

Em Russell and Norvig (2010), os autores definem propriedades essenciais para que uma entidade de software possa ser considerada um agente inteligente: (i) autonomia, para ter a condição de executar a maior parte de suas ações sem que haja a intervenção humana; (ii) exploração – para perceber e interagir com o ambiente em que está inserido; (iii) aprendizagem, para ter a capacidade de aprender por meio das observações e interações com o ambiente ou relações com outros agentes; e racionalidade, para executar suas ações envolvendo raciocínios lógicos para agir de forma racional.

Além das propriedades essenciais de um agente apresentadas, um agente deve possuir uma capacidade perceptiva e cognitiva, sendo capaz de exibir confiança, ter poder de argumentação e negociação, planejar suas ações de forma semelhante aos humanos (Wooldridge and Jennings, 1995). Dessa forma, para ser considerado um agente inteligente, as seguintes habilidades devem estar contempladas em seu projeto:

- Mobilidade – permitir o agente ter a capacidade de se deslocar, de maneira autônoma e sem intervenção, em um ambiente no qual está inserido;
- Veracidade – possibilitar indagar sobre as informações e os cenários observados, para que possa agir sobre ideais verdadeiros e não propagar informações falsas de maneira proposital;
- Capacidade de reação – possuir uma característica em que possa perceber, reagir e se adaptar às mudanças no ambiente em que estiver inserido, modificando o seu estado interno e, quando aplicável, realizar interações com o ambiente;
- Benevolência – assumir comportamentos produtivos, buscando sempre conquistar os objetivos nos quais foram projetados no ambiente em que está inserido;
- Capacidade pró-ativa – além de atuar em resposta às alterações ocorridas em seu ambiente, agentes do tipo deliberativo apresentam um comportamento orientado a objetivos, tomando iniciativas quando julgarem apropriado;
- Continuidade temporal – executar processos, de maneira contínua e ininterrupta, podendo estar ativos ou adormecidos (*foreground* ou *background*);
- Orientado a objetivos – buscar sempre um objetivo, possuindo a capacidade de observar mudanças no ambiente e, por meio dos seus sensores, identificar mudanças e ser capaz de aplicar raciocínios lógicos, lidando com tarefas complexas de alto nível;
- Habilidade social – possuir a capacidade de se comunicar e realizar ações interativas com outros agentes para, quando aplicável, auxiliá-los na resolução de problemas ou adquirir conhecimento e habilidades que possam ser incorporadas nas suas ações.

Em Wooldridge and Jennings (1995), os autores declaram que um agente deve ter três recursos para ser considerado um agente inteligente: (i) reatividade, para perceber um evento e responder para satisfazer seus objetivos; (ii) pró-atividade, um comportamento direcionado a objetivos para tomar a iniciativa de alcançar seus objetivos; e (iii) capacidade social de interagir com outros agentes.

Segundo Russell and Norvig (2010), a programação orientada a agentes envolve o uso de agentes como a principal unidade de encapsulamento, sendo composta por um conjunto de elementos independentes, como seus objetivos, escolhas, habilidades e crenças. Os agentes são classificados de acordo com seu atributo principal, conforme ilustrado na Figura 2.11, que demonstra a composição de três elementos (cooperação, aprendizagem e autonomia) para a caracterização de um agente inteligente (Nwana, 1996).

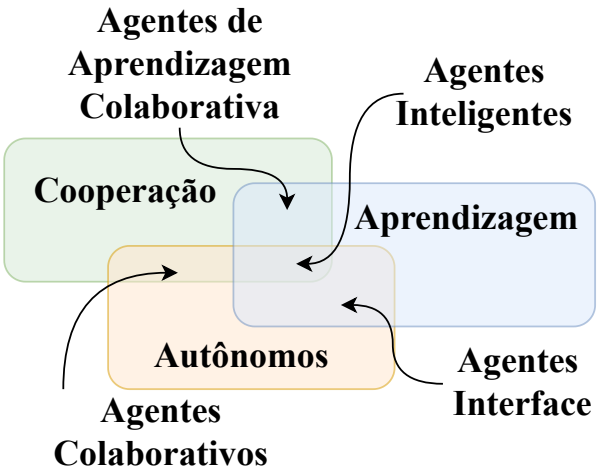


Figura 2.11: Topologia multiagente e suas classificações.

A caracterização do ambiente define a complexidade do projeto de agente inteligente. Quanto mais limites existirem sobre o ambiente, ou quanto mais estático, menos esforço é exigido no projeto. A qualidade das decisões de um agente está relacionada a quantidade e, principalmente, na qualidade das informações providas pelo ambiente em que está inserido (Kaminka, 2000; Moore, 1984).

Dessa forma, operar em ambientes que restringem o acesso às informações torna-se mais difícil do que em ambientes no qual as informações providas são suficientes para representar o cenário de execução (Weyns et al., 2004). Segundo Russell and Norvig (2010), os ambientes são classificados conforme Tabela 2.2 (Russell and Norvig, 2010).

Tabela 2.2: Classificação dos ambientes em cenários multiagente.

Classificação	**Descrição**
Completamente observável x Parcialmente observável	Caracteriza o ambiente sem restrições de acesso, em que os sensores dos agentes detectam todos os aspectos relevantes do ambiente. Sendo parcialmente observável, o ambiente possui restrições de acesso, limitando a visualização do agente a determinados recursos do ambiente.
Determinístico x Estocástico	Se o próximo estado do ambiente é completamente determinada pela ação, caracteriza-se como determinístico, caso contrário, como estocástico.
Episódico x Sequencial	Episódicos são aqueles em que a ação do agente é atômica. Sequenciais são mais complexos que ambientes episódicos porque uma decisão atual pode afetar uma decisão futura.
Estático x Dinâmico	Se o ambiente puder se alterar enquanto um agente está deliberando, é dinâmico, caso contrário é estático.
Discreto x Contínuo	Quando o tempo é finito, considera-se discreto. Se os aspectos temporais e espaciais são infinitos e as variáveis envolvidas variam de acordo com os ciclos de execução, contínuo.
Agente único x Multiagente	Definido a partir da quantidade de agentes no sistema. Agente único quando apenas um agente é inserido no sistema, multiagente quando houver mais de um agente.
Conhecido x Desconhecido	Possui relação ao estado do conhecimento dos agentes. O ambiente é conhecido quando toda ação realizada pelos seus agentes produzem saídas visíveis. O contrário acontece quando o ambiente é desconhecido, em que os agentes não tem acesso aos resultados das ações, forçando-os a um aprendizado para que possam tomar boas decisões.

Arquiteturas de Agentes

A definição do ambiente afeta diretamente o projeto de agentes, inclusive na escolha do mecanismo de raciocínio adequado a ser utilizado. Em Russell and Norvig (2010), os autores classificam cinco tipos de agentes, os quais são compostos por uma arquitetura básica relacionada aos aspectos de implementação da função que mapeia uma sequência de percepções em ações ($f : P^* \rightarrow A$):

- Agente reativo simples – baseia-se em regras de condição-ação apresentando um comportamento puramente reativo, o qual não considera históricos de percepções e ações. Como não há a presença de representação explícita do conhecimento, o algoritmo que implementa esse tipo de agente interpreta a entrada, verifica a regra correspondente e age por meio de um comportamento do tipo estímulo/resposta (Ferber, 1999). Adequado para ambientes observáveis, determinístico, episódico, estático, discreto e com poucas tarefas;

- Agente reativo baseado em estado – as ações são condicionadas pelas experiências adquiridas em interações anteriores com o ambiente. Dessa forma, o algoritmo do agente interpreta a entrada, analisa a experiência prévia, avalia as suas modificações no ambiente, e verifica a regra correspondente para utilizar (Müller and Ferber, 1996). Adequado para ambientes observáveis ou parcialmente observáveis, determinístico, episódico, estático e discreto;

- Agente baseado em objetivo – herdam características de agentes reativos baseados em estados, porém são mais flexíveis pois utilizam regras de comportamento que possuem objetivos. Para cada comportamento candidato à execução, os seus impactos no ambiente são analisados para que não confronte com os seus objetivos. É possível que diferentes comportamentos possam ser utilizado e refletir em um mesmo estado do ambiente, desde que direcionados para o alcance de seus objetivos previamente definidos. Em seu algoritmo o agente observa e analisa o estado atual do ambiente, quais os impactos de suas ações nas mudanças de estados, e age para conseguir atingir o seu objetivo;

- Agente baseado em utilidade – objetivos sozinhos podem não ser suficientes para resultar em uma ação com alta qualidade. Para isso, faz-se necessário definir uma função de utilidade que mapeie estados em um resultado de utilidade com um grau de satisfação associado. Essa utilidade pode ser calculada, por exemplo, pela média entre todos os estados resultantes, ponderados pela probabilidade do resultado. Além de serem utilizados como analisadores de conflitos, os valores de utilidade traduzem em uma proximidade de seus objetivos quando produzem valores elevados.

Por possuírem capacidades para obter e avaliar a utilidade em seus comportamentos, agentes desse tipo conseguem ser mais racionais e agir em ambientes não determinísticos;

- Agente com aprendizagem – incluindo características dos tipos anteriores, agentes com aprendizagem operam em ambientes inicialmente desconhecidos e se tornam mais competentes do que seu conhecimento inicial poderia permitir. Esses agentes possuem quatro componentes conceituais básicos: elemento de aprendizado - executa aperfeiçoamentos nas suas percepções; elemento de desempenho - realiza a seleção de ações externas; autocrítica - verifica como o agente funciona e determina como o elemento de desempenho pode ser melhorado para o futuro; e gerador de problemas - sugere ações que levarão a novas experiências e informações. Por meio do aprendizado, o agente adquire experiência e aperfeiçoa seu raciocínio, refletindo no alcance do seu desempenho e seleção dos seus objetivos.

De maneira geral, em Wooldridge and Jennings (1995) os autores apresentam três grandes categorias que representam de forma complementar a arquitetura de agentes:

- Arquitetura com agentes de raciocínio simbólico (1956 até hoje) – seguindo uma abordagem clássica de IA (IA simbólica), os agentes utilizam raciocínio lógico explícito (deliberativa) para decidir suas ações baseado em conhecimento do mundo (e.g., seleção de ações via prova automática de teoremas).

- Arquitetura com agentes reativos (1985 até hoje) – caracterizada por um modelo de tomada de decisão que não utiliza métodos de raciocínio simbólico com representações lógicas para serem provadas em tempo de execução, mas com comportamentos definidos em tempo de projeto.

- Arquitetura híbrida (1990 até hoje) – aplicando comportamentos de acordo com o estado atual, essa arquitetura herda as propriedades da arquitetura deliberativa e reativa.

Considerando que os agentes são visualizados como sistemas intencionais, ou seja, possuem estados mentais de informação e manipulam o conhecimento, a arquitetura denominada BDI (*Belief-Desire-Intention*) representa estados mentais internos nos seus agentes, com as seguintes características: crenças (*beliefs*), desejos (*desires*) e intenções (*intentions*) (Georgeff et al., 1999; Rao et al., 1995; Velleman and Bratman, 1991).

A arquitetura BDI possibilita a criação de SMA mais dinâmicos, semelhantes ao comportamento humano, em que os agentes buscam atingir seus objetivos por meio de suas

crenças, que podem sofrer alterações ainda em tempo de execução. Trabalhos que envolvem agentes para TF com diferentes técnicas em aplicações de diversas naturezas são apresentadas no Capítulo 3.

Sistemas Multiagentes

A área de IA, dentro da ciência da computação, trata de uma ciência que está em evidência por abordar temas que despertam o interesse e a curiosidade, como a possibilidade de criar entidades inteligentes. Nesse sentido, uma das principais áreas de pesquisa da IA e Inteligência Artificial Distribuída (IAD) que permite a execução de sistemas em um ambiente distribuído é a de Sistemas Multiagentes (SMA).

Na abordagem de SMA, é possível representar ambientes e entidades com capacidade cognitiva, colaborativa ou competitiva, permitindo reproduzir problemas que se assemelham com o ambiente real. Um agente representa uma entidade autônoma para resolução de problemas e o agrupamento desses agentes de maneira distribuída e colaborativa é chamada de SMA (Wooldridge and Jennings, 1995).

Em seu artigo, Michel et al. (2009) mostra que Sistemas Multiagentes (SMA) são vistos como um promissor instrumento de representação do conhecimento e de apoio à decisão, sendo uma tecnologia interessante para simular comportamentos humanos em sociedade por meio dessas entidades autônomas.

Um fator fundamental para garantir interoperabilidade e comunicação entre agentes heterogêneos é a utilização de uma linguagem de comunicação entre os agentes, recurso que permite a comunicação entre agentes projetados em diferentes tecnologias. Algumas linguagens foram criadas no âmbito da comunicação multiagente, e.g. *Knowledge and Query Manipulation Language* (KQML) e *Knowledge Interchange Format* (KIF). Entre as mais utilizadas, destaca-se a FIPA - ACL (Foundation for Intelligent Physical Agents - Agent Communication Language), que reduziu e padronizou o número de performativas para simplificar o projeto de SMA (Bellifemine et al., 1999).

Projetar um SMA significa especificar, rigorosamente, a maneira em que os agentes devem se comunicar por meio de protocolos de interação. Isso permite a presença de agentes heterogêneos em um sistema.

Dentre os protocolos de interação existe o *Publish/Subscribe* (Eugster et al., 2003), que fornece um meio simples e transparente de comunicação entre os agentes, com um protocolo de notificação de alto nível para promover cooperação eficaz por meio da observação e disponibilização de eventos para agentes que possuem interesses em comum (Platon et al., 2007).

Este protocolo permite que um agente se cadastre para ser informado quando um determinado conjunto de eventos ocorra. Normalmente gerenciado por um controlador

centralizado, todos os eventos e mensagens do sistema são capturados e catalogados em um conjunto de eventos $EV = ev_1, ev_2, ev_3, ..., ev_n$. Após isso, os agentes que estão interessados são identificados e, para cada evento observado $ev \in EV$, o controlador analisa e identifica o conjunto de agentes que possui interesse em ev, sinalizando-os com os dados relacionados a este evento ev, como apresentada na Figura 2.12.

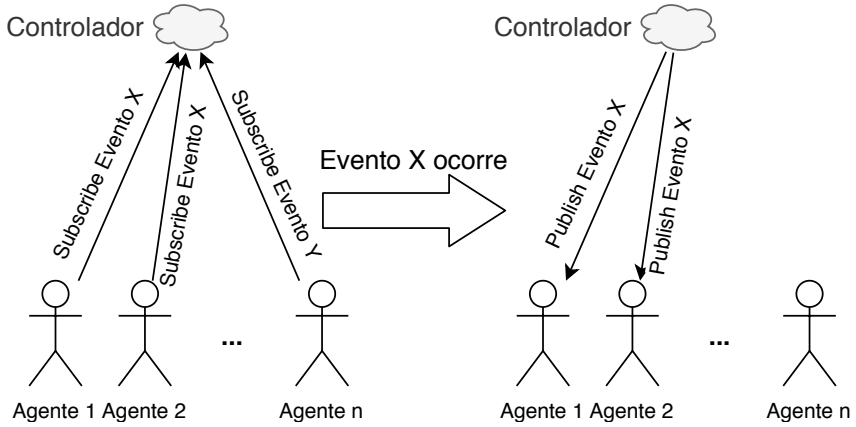


Figura 2.12: Funcionamento do protocolo *Publish/Subscribe*.

Na Figura 2.12, é possível observar o funcionamento deste protocolo, que consiste em duas etapas, a saber:

1. *Subscribe* (inscrição): quando um agente tem interesse em um determinado evento ev, inicialmente ele deve informar o controlador que deve ser notificado quando este evento ev ocorrer dentro do sistema. Neste momento, o controlador tem o mapeamento de todos os agentes que possuem interesse em cada evento conhecido;
2. *Publish* (publicação): Detalhes do evento ocorrido são enviados para cada agente, identificado por meio do mapeamento de inscrição do controlador. A ocorrência do evento é informada apenas para os agentes que estão interessados, acompanhados dos dados que envolvem o evento ocorrido, e.g. foi enviado um evento de falha em uma VM X, portanto todos os agentes que estão interessados na falha desta VM deverão ser informados. A ação do comportamento que será adotado na identificação deste evento não é controlado pelo controlador, a regra está inserida no agente que recebe a informação.

Considerado como um mecanismo atrativo para observar e estudar os impactos em um sistema por meio de variações em seus parâmetros, as simulações consistem em uma execução de modelos computacionais que representam ambientes e operações do mundo real (Gokhale et al., 1998). Portanto a arquitetura de um agente é fortemente influenciada pelas características do ambiente em que eles serão inseridos.

2.4 Análise de Sobrevivência

Modelos estatísticos oferecem mecanismos para extração de informação, muitas vezes com dados incompletos, prestando subsídios para organizar, analisar e apresentar dados que possam ser úteis na tomada de decisão.

A Análise de Sobrevivência é uma área da Estatística que trabalha com técnicas de confiabilidade, sendo aplicada em casos em que não é observada uma distribuição normal (Kishore et al., 2010). Segundo Colosimo and Giolo (2014), a Análise de Sobrevivência é uma das áreas que mais cresceu nas últimas décadas devido ao aprimoramento de técnicas estatísticas combinada com o poder de processamento oferecido pelos computadores.

Segundo Allison (2010), a Análise de Sobrevivência inclui uma classe de métodos estatísticos para empregabilidade em ocorrências que utilizam a observação do tempo e eventos como fatores de representação de dados. Este modelo, muito utilizado em pesquisas médicas, envolve variáveis associadas com o tempo, avaliando as observações até um evento de interesse, considerado o tempo de falha (Miller Jr, 2011).

O uso de Análise de Sobrevivência é indicado para diversos tipos diferentes de eventos associados a falha, como estudos de acompanhamento de doenças, falhas em linhas de produção, acidentes, garantias no fornecimento de produtos e até divórcios (Allison, 2010). Além disso, o modelo se destaca por incluir em sua análise os dados censurados, que consiste em observações incompletas ou parciais para evitar conclusões viciadas.

Em Colosimo and Giolo (2014), os autores citam que a Análise de Sobrevivência refere-se basicamente a situações médicas envolvendo dados censurados. Entretanto, condições similares ocorrem em outras áreas em que se usam as mesmas técnicas para analisar os dados em um ambiente real. Na Engenharia é aplicada em estudos em que produtos ou componentes são colocados sob teste para que se possa estimar características relacionadas aos seus tempos de vida, tais como tempo médio ou a probabilidade de um produto durar mais do que um tempo t (Meeker and Escobar, 2014; Nelson, 1990).

Por meio de um estimador, uma função de sobrevivência deve ser definida para identificar a probabilidade de uma observação não falhar até um tempo determinado t_{falha}. Em Kishore et al. (2010), os autores apresentam o estimador de *Kaplan-Meier* para produzir um gráfico que envolve o cálculo de probabilidades de ocorrência de um evento, em uma linha do tempo, conhecida como curva de sobrevivência (Efron, 1988; Meeker and Escobar, 2014).

Exemplos de gráficos que utilizam este estimador são apresentados na Figura 2.13, que mostram as probabilidades utilizando a unidade de medida em horas para representar o tempo de sobrevivência de recursos transientes em computação em nuvem.

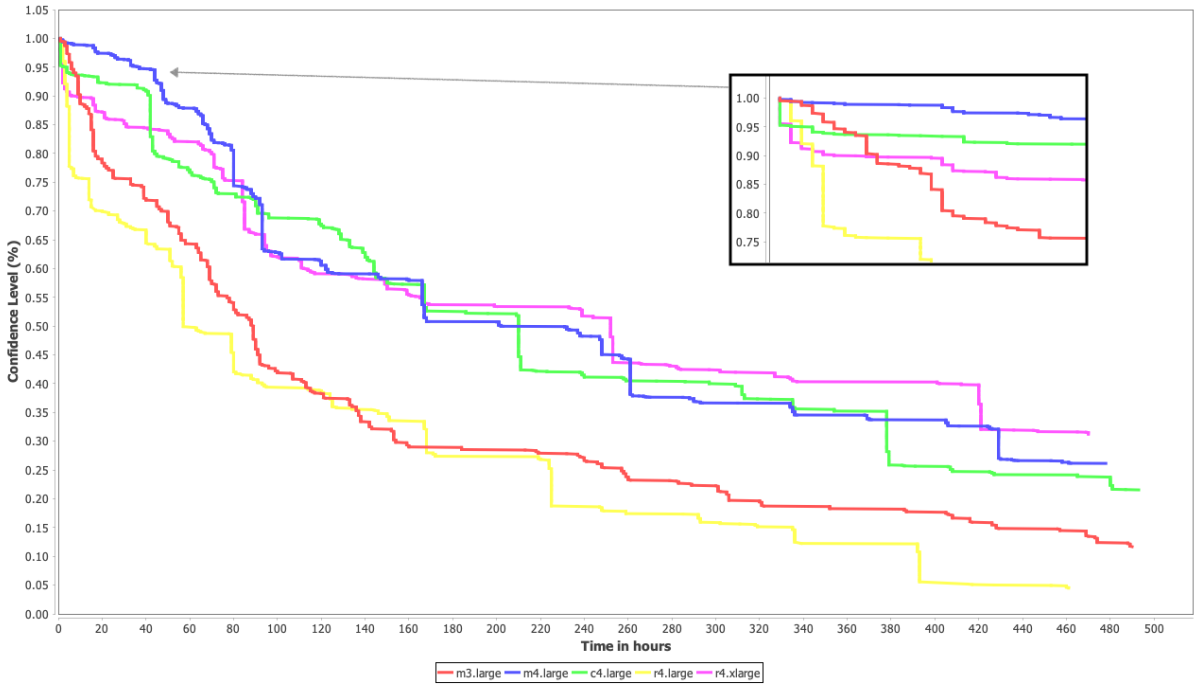


Figura 2.13: Exemplos de curvas de sobrevivências em instâncias spots.

Assim, observando a Figura 2.13, gerada a partir dos casos semelhantes de cinco instâncias spots, durante o período de abril de 2017 a fevereiro de 2019, percebe-se que cada instância tem uma curva distinta de sobrevivência, muitas vezes representando cenários favoráveis em um mesmo nível de confiança. Considerando a probabilidade de sucesso de 95% (i.e. 5% de probabilidade de falha) o tempo de sobrevivência de uma instância de uso geral e otimizada para memória (*m3.large*) é de aproximadamente 10 horas. Enquanto o tempo estimado de sobrevivência de uma instância otimizada para processamento (*r4.xlarge*) é consideravelmente inferior (1 hora).

O cenário de instâncias spots, apresentado na Figura 2.13, refere-se a uma quinta-feira às 2:00. Note que existe uma probabilidade aproximada de 95% de sobrevivência da instância *m4.large* de até 42 horas. Mas diminuindo o nível de confiança para 60%, o tempo de sobrevivência aumenta para aproximadamente 120 horas. Em contraste, a instância *m3.large* mostra um cenário em que a sua probabilidade de sobrevivência até 30 horas é de aproximadamente 65%.

Conforme explicitado por Kishore et al. (2010) e Cox (1984), a análise de sobrevivência considera dados censurados (i.e. observações incompletas para evitar conclusões viciadas), o que consideramos importante para o problema deste trabalho, pois casos atípicos de tempo de sobrevivência, i.e. casos que registram altos valores de tempo, podem ser ignorados no processamento do tempo de sobrevivência. Neste trabalho, a análise de sobrevivência envolve variáveis associadas com unidades de tempo (horas, minutos), observando comportamentos até a ocorrência de um evento de interesse: uma falha.

Capítulo 3

Revisão da Literatura

Este Capítulo tem o objetivo de apresentar os principais trabalhos relacionados ao problema desta pesquisa, demonstrando a interseção com a abordagem presente nesta proposta. A metodologia de pesquisa que direcionou o desenvolvimento desta seção é apresentada, envolvendo os conceitos das áreas de pesquisa relacionadas, incluindo o uso de técnicas de TF aplicadas no uso eficiente de instâncias transientes em ambientes de computação em nuvem.

Apesar de não existirem propostas semelhantes na literatura envolvendo todos os elementos desta proposta, alguns trabalhos produzem resultados que complementam e auxiliam nas definições e no desenvolvimento deste trabalho.

3.1 Metodologia Adotada

Para identificar os trabalhos mais relevantes e dar suporte na definição desta proposta, a revisão do estado da arte foi baseada na técnica de Revisão Sistemática, que propõe uma estratégia de pesquisa bem definida e documentada (Kitchenham, 2004). Considerada uma metodologia útil para alcançar qualidade nos seus resultados, a revisão sistemática auxilia a busca, a coleta e a avaliação de estudos relevantes que estejam relacionados com um tema específico.

Para isso, essa metodologia provê um fluxo em que estão definidas três etapas metodológicas, as quais seguem protocolos desenvolvidos com raciocínio, o que diferencia das revisões comumente aplicadas: (i) planejamento - em que é definido o problema a ser pesquisado, os critérios de seleção dos resultados e o protocolo de revisão sistemática que será utilizado durante a revisão bibliográfica; (ii) execução, em que é realizada uma análise da bibliografia utilizando os critérios estabelecidos no protocolo. Nessa etapa, cada resultado é minunciosamente analisado para que informações possam ser extraídas das pesquisas

para decidir se o trabalho será incluído ou excluído na bibliografia; e (iii) análise, em que os resultados são documentados e obtidas conclusões da revisão sistemática.

A Figura 3.1 apresenta o protocolo de uma revisão sistemática utilizado para o planejamento na busca pelos trabalhos (revisão bibliográfica), com o objetivo de garantir a execução com rigor de pesquisa. Considerando que o objetivo deste estudo é realizar uma caracterização dos elementos da proposta, não haverá comparação e nem será possível aplicação de meta-análise. Desta forma, pode-se definir este tipo de estudo como secundário, apesar de sistemático, como uma revisão quasi-sistemática (Pedreira et al., 2007).

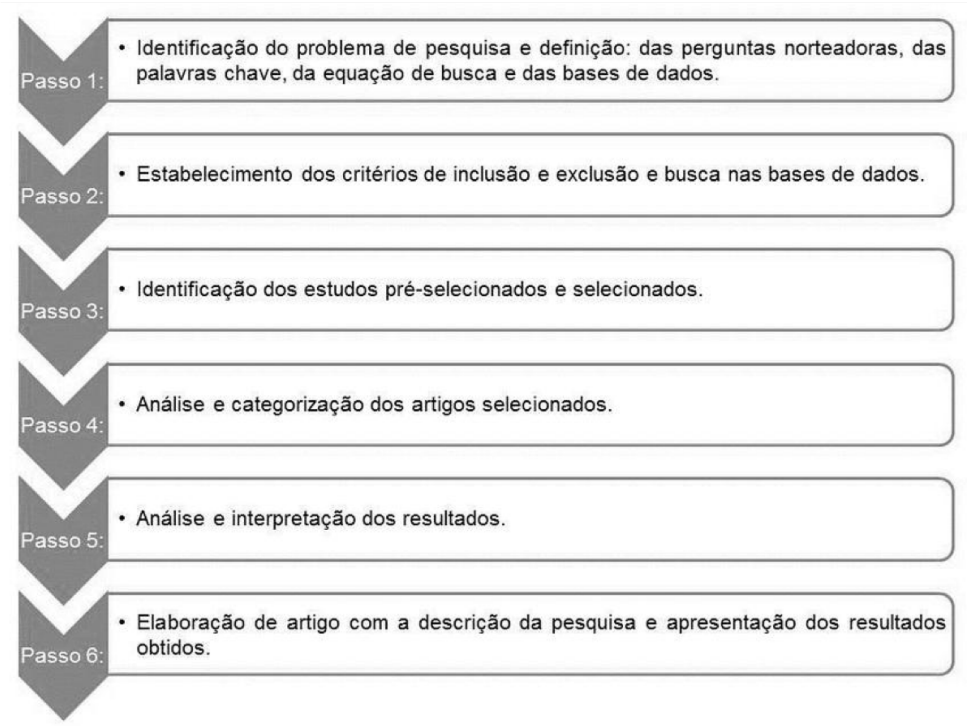


Figura 3.1: Protocolo da revisão sistemática utilizada.

A definição do protocolo definido nesta revisão sistemática inclui as definições dos objetivos, as questões de pesquisa, as fontes utilizadas, os critérios de inclusão e exclusão e as palavras-chave a serem utilizadas nas pesquisas, a saber:

- Objetivos - esta definição é composta por três etapas: (i) sistematicamente investigar as pesquisas existentes na literatura que utilizam técnicas de TF na computação em nuvem usando instâncias transientes; (ii) identificar trabalhos que possuem foco exclusivamente em instâncias transientes; e (iii) avaliar os estudos experimentais realizados envolvendo técnicas de TF no uso exclusivo de instâncias transientes;
- Questões de pesquisa - foi definida a seguinte questão principal: "*Quais são as abordagens de TF em ambientes de computação em nuvem com instâncias transientes?*";

e como questões secundárias: "*Como aplicar as técnicas de TF em instâncias transientes na computação em nuvem?*"; "*Quais as técnicas de TF são suportadas pelos instâncias transientes?*"; "*Como prover um ambiente que garanta a execução de aplicações no uso exclusivo de instâncias transientes?*"; "*Como definir dinamicamente a técnica de TF apropriada a ser utilizada na execução de uma tarefa em um servidor transiente?*"; e "*Qual o estado-da-arte em pesquisas que utilizem os servidores Spot da Amazon ou máquinas preemptáveis do Google?*";

- Fontes utilizadas - foram selecionadas bases que permitem consultas através da *Internet*, sendo reconhecidas pela riqueza de informação provida na área de computação, a saber: *IEEE Xplore Digital Library*, *Science Direct* e *ACM Digital Library*;
- Critério de inclusão - para serem incluídos na revisão bibliográfica, os resultados devem atender aos seguintes requisitos: (i) possuir foco principal em computação em nuvem; (ii) foco em técnicas de TF; e (iii) explorar ambientes de computação em nuvem com foco em instâncias transientes;
- Critério de exclusão - serão excluídos os trabalhos que atenderem aos seguintes requisitos: (i) não estar escrito em inglês; (ii) indisponibilidade total ou parcial do documento; (iii) documentos repetidos ou publicações semelhantes; (iv) apresentar técnicas de TF apenas em instâncias *on-demand*; (iv) não envolverem, de forma direta ou indireta, instâncias transientes em computação em nuvem; e (v) não publicados até o final de 2019;
- Palavras-chave - as palavras utilizadas para conduzir esta revisão sistemática foram: *cloud computing, transient server, fault tolerance, preemptible, google, spot, amazon, AWS* e *GCP*.

Esse protocolo foi definido com base no modelo proposto por Pedreira et al. (2007) e será adotada a abordagem PICO (**P**opulação, **I**ntervenção, **CO**mparação e resultados) que estrutura a questão de pesquisa em quatro elementos básicos:

i população, em que serão levados em consideração os trabalhos que, de alguma maneira, possuem relação à pesquisas que utilizam computação em nuvem, TF e sistemas distribuídos;

ii intervenção, em que devem ser considerados trabalhos que utilizam TF com foco nos instâncias transientes em computação em nuvem;

iii comparação, elemento que não será utilizado devido ao objetivo desta revisão ser apenas exploratória, que busca insumos para auxiliar nos elementos desta proposta;

iv resultados, envolvendo a utilização de instâncias transientes em ambientes de computação em nuvem, pretende-se obter os seguintes resultados: caracterização de TF,

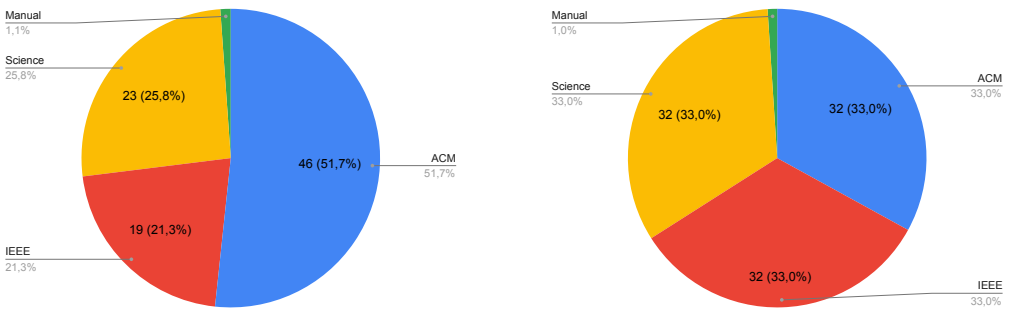
identificação de oportunidades e desafios no uso de técnicas de TF, identificação de propostas para TF, identificação de resultados obtidos por pesquisadores no uso de abordagens de TF.

Após a definição das propriedades e finalização da etapa de protocolo da revisão, a fase de execução foi iniciada. Com o objetivo de encontrar resultados mais confiáveis, esta fase utiliza a combinação das palavras-chave definidas para encontrar trabalhos mais relevantes que possuem relação com o tema desta proposta.

Aplicar variações nessas palavras e utilizar operadores lógicos permitem aumentar o nível de qualidade nos resultados. Em buscas realizadas durante o período de novembro de 2017 e Julho de 2019, a distribuição dos resultados em uma compilação inicial e final é apresentada na Figura 3.2, complementada pela Tabela 3.1, que apresenta, em sua versão final, alguns dos exemplos de estratégias de busca realizadas nas bases selecionadas e seus respectivos resultados quantitativos.

Tabela 3.1: Estratégias de busca nas bases selecionadas.

Base	**Estratégia de busca**	**2017**	**2019**
IEEE	`"Document Title": (cloud OR preemp* OR spot OR transient OR shar*) AND "Abstract": (cloud OR preemp* OR spot OR transient OR shar*) AND ("Abstract": (google OR amazon OR AWS OR GCP) OR "Document Title": (google OR amazon OR AWS OR GCP)) AND (fault* AND toleran*)`	19	32
ACM	`Title:(preempt* spot transient cloud shar*) AND Abstract:(preempt* spot transient) AND (Title:(google amazon preempt* spot transient cloud) OR Abstract:(google amazon cloud preempt* spot transient)) AND Content:(+fault +toleran*)`	46	32
Science Direct	`Title-Abstr-Key: (preemp* spot transient AWS GCP) AND Title-Abstr-Key: (google amazon cloud AWS GCP) AND All(fault* toleran*)`	23	32



(a) 2017. (b) 2019.

Figura 3.2: Resultados quantitativos das bases selecionadas em 2017 e 2019.

Observa-se que as variações das palavras-chave não incluem termos que envolve SMA e a sua inclusão tem forte reflexo nos resultados. Independente das variações utilizadas na geração da busca, a utilização de palavras relacionadas a SMA no filtro de pesquisa não apresenta resultados. A falta de resultados na busca de trabalhos relacionados a agentes pode sugerir um *gap* de pesquisa na aplicação de técnicas de TF em instâncias transientes.

Ao observar a Figura 3.2, percebe-se que houve um aumento de 10 artigos no *Science Direct* e 13 artigos no *IEEE*. Na *ACM*, houve uma diminuição de 14 artigos. É importante mencionar que os resultados atualizados incluem os 7 artigos publicados durante esta pesquisa. Os demais artigos foram analisados de acordo com o teste de relevância. Com o objetivo de uma busca exploratória, os termos utilizados devem ser detalhados e selecionados para buscar trabalhos que possuem alguma relação com esta proposta, mesmo que não envolva a busca por resultados comparativos com esta proposta.

Na etapa de execução, as pesquisas são realizadas e categorizadas em suas respectivas fontes. Para isso, títulos e resumos são analisados para identificar os trabalhos que possuem relevância com o tema da pesquisa. A escolha dos trabalhos é baseada nas definições estabelecidas nos critérios de inclusão e exclusão, sendo complementado por um teste de relevância que se faz uso dos resumos e referências dos trabalhos encontrados (Lopes et al., 2008). Este teste tem como objetivo refinar os resultados que serão acessados na íntegra, sendo composto por um conjunto de perguntas que são avaliadas e produzem respostas afirmativas ou negativas.

Para a seleção dos trabalhos, um teste de relevância foi realizado, que valida elementos como: a relevância com o tema, que relaciona o tema da publicação com este trabalho; período de publicação, limitando a publicações entre 2011 a 2019; idioma em inglês; o foco em instâncias transientes na computação em nuvem; e apresentação do problema que está sendo investigado.

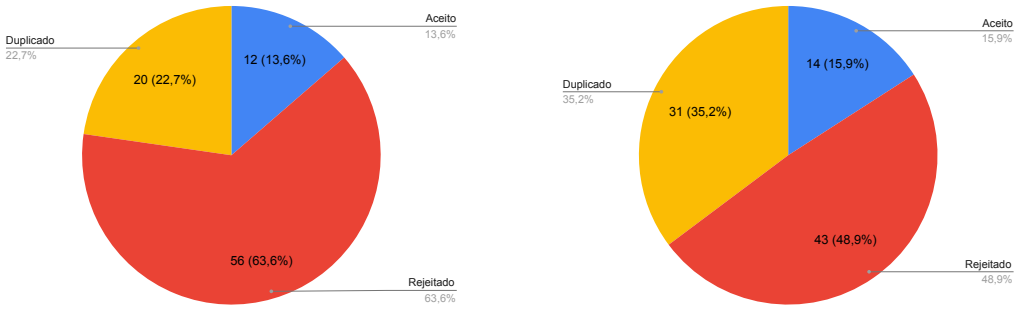
A seleção dos trabalhos é feita durante a etapa de execução, em que os resultados são obtidos, identificados e classificados de acordo com as seguintes propriedades: seleção, que define o *status* do trabalho entre aceito, rejeitado ou duplicado; e a pontuação, que indica a pontuação dada ao trabalho.

Os gráficos indicativos das seleções de leitura nas etapas inicial e final da revisão são apresentados na Figura 3.3.

A Figura 3.3 (a) indica que existe uma quantidade considerável de trabalhos duplicados (20) e rejeitados (56) por não atenderem aos critérios definidos durante o processo de protocolo ou por terem uma relação direta com trabalhos aceitos, e.g. trabalhos dos mesmos autores referentes a mesma pesquisa.

A quantidade de trabalhos duplicados torna-se relevante devido a utilização de plataformas distintas de indexação, que possuem uma redundância em suas publicações arma-

zenadas. Considerando que já foi realizado um processo de filtragem, na Figura 3.3 (a) observa-se que existem poucos trabalhos que possuem relação direta com o tema abordado nesta proposta (11). Na Figura 3.3 (b), o comportamento é semelhante a Figura 3.3 (a), porém como inclui os resultados prévios, a quantidade de duplicados aumenta (31).



(a) Resultado em 2017.

(b) Resultado em 2019.

Figura 3.3: Gráfico indicativo de seleção dos resultados.

Após a etapa que avalia a relevância dos trabalhos, inicia-se a fase de aquisição dos artigos. Os artigos devem ser acessados na íntegra para que seja possível acessar o seu conteúdo e analisar a sua proposta. Classificados por tipo de publicação, o resultado final da revisão sistemática é apresentado na Tabela 3.2.

Tabela 3.2: Teste de relevância.

Revisão sistemática em números:	
Tipo	**Quantidade**
Conferências	8
Periódicos	5
Teses	1
Total	**14**

Vale ressaltar que foram incluídos dois trabalhos manuais, que não foram resultados das consultas nas bases selecionadas. Sobretudo os resultados atendem aos requisitos definidos na revisão sistemática, envolvendo estudos que utilizam, de alguma maneira, instâncias transientes na computação em nuvem. Sobretudo, apenas os servidores Spot, do provedor Amazon, são utilizados nos resultados apresentados.

3.2 Instâncias Transientes

A partir dos resultados alcançados pela revisão sistemática, observa-se que os estudos envolvem estratégias para redução de custos (Binnig et al., 2015; Buyya et al., 2011; He et al., 2015; Jangjaimon and Tzeng, 2015; Li et al., 2015; Subramanya et al., 2015;

Voorsluys and Buyya, 2012; Yi et al., 2012), provisionamento de recursos (Lu et al., 2013; Voorsluys and Buyya, 2012) e técnicas de TF (Binnig et al., 2015; He et al., 2015; Jangjaimon and Tzeng, 2015; Lee and Son, 2017; Poola et al., 2016; Sharma et al., 2017; Subramanya et al., 2015; Voorsluys, 2014; Yi et al., 2012; Zhou et al., 2017). Observa-se que existem trabalhos que estão inseridos em mais de uma categoria. Isso ocorre quando os trabalhos exploram mais de uma abordagem em seus objetivos

Por meio da categorização dos trabalhos, percebe-se que não foi encontrado nenhum trabalho que envolva as instâncias transientes do Google. Um fator que influencia fortemente a concentração dos estudos em instâncias Spot é a transparência na publicação do histórico dos preços praticados, o que não acontece no serviço de instâncias transientes fornecidas pelo provedor Google.

Uma taxonomia mais detalhada é apresentada na Figura 3.4, que mostra os principais trabalhos envolvendo TF em instâncias transientes na computação em nuvem, e que possuem relevância com o tema deste trabalho. Será utilizada esta taxonomia para apresentar os trabalhos correlatos, enumerando as principais pesquisas encontradas na literatura que abordam a utilização de instâncias transientes. Além disso, apesar de não apresentarem resultados que utilizam exclusivamente SMA em instâncias transientes, são apresentados os trabalhos que envolvem SMA para monitoramento e aplicação das técnicas de TF.

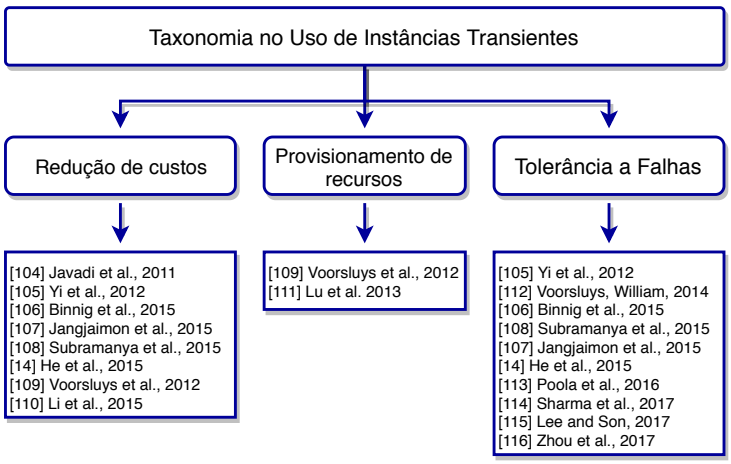


Figura 3.4: Categorização dos trabalhos encontrados na RS.

Os trabalhos estão organizados em 4 seções consecutivas. A Seção 3.3 – redução de custos, que envolvem trabalhos em que o principal foco é a estratégia de utilização das instâncias transientes, muitas vezes relacionadas aos lances adequados para atender ao tempo necessário para execução de aplicações ou utilização de estratégias que envolvem os lances e preços das instâncias spots. A Seção 3.4 – provisionamento de recursos, em que estão listados os trabalhos que se preocupam com a disponibilização de recursos para a execução de aplicações, principalmente quando há uma necessidade de incrementar os

recursos em tempo de execução. A Seção 3.5 – TF, que envolvem trabalhos que realizam experimentos em diversas técnicas de TF para identificar a menos custosa de maneira estática. A Seção 3.6 – a abordagem baseada em agentes, em que são elencados os principais trabalhos encontrados na literatura que envolvem agentes para tratamento de falhas.

Tabela 3.3: Tabela comparativa dos trabalhos relacionados, comparados aos elementos deste trabalho, envolvendo o uso de instâncias transientes.

Referência	SMA	Servidor Transiente	Geração	Transparência para o usuário	Nível TF	TF dinâmica	CBR	*Retry*	*Checkpoint/Restore*	Replicação	Validação
Yi et al. (2012)	–	✓	O	–	S	–	–	–	✓	–	S
Lu et al. (2013)	–	✓*	O	✓	S	–	–	✓	–	–	R
Voorsluys (2014)	–	✓	O	–	S	–	–	✓	✓	✓	S
Binnig et al. (2015)	–	✓	N	–	A	–	–	✓	–	–	R
He et al. (2015)	–	✓*	N	–	A	–	–	–	✓	–	P
Li et al. (2015)	–	✓	O	–	–	–	–	–	–	–	–
Subramanya et al. (2015)	–	✓*	N	✓	A	–	–	–	✓	✓	P
Jangjaimon and Tzeng (2015)	–	✓	N	–	A	–	–	–	✓	–	P
Poola et al. (2016)	–	✓	N	–	A	–	–	✓	–	✓	S
Sharma et al. (2017)	–	✓*	N	✓	S	–	–	–	✓	–	R
Lee and Son (2017)	–	✓	O	✓	A	–	–	–	✓	–	–
Zhou et al. (2017)	–	✓	N	✓	A	–	–	–	✓	–	–
[*]	✓	✓	N	✓	A	✓	✓	✓	✓	✓	S+R

Legenda:
[*] Representa esta proposta
* Indica que o trabalho envolve outros tipos de servidores, e.g. *on-demand*
Geração de tipos de servidores **O**: Antigo – **N**: Novo
Nível de TF **S**: Sistema – **A**: Aplicação
Abordagem para validação **S**: Simulação – **P**: Protótipo – **R**: Caso real

Vale ressaltar que outros trabalhos não encontrados durante a revisão sistemática foram incluídos devido a descoberta na leitura dos trabalhos selecionados. Além disso, existem trabalhos que podem estar inseridos em mais de uma categorização devido a sua natureza, e.g. um trabalho que aplica a tolerância a falhas para a redução dos custos, tanto na perspectiva do usuário quanto do provedor de computação em nuvem.

3.3 Redução de Custos

Como mencionado na Seção 2.1, instâncias transientes permitem que usuários possam reduzir os custos para execução de aplicações em grande escala. Para isso, as aplicações devem estar preparadas para cenários em que haja uma degradação de performance, demoras não esperadas e perda do servidor de execução.

Em trabalhos que envolvem o uso de instâncias Spot da Amazon, estratégias que definem o valor de um lance possibilitam a diminuição de revogações. Inseridos nesta categoria, foram identificados os seguintes trabalhos:

- Jangjaimon and Tzeng (2015) – os autores propõem um modelo para redução de custos em instâncias transientes ao apresentar uma adaptação de *checkpoint* incremental específica para aplicações *multi-threads*. Esse modelo, que trata de falhas provocadas por eventos de revogação das instâncias, baseia-se no modelo de Markov para predição da ocorrência destes eventos.

  Esse modelo foi adotado por permitir previsões rápidas com altos níveis de acurácia. Para isso, é necessário que o usuário disponibilize o tempo de execução do programa submetido e o valor do seu lance. Essas entradas são complementadas pelos dados históricos com informações dos preços praticados durante 112 dias. Esses dados de entrada são utilizados para uma função que aponta a probabilidade de revogação da instância transiente durante a execução do programa.

  Diante disso, uma estratégia de *checkpoint* é definida, que utiliza uma abordagem multinível para diminuir o *overhead*, alcançando resultados com diminuição dos custos e melhorias na complexidade de tempo e de espaço em comparação ao seu trabalho anterior (Moody et al., 2010).

  Considerações: É possível estabelecer uma relação desse trabalho com esta proposta. A técnica de *checkpoint* poderá ser útil por permitir que, durante a execução de uma aplicação, os seus estados possam ser armazenados.

  Isso permite, na ocorrência de uma falha, a reexecução do sistema a partir de um ponto exato da sua execução. Além disso, esse trabalho apresenta uma estratégia de *checkpoint* incremental, que consiste em armazenar apenas os dados modificados, evitando o armazenamento redundante de todos os dados da sua execução. Isso poderá ser útil para diminuir o *overhead* da aplicação quando for necessário salvar o seu estado.

- Buyya et al. (2011) – os autores descrevem um modelo estatístico que analisa dados históricos dos preços praticados em diferentes tipos de instâncias Spot durante o período de um ano. Com essa análise, esse trabalho determina as variações e o tempo de permanência dos preços, categorizando-os em uma relação de hora por dia e dias por semana.

  Como resultado desse trabalho, ao utilizar um modelo estatístico de distribuição Gaussiana, essa análise permite identificar os melhores dias e horários da semana

para alocação dos recursos necessários e agendamento para a execução de uma aplicação específica.

Considerações: Vale considerar a categorização dos preços praticados em horas e dias da semana, pois ele pode ser útil para gerar modelos na análise não apenas do melhor dia e hora para execução, mas para analisar o cenário no momento da submissão da aplicação.

- Yi et al. (2012) – os autores utilizam o histórico de preços semelhante a Buyya et al. (2011) para avaliar métodos de predição no dinamismo dos preços praticados nas instâncias Spot, resultando em uma compilação que mostra a disponibilidade dessas instâncias nos horários do dia.

  Nesse trabalho, os autores avaliam a utilização de dois mecanismos de TF: *checkpoint* e migração. Os seguintes esquemas de *checkpoint* são apresentados: *NONE*, que não salva o estado da aplicação em nenhum momento durante a sua execução; *HOUR*, em que são realizados *checkpoints* periódicos a cada hora, desde o início da execução; *EDGE* que realiza o *checkpoint* a cada aumento no valor do preço da instância Spot; *BASIC*, que analisa e define, a cada dez minutos, o intervalo de *checkpoint* a partir de uma função que avalia o tempo de recuperação em caso de falhas; e *PRICE-BASED* que semelhante ao esquema *BASIC*, avalia e define o intervalo de *checkpoint* a cada dez minutos, utilizando como referência o valor atualizado da instância Spot.

  Além de avaliar a utilização de *checkpoint* em diversos cenários, os autores propõem a utilização de migração como mecanismo de TF. Esse mecanismo permite que uma execução possa ser transferida para outras instâncias e continuar a sua execução. Os autores utilizam heurísticas que tomam decisões baseadas em dois fatores: como realizar a migração e qual o novo lance será dado para a(s) nova(s) instância(s), baseado nos valores das instâncias disponíveis.

  Considerações: Vale considerar a utilização de migração, que implicitamente aplica *checkpoint*, para permitir copiar o estado da execução para outras instâncias. Além disso, os esquemas de *checkpoint* apresentados são específicos para instâncias Spot, sendo úteis para serem considerados na decisão da estratégia de *checkpoint* para execução de tarefas.

- Li et al. (2015) – os autores exploram instâncias Spot propondo um mecanismo de análise e definição do lance de forma que garanta o tempo necessário que o usuário deseja para executar a sua aplicação, evitando a falha de revogação. Usando um controle de *feedback*, que analisa falhas passadas, esse trabalho utiliza lógicas *fuzzy* e intervenção humana para definição dos novos valores de lance.

Considerações: Associado a um fator de temporalidade, vale considerar a análise de falhas anteriores para refinar o raciocínio da estratégia de lance. Esse trabalho avalia cinco estratégias de lances (mínimo registrado, média, maior registrado, atual e *on-demand*) que podem ser exploradas nesse trabalho. Além disso, o histórico de preços pode ser explorado para permitir identificar variações entre os valores praticados, auxiliando na identificação de instâncias que estão com uma quantidade considerável de alterações recentes.

- He et al. (2015) – o trabalho descreve uma abordagem diferente para o problema de disponibilização de serviços *online*, explorando uma nova possibilidade de utilizar instâncias Spot para publicar serviços *online* com altos índices de disponibilidade e reduzir os seus custos. Para isso, os autores propõem um modelo de agendamento que utiliza uma abordagem híbrida entre instâncias Spot e *on-demand* para prover um mecanismo de migração e garantir a disponibilidade do serviço.

  Nesse modelo os autores apresentam um algoritmo que utiliza as seguintes estratégias: reativa, que age quando ocorre uma interrupção na disponibilização da instância transiente; e proativa, que decide a ação baseado no histórico dos preços praticados nas instâncias Spot.

  Como resultado desse trabalho, os autores conseguem mostrar que é possível alcançar altos níveis de disponibilidade nos serviços *online* e reduzir os custos a utilização de instâncias *on-demand* junto com uma estratégia proativa.

  Considerações: Esse trabalho é relevante para esta proposta, pois apresenta uma possibilidade de utilização das instâncias *on-demand* para suportar a execução de uma aplicação. Em um cenário em que o valor de leilão nas instâncias Spot se equiparam e até ultrapassam o valor de instâncias *on-demand*, utilizar a garantia de disponibilidade oferecidas nas instâncias não revogáveis torna-se uma estratégia mais interessante para execução de uma aplicação.

3.4 Provisionamento de Recursos

No provisionamento de recursos, os trabalhos encontrados concentram esforços em arquiteturas que exploram mecanismos para aumentar o nível de disponibilidade de recursos para a execução de uma aplicação, a saber:

- Voorsluys and Buyya (2012) – os autores apresentam uma estratégia para alocação de recursos, propondo uma política de provisionamento de recursos que utiliza mecanismos para estimar o tempo de execução da aplicação e o valor de leilão das instâncias Spot.

Esse trabalho complementa um estudo anterior, que demonstrava a viabilidade de alocação dinâmica em *clusters* virtuais compostas por instâncias Spot, que utilizavam *checkpoint* para garantia da execução. Um mecanismo para definição do lance dado pelo usuário é apresentado, que busca evitar a falha na revogação da instância. O autor assume que, definido um lance que está sempre inserido no valor praticado pelo leilão, a instância terá uma disponibilidade de 100%. Os autores avaliam cinco estratégias para definição do preço: valor mínimo registrado, média histórica, valor da instância *on-demand*, valor máximo registrado e o valor de leilão atual. Em seu experimento, esse trabalho compara as cinco estratégias para definição do preço com as três técnicas de TF. Como resultado, os autores mostram que é possível prever a falha de uma instância, permitindo a migração de sua execução para outra nova instância (novo lance) oferecendo maior disponibilidade na utilização.

Considerações: Resultados apresentados nesse trabalho indicam que a migração é uma alternativa a ser levada em consideração no modelo multiestratégico proposto. Os autores indicaram que a sua utilização não gera *overhead* significativo, porém necessita de alocação de mais de uma máquina para que possa realizar a migração. Além disso, gráficos mostram que as melhores estratégias para definir o valor do lance envolvem a utilização do mínimo registrado e do valor atual. Por fim, as regras na definição da estratégia de TF dá indícios de que é necessário assumir valores e prioridades iniciais, possibilitando a modificação dessas regras para se adequar a realidade de cada execução.

- Lu et al. (2013) – os autores apresentam uma proposta que também utiliza o histórico dos preços para redução de custo e provisionamento de recursos para execução de aplicações em larga escala. Apesar de ter muita semelhança com outros trabalhos, essa proposta apresenta dois mecanismos de TF: *backup*, que realiza cópias das instâncias; e provisionamento de recursos dinâmicos, que utiliza esses backups para serem restaurados em instâncias *on-demand*.

  No experimento, os autores migraram uma aplicação de nuvem privada para a Amazon, utilizando mil instâncias Spot para avaliar a sua escalabilidade. Os resultados mostram que, apesar do provisionamento de recursos híbridos permitir uma redução de 23% do seu custo, o tempo total de execução aumentou em 5.3% devido à política de TF utilizar o *backup*.

  Considerações: Apesar de ter tido um aumento no tempo total de execução, vale considerar que a utilização de *backup* possa ser válida quando a quantidade de dados a ser armazenado não influencie consideravelmente o tempo de execução. Além disso, resultados mostram que utilizar apenas instâncias Spot permite uma redução de custo maior do que o modelo híbrido apresentado.

3.5 Tolerância a Falhas

Em relação ao tratamento para evitar reflexos quando uma falha ocorre, trabalhos aplicam técnicas de TF e incluem métodos que buscam minimizar os impactos na aplicação. Nesse direcionamento, os trabalhos realizam experimentos em diversas técnicas de TF para identificar a menos custosa em diversos cenários de execução.

Nessa abordagem, destacam-se os seguintes trabalhos:

- Voorsluys (2014) – como resultado e complemento do estudo apresentado em Voorsluys and Buyya (2012), o autor apresenta um estudo que propõe um gerenciamento de recursos que utiliza instâncias Spot para aplicar políticas de agendamento para execução de tarefas. Nesse trabalho, o autor explora o histórico da variação de preços para montar um modelo que contenha os padrões adotados, possibilitando a identificação do melhor momento para executar a tarefa.

  A solução apresentada utiliza três técnicas de TF: *checkpoint*, que salva o estado completo da instância; replicação, que possibilita a execução paralela de uma aplicação em instâncias distintas; e migração, que permite a movimentação da execução em instâncias distintas. A decisão de qual técnica será utilizada atende às seguintes regras: usar replicação se o tempo de execução exceder uma hora; usar *checkpoint* a cada hora; e aplicar a migração quando houver um aumento no valor de leilão.

  Além disso, um modelo de predição definido para estimar o tempo de execução de uma tarefa é utilizado. Para esta estimativa, o autor utiliza os seguintes métodos: "Tempo atual", que assume um valor aleatório, sem se basear em dados históricos; "Tempo atual com erro", que também assume um valor, adicionando um percentual randômico entre 0 e 10%; "Definido pelo usuário", que assume o valor do tempo baseado nos dados fornecidos pelo usuário; "Fração da definição do usuário", que assume que o usuário superestima o tempo, definindo uma fração de $\frac{1}{3}$ em relação ao valor especificado pelo usuário; e "Média recente", que consiste na utilização do tempo de execução das tarefas executadas anteriormente pelo mesmo usuário.

  A política de agendamento proposta confia na estimativa de execução, que consiste em uma média das últimas duas execuções, para decidir quando e qual o tipo de instância Spot é mais adequado para a execução da tarefa.

  Por fim, o autor propõe como trabalhos futuros o refinamento nas técnicas de TF para reduzir o *overhead* e avaliá-las em aplicações de diversas naturezas. Além disso, propõe uma criação de políticas que possam utilizar também instâncias *on-demand* e avaliar a utilização de outros modelos de disponibilização de instâncias transientes, como a plataforma de computação em nuvem do Google.

Considerações: Apesar de não produzir uma técnica precisa de predição, esse trabalho apresenta regras que podem ser úteis para o módulo de predição dessa proposta, que pode ou não utilizar a entrada do usuário como referência do tempo de execução.

Além disso, esse trabalho utiliza o *live-migration*, que permite uma migração de uma instância sem que haja perdas consideráveis no tempo total de execução. Isso corrobora com a ideia de também utilizar a migração como técnica de TF.

- Poola et al. (2016) – os autores descrevem algoritmos que utilizam a replicação como principal mecanismo para garantir a execução de uma tarefa, sendo utilizado para re-submissão ou duplicação de tarefas. Na re-submissão de tarefas, o modelo se comporta da seguinte maneira: sempre uma execução será replicada em duas instâncias e, na ocorrência de uma falha, outra instância é alocada e a tarefa é submetida a esta nova instância. Na duplicação de tarefas, o modelo se comporta como uma replicação normal, que executa a tarefa em mais de uma instância. Os autores definem que as tarefas possuem um *deadline* que quantifica o tempo de execução da tarefa.

  Como experimento, esses trabalho apresentam heurísticas utilizadas no algoritmo que são aplicadas a alguns cenários para o mapeamento da tarefa: (i) utilizar uma instância Spot já alocada; (ii) adquirir uma nova instância Spot; (iii) utilizando uma instância *on-demand*; e (iv) duplicação de tarefas, que considera tarefas críticas as que estão sendo executadas e que possuem um nível de *deadline* baixo.

  O resultado desse trabalho mostra que, na presença de tarefas com altos níveis de criticidade, a replicação com re-submissão na ocorrência de falhas apresenta resultados mais favoráveis em relação à replicação tradicional.

  Considerações: Essa abordagem de replicação com re-submissão poderá ser útil para este trabalho, que pretende usar a técnica de *retry*. Diferente do *retry*, essa re-submissão ocorre em instâncias replicadas sem envolver replicação ativa. Além disso, os autores propõem a utilização de múltiplos *datacenters* que oferecem instâncias transientes, considerando o custo na transmissão de dados.

- Subramanya et al. (2015) – apresenta um *framework* para execução em instâncias *spots* que utiliza as técnicas de *checkpoint/restore*, migração reativa e replicação. A proposta trata a TF em nível de aplicação de maneira não dinâmica, i.e. não envolve outra técnica de TF durante a execução de uma aplicação.

  Considerações: Essa abordagem utiliza um modelo híbrido de utilização de VMs, i.e. o trabalho não utiliza exclusivamente instâncias transientes. Instâncias *on-demand* são utilizadas quando uma falha ocorre, aumentando o custo do usuário.

Além disso, as ações de recuperação de uma falha ocorrem quando uma falha é identificada, e.g. migrar a execução para outra instância *on-demand*, mas nem sempre é possível devido ao tempo limite de revogação da instância, principalmente quando uma execução utiliza muitos recursos de memória, exigindo um tempo maior para a transferência dos dados para uma nova instância. Por fim, o *framework* não está mais disponível, impossibilitando a comparação com as abordagens da literatura.

- Binnig et al. (2015) – apresenta uma proposta de arquitetura para montar um *cluster* utilizando instâncias Spot. Nesse trabalho, os autores descrevem uma técnica de TF que é baseada no custo na alocação da instância. Utilizando o MTBF (*Mean Time Between Failures*), que indica a média de tempo que uma instância está disponível até a sua revogação, a estratégia baseada no custo utiliza os parâmetros inseridos pelo usuário, e.g. o tempo de execução, para criar um conjunto de planos de execuções e minimizar a possibilidade de falhas. Nesse modelo, quando uma falha ocorre, apenas os sub-planos são reiniciados, portanto, apenas uma técnica de TF é utilizada.

  Para definir o plano mais adequado para a execução de tarefas, vários planos são criados e avaliados através de uma função que busca o melhor plano que possui o menor tempo de execução.

  Considerações: Ressalta-se a utilidade do trabalho, pois a criação de planos é uma abordagem válida para inserir no modelo proposto, podendo ser utilizado para avaliar instâncias ótimas para execução em diferentes provedores.

- Lee and Son (2017) – os autores exploram instâncias *spots* com recursos de GPU para otimização na execução de *deep learning workloads*. Nesse trabalho, os autores utilizam como técnica de TF apenas a migração de tarefas. A migração é iniciada no momento em que a instância é revogada do usuário. Um protótipo implementado na Amazon AWS é citado no trabalho, porém o experimento não inclui o uso desse protótipo.

  Considerações: A estratégia de realizar a migração no momento em que a instância é sinalizada que será revogada é arriscada, pois quando o tipo de aplicação que está sendo executada for de uso intensivo de memória, a quantidade de dados que deverá ser transmitida é alta. O provedor de nuvem computacional da Amazon disponibiliza apenas 2 minutos após sinalizar que a instância será revogada. Apesar dos autores afirmarem que esse tempo é suficiente, isso é correto apenas se a aplicação não utilizar muitos recursos de CPU e memória da instância, caso contrário, o tempo pode não ser suficiente.

- Sharma et al. (2017) – os autores propõem uma plataforma que provê a ilusão de um IaaS com instâncias não revogáveis e sempre ativas, utilizando instância Spot. Esse trabalho tem como objetivo a criação de um *middleware* para a revenda de recursos transientes utilizando instâncias *on-demand* e Spot para prover altos níveis de disponibilidade próximos ao estabelecidos em instâncias não revogáveis (*on-demand*).

  Para isso, os autores definiram as seguintes políticas: (i) estratégia de lances, que define valores para serem utilizados no leilão; (ii) seleção da instância, avaliando as instâncias disponíveis e escolhendo qual é a mais indicada para a execução da tarefa; (iii) *backup*, que periodicamente armazena instâncias; e (iv) *hot-spares*, que são instâncias *on-demand* sempre disponíveis para serem usadas na reposição de instâncias que estão prestes a serem revogadas.

  Como resultado, esse trabalho mostra que é possível criar um ambiente de alta disponibilidade utilizando instâncias transientes. A plataforma proposta permite uma relação transparente para o usuário, criando a ilusão de utilização de instâncias não-revogáveis.

  Considerações: Vale considerar a utilização de *hot-spares* no modelo para agilizar o processo de migração quando ocorre uma revogação no acesso a instância. Apesar de gerar um custo adicional, dependendo da tarefa em execução, uma instância *on-demand* poderá ser utilizada até que uma nova instância seja disponibilizada. Essa abordagem permite que uma aplicação possa continuar a sua execução ao substituir, usando replicação ou não, as instâncias que foram revogadas por novas instâncias.

- Zhou et al. (2017) – os autores exploram as instâncias Spot que disponibilizam recursos de GPU para execução de aplicações de alto desempenho. Nesse trabalho foi criado um *cluster* de execução na mesma zona para diminuir o *gap* na comunicação entre as instâncias. Esse trabalho utiliza o *checkpoint/restore* como técnica de TF com intervalos de 10 e 30 minutos.

  Considerações: Apesar dos autores optarem por um baixo intervalo de *checkpoint*, o que compromete o tempo total de execuções longas, para a execução de processamento em GPUs, eles demonstram que o intervalo de 10 minutos não afeta na performance de GPU. Sobretudo, nesse trabalho, os autores demonstram a importância de avaliar o preço de uma GPU, comparada a instâncias convencionais, quando a natureza da aplicação é de execução intensa de CPU, i.e. não necessita de alto tráfego do estado de memória.

3.6 Abordagem Baseada em Agentes

Utilizando um modelo baseado em agentes, a arquitetura proposta neste trabalho consiste na inclusão de agentes distribuídos para monitoramento e gerenciamento de execuções. Nesta abordagem, apesar de não existirem trabalhos relacionados a computação em nuvem, destacam-se os seguintes trabalhos:

- Souchon et al. (2003), Xu and Deters (2004) e Platon et al. (2008) – respectivamente, os autores apresentam o conceito de agentes de exceção em SMA, apresentando uma arquitetura em que há a presença de agentes de monitoramento que possuem comportamentos exclusivos para tratamento de eventos. Esses agentes manipulam as informações contidas nas exceções, transformando em eventos e propagando na aplicação.

  Considerações: Essa abordagem poderá ser útil nesta proposta por proporcionar técnicas de propagação distribuída entre os agentes em forma de eventos. Utilizar eventos para distribuição de informações é mais interessante e confiável do que utilizar mensagens. Dependendo da falha, um agente pode ficar impossibilitado de receber mensagens, e isso pode comprometer o tráfego de envio e de recebimento de mensagens importantes dentro de uma arquitetura.

- Hägg (1997) – o autor apresenta o uso de sentinelas para monitoramento dos recursos em um sistema baseado em agentes. Utilizando um esquema de endereçamento semântico e quebra de encapsulamento, os agentes sentinelas podem monitorar não apenas se um agente está ativo, mas também observar as suas comunicações e comportamentos, realizando intervenções quando necessário. Isso é útil em um cenário que demanda uma redundância no monitoramento das execuções.

  Considerações: O conceito de sentinelas é uma técnica interessante e aplicável nesta proposta. Vale considerar o estudo das técnicas utilizadas nesse trabalho para substituição dos agentes quando houver uma falha em tempo de execução. Essa abordagem permite que uma aplicação possa continuar a sua execução ao substituir, usando replicação ou não, os agentes com falhas por novos agentes.

- De la Prieta et al. (2014); Siddiqui et al. (2012) – os autores utilizam nuvens públicas e adotam o uso de agentes reativos para provisionar recursos de acordo com a utilização durante a execução da aplicação. Nesse trabalho, os autores não utilizam um modelo de raciocínio nos agentes para a tomada de decisões. Os agentes monitoram o uso do recurso, sem dados anteriores, assumindo comportamentos para provisionar recursos elasticamente de forma vertical – funcionalidade já oferecida pelo provedor de computação em nuvem – quando há a necessidade de estender sua

capacidade, i.e. quando os recursos disponíveis já estão sendo utilizados em sua totalidade.

Considerações: A abordagem de monitoramento dos recursos em tempo real torna-se interessante para poder observar o comportamento na execução da aplicação. Além disso, observar que a execução está consumindo todos os recursos pode ser um indicativo que a instância utilizada pode ser substituída por outra, desde que haja mais recursos.

Para buscar garantia nas execuções de sistemas em instâncias transientes, trabalhos utilizam as técnicas de TF das mais variadas formas: Sharma et al. (2017); Zhou et al. (2017) exploram o uso de *checkpoint* na busca da garantia de finalização da execução; He et al. (2015); Voorsluys (2014) utilizam a técnica de *checkpoint* para salvar os estados da execução e facilitar a migração de máquinas virtuais; em Gong et al. (2015); Subramanya et al. (2015), os autores combinam o uso de *checkpoint* com replicação para reduzir a taxa de falhas; e Jangjaimon and Tzeng (2015), que fazem uso de *checkpoint* para armazenar os estados da execução de maneira incremental, captando apenas as mudanças existentes entre os estados para diminuir o *overhead* da aplicação.

Além da utilização de técnicas de TF na busca da garantia de execução, os trabalhos que exploram as instâncias transientes em computação em nuvem possuem algo em comum: a utilização de instâncias Spot, da Amazon. Um fator que influencia fortemente os estudos em instâncias Spot é a publicação histórica dos preços praticados, permitindo a criação de diversos modelos de predição, o que não acontece nas máquinas transientes fornecidas pelo Google, que são denominadas máquinas virtuais preemptáveis (*preemptible VMs*).

Por meio do Google Clusted Data Cirne et al. (2012), dados das execuções de diversas tarefas são disponibilizadas através de *logs* que refletem os dados de 25 milhões de execuções em 12 mil servidores. Apesar desses dados serem ricos de informações sobre execuções, escalonamentos e carga de recursos, possibilitando o seu uso em diversas áreas de pesquisas, inclusive para a criação de um modelo de predição, o cenário de captura destas informações não era composto exclusivamente por instâncias transientes, impossibilitando o seu uso em trabalhos que envolvem as máquinas preemptáveis do GCP.

Assim, percebe-se que os trabalhos que envolvem instâncias transientes buscam a diminuição de custos, transparência e praticidades da computação em nuvem fornecida pela Amazon em suas instâncias Spot, restando poucos trabalhos que exploram a Google GCP, principalmente em se tratando das instâncias preemptáveis.

Capítulo 4

Arquitetura Resiliente Baseada em Agentes

Considerando as vantagens que a abordagem orientada a agentes oferece, este trabalho propõe uma arquitetura para provimento de um ambiente tolerante a falhas, chamada de BRA2Cloud[1] (*A **BR**and new **A**gent-based **A**rchitecture for **Cloud** computing*). Assim, uma arquitetura resiliente baseada em agentes para instâncias transientes na computação em nuvem é apresentado.

O objetivo é minimizar os impactos na execução de tarefas na presença de falhas de revogação das instâncias transientes, em que agentes escolhem a técnica de TF mais adequada considerando o momento da submissão da tarefa.

Este Capítulo apresenta a proposta de solução deste trabalho incluindo: na Seção 4.1, o modelo conceitual com as definições relacionadas aos agentes e respectiva caracterização do ambiente de computação em nuvem; na Seção 4.2, o modelo arquitetural, incluindo aspectos físicos e de software; e na Seção 4.3, o modelo de raciocínio para a definição do plano de execução é apresentado.

4.1 Modelo Conceitual

Algumas definições são necessárias para apresentar os elementos desta proposta, com foco em uma arquitetura multi-estratégica de TF baseada em agentes na computação em nuvem.

[1]Disponível em `http://bra2cloud.cic.unb.br`

4.1.1 Computação em Nuvem

Na Seção 2.1 é apresentada a arquitetura de computação em nuvem, que consiste em um ambiente em que os recursos são oferecidos em diversas classes de serviço. Um provedor de computação em nuvem provê recursos computacionais úteis (e.g., VMs), a serem utilizados através da Internet, normalmente oferecidos como *WebServices* (Buyya et al., 2009).

Definição 1 (Máquina Virtual) *Uma Máquina Virtual (VM) é um recurso oferecido pelos provedores, na classe de serviço IaaS, que possui características semelhantes a um computador, sendo representado por uma 6-upla (C_{cpu}, C_m, C_{arm}, q, v, C_{net}), onde:*

1. *C_{cpu} é o atributo que indica o poder de processamento;*
2. *C_m é o atributo que indica a capacidade de memória volátil;*
3. *C_{arm} é o atributo que indica a capacidade de armazenamento;*
4. *q é o atributo que indica o estado atual de execução;*
5. *v é o valor praticado em unidade de tempo (normalmente em intervalos fechados de horas);*
6. *C_{net} é o atributo que indica a capacidade de transferência de dados em rede.*

Considerando que as VMs são provisionadas dinamicamente e sob demanda, uma porção significativa de recursos pode se manter ociosa, como ilustrado na Figura 4.1, que ilustra o gráfico de utilização de recursos do servidor que hospeda o projeto desta pesquisa, localizado no laboratório COMNET, durante o período de 26 a 27 de fevereiro de 2019.

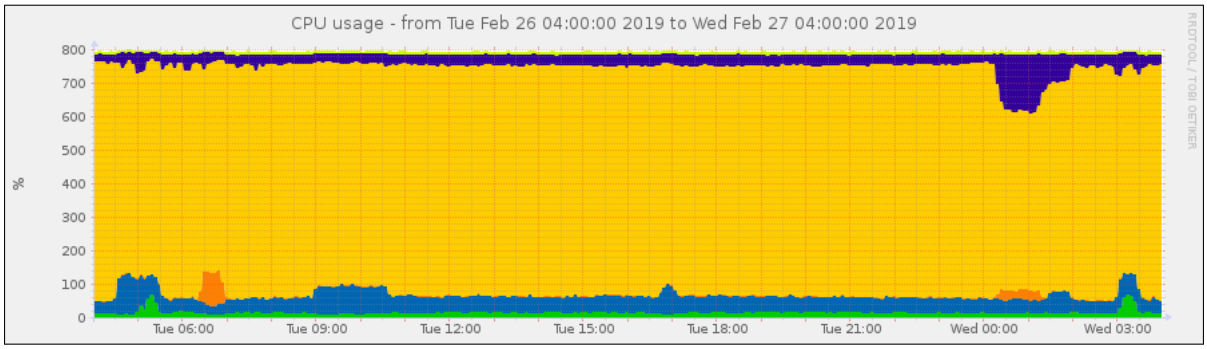


Figura 4.1: Exemplo de utilização dos recursos de uma VM.

Como observado na Figura 4.1, existe uma quantidade considerável de recursos ociosos, marcado em amarelo, presentes na linha do tempo (Eixo X).

Devido a natureza preemptiva aplicada às instâncias transientes, estas não são utilizadas para hospedar serviços que buscam alta-disponibilidade e confiabilidade, como servidores web ou qualquer outra aplicação que não possa ser interrompida por uma falha de revogação (mesmo que por um tempo mínimo). Sobretudo, aplicações BoT são adequadas para estas instâncias, desde que exista um meio de recuperação para as falhas de revogação (Irwin et al., 2017).

As instâncias transientes são oferecidas em um ambiente que provê a ilusão de disponibilidade infinita de recursos, permitindo que os usuários possam realizar requisições de diversas instâncias, respeitando os limites de cada provedor, e aguardar as suas disponibilizações.

Definição 2 (Instância Transiente) *Define-se que uma instância transiente λ como:*

1. *Uma instância, em forma de VM, fornecida por um provedor de computação em nuvem para ser utilizada pelos usuários, por meio do modelo IaaS;*
2. *Uma VM que possui um estado transiente imutável e sem garantias de disponibilidade;*
3. *Um serviço que pode ser revogado de acordo com as regras pré-estabelecidas pelo provedor de computação em nuvem;*
4. *Uma VM criada no momento da requisição do usuário;*
5. *Oferecida por um valor inferior as VMs convencionais (on-demand).*

Para representar um conjunto de definições necessárias sobre o uso de VMs, uma instância transiente é composta por uma 11-tupla (*reg, zona, tipo, gen, cpu, mem, arm, rede, estado, group-sec, preco*) conforme detalhado na Tabela 4.1.

Tabela 4.1: Definições dos elementos para utilização de uma instância transiente.

Propriedade	**Descrição**
reg	Representa a região.
zona	Representa a zona inserida na região utilizada.
tipo	É um indicador do tipo de instância utilizada.
gen	O atributo que indica a geração de seus recursos.
cpu	Indica a capacidade de processamento, incluindo quantidade de vCPU virtuais e respectiva velocidade de *clock*.
mem	Atributo que indica o recurso associado a um tamanho volátil de memória RAM, incluindo velocidade de acesso e capacidade estendida, quando aplicado.
arm	Indica a capacidade de armazenamento em disco virtual disponível para o usuário, sendo usada para armazenar arquivos e dados necessários durante o processo de execução.
rede	Representa a velocidade e a capacidade de transferência de dados em um adaptador de rede específico.
estado	Atributo que indica o estado de execução da VM, que pode ser: iniciado, executado, revogado ou parado.
group-sec	Formado por um conjunto de atributos de um grupo de segurança com regras de rede que permitem ou negam acesso externo a VM, e.g. portas liberadas e chave de segurança privada para controle de acesso.
preco	Representa o preço da instância no instante de sua utilização (por ciclos de hora), adquirido de um provedor de computação em nuvem.

Os provedores sublocam estes recursos ociosos e oferecem de maneira compartilhada para melhor aproveitamento do hardware disponível (Shastri et al., 2016). Desta forma, o

provedor aumenta seu retorno financeiro e os usuários podem melhor utilizar os recursos de hardware.

Definição 3 (Provedor de Instância Transiente) *Um provedor que oferece o serviço de instância transiente é um ambiente de computação em nuvem que oferece, através da classe de serviços IaaS, recursos ociosos em formas de serviços de VMs não confiáveis, os quais podem ser revogadas a qualquer momento, de forma imprevisível, sem permitir a intervenção do usuário.*

Definição 4 (Conjunto de Instâncias Transientes) *Um ambiente composto por λs distintos é um conjunto de instâncias transientes $\Lambda = \{ \lambda_0, \lambda_1, ..., \lambda_n \}$, representada por um conjunto finito com n instâncias transientes que serão utilizadas para executar aplicações BoTs.*

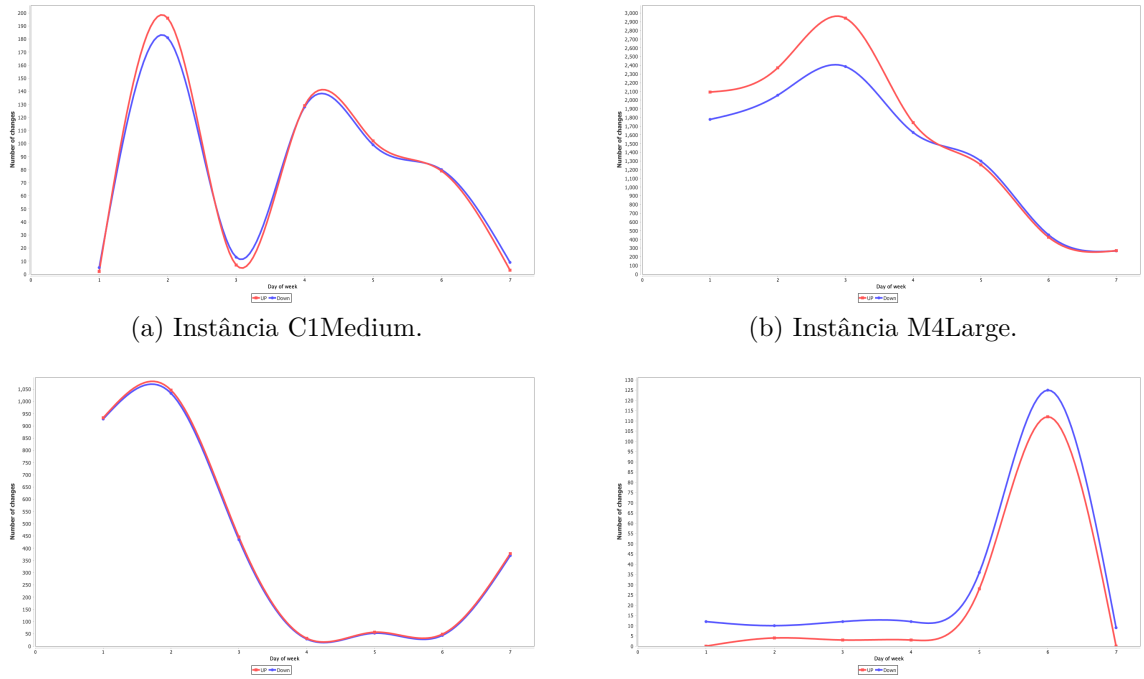
Definição 5 (Técnica de TF) *Nesta proposta, é considerada uma técnica de TF como a definição do mecanismo tolerante a falhas utilizado para a execução da aplicação, com seus respectivos parâmetros. Para a replicação, o parâmetro que define a quantidade de execuções é definido como a duplicação da execução de forma ativa. Para o checkpoint/restore, o parâmetro do intervalo (C_{int}) de checkpointing é calculado de acordo com o cenário de execução no momento da submissão da aplicação.*

4.1.2 Raciocínio dos Agentes

Apesar do valor oferecido pelas instâncias transientes ser consideravelmente inferior ao praticado pelas instâncias não-transientes, provedores de computação em nuvem utilizam políticas diferentes na definição de seus preços. Como apresentado na Seção 2.1, o provedor da Amazon define os seus preços através de uma política de mercado, que os preços variam continuamente, baseado em leis da oferta e procura.

A disponibilidade destas instâncias pode variar dependendo das suas configurações, localidade e disponibilidade. Infelizmente poucos provedores fornecem dados suficientes sobre os seus estados internos que levam a aplicação de regras no momento da revogação. Diante disso, usuários são forçados a analisarem fatores não-determinísticos na tentativa de predizer o momento da revogação de suas instâncias transientes.

Observando o histórico com as alterações de preços nas instâncias Spot, coletados por meio de pouco mais de 20 milhões de registros, durante o período de abril de 2017 e fevereiro de 2019, incluindo todas as zonas e instâncias das regiões *US-WEST-1* e *US-WEST-2*, observa-se que há um padrão de comportamento em algumas instâncias no que se refere ao volume de alterações dos seus preços, como pode ser observado na Figura 4.2.



(a) Instância C1Medium.

(b) Instância M4Large.

(c) Instância CR18XLarge.

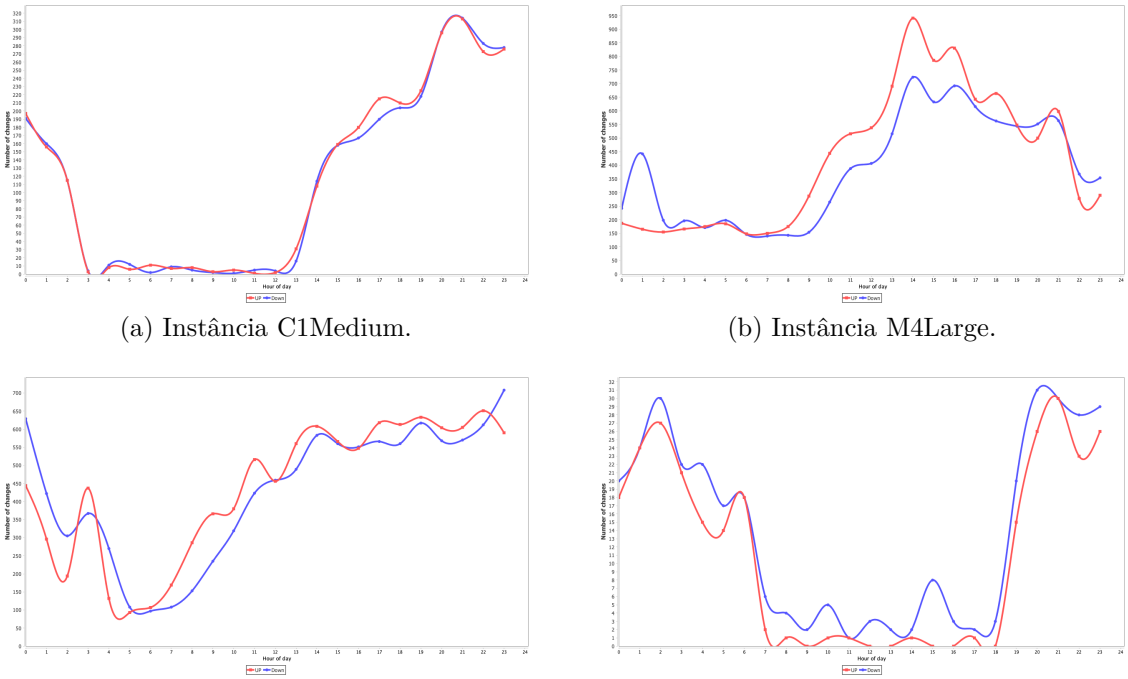
(d) Instância T2Small.

Figura 4.2: Padrões observados em mudanças de preços considerando os dias da semana.

Note nos gráficos da Figura 4.2, os quais apresentam a quantidade de alterações de preços (Eixo Y) em dias da semana (Eixo X), de queda (azul) e aumento (vermelha) de preço, que: o volume de alterações em *C1Medium* (a) varia durante a semana, com baixas no final de semana; o padrão da instância *M4Large* (b) sugere que há uma redução no volume de alterações no seu preço quando está próximo do final de semana (sexta e sábado); as alterações de preço na instância *CR18XLarge* (c) apresentam um pico durante o início da semana comparados ao final da semana (sábado e domingo); e a instância *T2Small* (d) atinge altos níveis de alterações apenas na sexta-feira.

Conforme apresentado, observa-se que volume de alterações varia de acordo com diferentes instâncias, podendo chegar a uma diferença aproximada de até 3000%, considerando o volume de alterações.

Considerando as horas do dia, a Figura 4.3 retrata as alterações dos mesmos tipos de instâncias, durante o mesmo período apresentado na Figura 4.2. Note que: a instância *C1Medium* (a) possui um volume considerável de alterações de preços a partir das 13 horas; as alterações na instância *M4Large* (b) aumentam no intervalo entre 11 e 23 horas; há uma queda no volume de alterações da instância *CR18XLarge* (c) entre 4 e 8 horas; e a instância *T2Small* (d) atinge baixos níveis de alterações durante o intervalo entre 7 e 18 horas.



(a) Instância C1Medium.

(b) Instância M4Large.

(c) Instância CR18XLarge.

(d) Instância T2Small.

Figura 4.3: Padrões observados considerando os horários no decorrer do dia.

Observando as Figuras 4.2 e 4.3, conclui-se que a probabilidade de falhas na aquisição das instâncias Spot aumentam quando há um volume maior de alterações de preços. Desta forma, um pequeno volume de alteração indica menor chance de falha de revogação. Nestes padrões, observa-se uma disparidade no volume de alterações que chega a ultrapassar 3000% na Figura 4.3 (a).

Então, percebe-se que o dia da semana e a hora do dia pode influenciar no tempo em que o usuário mantém ativa uma instância até que ela seja revogada, dependendo da estratégia que for utilizada para definir o valor do lance dado pelo usuário. Esta abordagem é tratada na literatura como uma estratégia a ser explorada, apontada como desafios em trabalhos futuros por alguns autores (Buyya et al., 2011; Voorsluys, 2014; Voorsluys and Buyya, 2012; Yi et al., 2012).

Em Anarado and Andreopoulos (2016); Binnig et al. (2015); Buyya et al. (2011); Jangjaimon and Tzeng (2015); Yi et al. (2012), os autores apresentam diversas estratégias para redução de custos ao utilizarem as instâncias Spot , e exploram estas variações em diversos modelos distintos em estratégias de lance. Como o foco deste trabalho não envolve o estudo da melhor estratégia de lance, assume-se o uso das seguintes estratégias: (i) valor atual da instância Spot ; (ii) média dos últimos 7 dias; (iii) mediana dos últimos 7 dias; (iv) média do último dia; (v) mediana do último dia; e (vi) valor atual da instância *on-demand*.

Desta forma, para o exemplo apresentado na Figura 4.2 (a), a probabilidade de falha de revogação quando o usuário utiliza a instância *C1Medium* durante a semana, principalmente, no intervalo entre 3 e 13 horas, é superior ao final de semana, devido a quantidade de mudanças de preço apresentadas nos gráficos.

Como apresentado na Seção 2.3, o modelo CBR possibilita a manutenção de casos previamente conhecidos para oferecer meios em que inferências possam ser realizadas na resolução e aprendizado baseado em problemas prévios.

Definição 6 (Caso) *Considera-se um caso δ como o cenário de aquisição de uma instância transiente, sendo uma 14-upla, onde:*

1. *I é composto por um subconjunto que indica qual a instância utilizada, incluindo o tipo de instância e seus recursos;*
2. *P_p é o provedor;*
3. *P_r é a região do provedor;*
4. *P_z é a zona do provedor, inserida na região indicada;*
5. *T_d é o dia da semana;*
6. *T_h é a hora do dia;*
7. *B_{est} é a estratégia de lance adotada;*
8. *V_0 é o valor pago por ciclos de hora;*
9. *V_x é o valor total pago;*
10. *T_i é o timestamp da aquisição;*
11. *T_f é o timestamp da cessão;*
12. *FT_{params} é um subconjunto de atributos contendo a técnica de TF utilizada e seus respectivos parâmetros e valores;*
13. *μ representa o tempo até a revogação;*
14. *ω indica se o caso representa uma censura.*

Desta forma, por meio do histórico de alterações de preço, observados nas Figuras 4.2 e 4.3, é possível construir uma base de casos ao simular um ambiente de leilão, em que instâncias são adquiridas através de um lance e, consequentemente, revogadas após o valor deste lance ser ultrapassado pelo valor modificado da instância. Com foco no período de tempo, os casos são criados com um atributo que indica o intervalo de permanência da instância até a sua revogação.

Exemplos de quantidade de casos gerados estão presentes na Tabela 4.2, que apresenta, para onze instâncias Spot (sistema operacional Linux) comumente utilizadas nas regiões *US-WEST-1* e *US-WEST-2*, a quantidade de alterações de preços e a quantidade de casos gerados através de uma simulação de ambiente de leilão (detalhada na Seção 4.3). Essa simulação considerou um histórico de aproximadamente 20 milhões de registros com alterações de preço.

Uma questão importante nesta pesquisa é como utilizar, de maneira apropriada, instâncias transientes na computação em nuvem para oferecer um ambiente resiliente para

a execução de aplicações de forma transparente (i.e., sem adaptação na aplicação). Esta questão levantou várias abordagens de pesquisa: trabalhos que exploram, quando aplicáveis, modelos estatísticos para definir um valor ótimo de lance a ser ofertado; trabalhos que utilizam técnicas de TF com parâmetros estáticos; e trabalhos que envolvem instâncias não-transientes para garantir a execução de aplicações.

Tabela 4.2: Quantidade de casos extraídos por instância.

Instância	**Qtd. alterações**	**Qtd. casos gerados**
c1.medium	16.650	6.163.710
c3.large	97.882	7.673.934
c3.8xlarge	1.145.136	5.539.384
c4.xlarge	244.041	7.530.124
cr1.8xlarge	20.938	391.907
m3.medium	113.327	5.700.272
m3.large	52.451	17.678.296
m3.xlarge	511.639	12.188.472
m4.large	139.671	8.570.063
r3.large	78.033	8.129.292
r4.large	204.087	3.368.551

Dada as diferentes abordagens existentes na literatura, chega-se a três possibilidades de melhoria conforme perspectivas específicas:

I Baseada em variação de mercado - na tentativa de evitar falha de revogação, essa perspectiva tem foco em estratégias para descobrir o melhor valor a ser oferecido na aquisição de uma instância, quando aplicável. Desta forma, um valor consideravelmente alto aumenta a probabilidade de garantia da instância alocada. Além disso, inclui a estratégia que envolve a dedução do melhor horário para execução de uma aplicação baseada no histórico de preços. Normalmente é utilizada apenas uma técnica de TF, não sendo o foco dos trabalhos a abordagem resiliente.

II Estática - nesta perspectiva, técnicas de TF são fixamente definidas sem levar em consideração aspectos do ambiente de execução. Neste caso, o foco está no experimento da aplicação com uso de uma técnica específica de TF: na utilização de replicação, fixar em duas cópias ou definir um intervalo fixo de *checkpointing*.

III Dinâmica - baseada em experimentos que mostram a sua eficiência, esta perspectiva foca na análise das características da aplicação, combinada com o ambiente de execução. O objetivo desta abordagem é analisar o cenário atual e definir qual a técnica de TF é mais adequada para ser aplicada na execução de aplicações.

Analisando as possibilidades propostas para melhoria na utilização de instâncias transientes, utiliza-se a estratégia da perspectiva dinâmica (Item III). Assim sendo, por meio da perspectiva dinâmica é possível escolher o mecanismo de TF apropriado. Busca-se um modelo arquitetural para o uso dessas instâncias, permitindo um ambiente resiliente para a execução de aplicações BoT de maneira distribuída, como proposto por Irwin et al. (2017).

Na Seção 2.3, a definição e a classificação de ambiente foi apresentada segundo nomenclatura apresentada em (Russell and Norvig, 2010). No cenário desta proposta, que envolve instâncias transientes na computação em nuvem, considera-se a composição destas instâncias como um ambiente distribuído, episódico, acessível, não determinístico, dinâmico e discreto.

Definição 7 (Ambiente) *No escopo desta proposta, que envolve o conjunto de instâncias transientes Λ, um ambiente ϵ é composto por todas as instâncias que estão sendo utilizadas para a execução de aplicações.*

Desta forma, um ambiente deve prover insumos suficientes para que recursos externos possam interagir e identificar modificações em seus estados. Como apresentado na Figura 2.10, um agente é representado por uma entidade computacional, que possui a capacidade de observar e perceber alterações no ambiente ϵ, através dos seus sensores, agindo sobre este ambiente ϵ, através de seus atuadores (Russell and Norvig, 2010).

Definição 8 (Agente) *Um agente α é uma tripla (C, S, F), onde:*

1. *C é um conjunto finito de comportamentos, com $C = \{c_1, c_2, ..., c_n\}$;*
2. *S é o conjunto de sensores;*
3. *F é uma função que avalia um evento para definir qual comportamento será adotado.*

Por meio do conjunto S, o agente α observa um ambiente ϵ, e na presença de um evento (mudança de estado), invoca a função F que indaga sua base de casos θ para decidir qual comportamento será adotado, obtendo um subconjunto de C (resultado da função F).

Os agentes presentes na arquitetura utilizam diferentes comportamentos considerando o instante em que o usuário submete uma aplicação após a análise dos estados do ambiente. No contexto desta proposta, seus comportamentos tem como objetivo a definição do plano de execução tolerante a falhas, em que a execução da aplicação, monitoramento das VMs e aplicabilidade de TF e seus respectivos parâmetros são gerenciadas dinamicamente e de forma autônoma. Uma visão abstrata dos agentes e seus comportamentos autônomos pode ser formalizado.

Definição 9 (Conjunto de Estados) *$E = \{e'_1, e'_2, e'_2, \cdots, e'_n\}$ representa um conjunto finito de estados discretos, e.g. disponibilidade de instâncias (e.g. não-disponível, disponível e inexistente), suas alterações de preços (e.g. aumento ou diminuição) e a disponibilidade de suas VMs com seus respectivos estados corrente de execução (e.g. preparando, pronta, finalizada).*

Definição 10 (Conjunto de Comportamentos) *Cada agente possui um conjunto finito de comportamentos $B = \{\beta'_1, \beta'_2, \beta'_3, \cdots, \beta'_j\}$ que age, interage e transforma os estados dos ambientes. Neste contexto, a decisão de qual comportamento β' será acionado depende do estado do ambiente e', i.e. E fornece um conjunto de estados e o agente define um conjunto de comportamentos (BE) que será executado.*

Este conjunto BE corresponde a uma sequência intercalada de estados e comportamentos $BE : e'_0 \xrightarrow{\beta_0} e'_1 \xrightarrow{\beta_1} e'_2 \xrightarrow{\beta_2} \cdots \xrightarrow{\beta_0 - 1} e'_u$. Como observado, cada estado e' reflete nas ações do agente em um processo contínuo de interação com o ambiente e execução até o último e_u ser alcançado.

Definição 11 (Histórico de Comportamentos) *Considere o histórico HIS como um conjunto de todas as possíveis sequências finitas (através de E e B), HIS^{BE} como um subconjunto de uma tupla $T =< e'_x, \beta'_x >$ que representa o estado e'_x que influencia o comportamento β'_x e HIS^E como um subconjunto de uma tupla $T =< \beta'_x, e'_{x+1} >$ como um novo estado e'_{x+1} que é alcançado como consequência do comportamento β'_x. Para representar o efeito que um comportamento de um agente implica em um ambiente, uma função que modifica um estado é definida como $\tau : HIS^{BE} \rightarrow \varphi(E)$.*

No contexto desta proposta, um ambiente Σ é formalizado como uma tripla $\Sigma =< E, e'_0, \varphi >$, com E representando um subconjunto de estados do ambiente, $e'_0 \in E$ como o seu estado inicial e φ como uma função modificadora de estado, após o raciocínio do agente.

De acordo com a definição e classificação de um ambiente, apresentado em Russell and Norvig (2010), no cenário desta proposta, a composição de instâncias transientes é considerada como distribuída, acessível, não-determinística, dinâmica e discreta.

Definição 12 (Conjunto de Agentes) *Esta proposta contém um conjunto de agentes autônomos composto por uma quádrupla $BR_{AG} =< AG, PR, BE, G >$:*

1. *AG representa um conjunto dos agentes da proposta;*
2. *PR é um conjunto finito de preposições a serem usadas pelos agentes;*
3. *BE descreve o conjunto de comportamentos disponíveis para os agentes, com um comportamento b' definido a partir de um subconjunto de condições $cond(b') \subseteq PR$;*
4. *G representa o conjunto de objetivos a serem alcançados $G \subseteq PR$.*

Para entender os elementos desta proposta em relação aos agentes e sua modelagem CBR, a arquitetura macro baseada em agentes é apresentada na Figura 4.5 e algumas definições podem ser formalizadas da seguinte maneira:

- *sen-avail* é comporto por um sensor que observa a disponibilidade de instâncias transiente em ambientes de computação em nuvem;
- *sen-price* representa um conjunto finito de sensores que monitoram as alterações de preços em instâncias transientes. De acordo com a natureza de um ambiente de leilão, o preço varia de acordo com a oferta e a procura. Neste contexto, o modelo necessita que as mudanças de preço sejam capturadas e que estejam disponíveis de maneira atualizada;
- *sen-fail* é um conjunto de sensores que observam a disponibilidade e a execução de instâncias transientes no ambiente de computação em nuvem, observando eventos de revogação;
- *sen-exec* é composto por sensores que observam e monitoram um conjunto de n VMs que estão em execução, composto por um BoT com um subconjunto de tarefas $T = \{t_1, t_2, \cdots, t_n\}$. Cada VM recebe uma tarefa t_n e executa o mesmo BoT de maneira independente;
- B é um conjunto com comportamentos previamente implementados, individualmente criados para permitir alcançar os seus respectivos objetivos de acordo com o plano de execução. Este plano, detalhado na Seção 4.3, é composto por um conjunto de comportamentos $B = \{\beta_1, \beta_2, \beta_3, \cdots, \beta_n\}$ cujo objetivo é garantir a execução da aplicação em um ambiente resiliente;
- *database* representa um banco de dados, localizado apenas no *container* principal, que contém um conjunto de todas as alterações de preços e dados processados, incluindo os casos usados pelo modelo CBR;
- ξ retrata a função principal que avalia um evento observado para decidir que comportamento será realizado a partir de um conjunto de ações previamente estabelecidos B;
- *location* representa um marcador (*flag*) que indica se corresponde a um agente local ou remoto. Os agentes desta proposta estão inseridos em um contêiner centralizado que permite a criação de agentes remotos e este atributo indica se a execução do agente está na máquina controladora ou na VM que está sendo executada a tarefa (máquina remota).

Agentes são resolvedores de problemas independentes (e.g., Monitor de Preços e Recuperador) que compartilham e colaboram para atingirem objetivos globais (e.g., garantia da execução) considerando restrições e objetivos individuais (e.g., monitoramento de preços e recuperação de falhas). Cada agente possui um objetivo e é apoiado pelos seus sensores e atributos, como apresentado na Tabela 4.3.

Tabela 4.3: Definições de atributos e sensores dos agentes.

Agente	*sen-avail*	*sen-preco*	*sen-falha*	*sen-execucao*	B	*base-dados*	ξ	*localização*
Verificador	✓	-	-	-	✓	-	✓	local
Monitor de Preço	✓	✓	-	-	✓	✓	-	local
Core	-	-	-	-	-	✓	✓	local
Gerente de Execução	✓	-	-	-	-	-	✓	local
Executor	-	-	✓	✓	✓	-	✓	remoto
Monitor	✓	-	✓	✓	✓	-	✓	local/remoto
Recuperador	-	-	-	✓	✓	-	✓	remoto

Os sensores mencionados estão relacionados a observação da ocorrência de um evento, descrito em seus itens. No contexto desta proposta, a observação se dá através de submissão de interesse em determinados eventos pelo protocolo *Publish/Subscribe*, como apresentado na Seção 2.3.

4.2 Arquitetura Baseada em Agentes

Nesta seção é apresentada a arquitetura baseada em agente proposta, em que os agentes analisam a disponibilidade atual das instâncias transientes para definir uma estratégia de TF apropriada. Entre as técnicas de TF consideradas estão: a repetição (*retry*), os pontos de verificação (*checkpoint/restore*) e a replicação (*replication*), com seus respectivos parâmetros.

4.2.1 Arquitetura Física

A arquitetura proposta é composta por um contêiner global que compartilha o ambiente entre agentes locais e remotos. A Figura 4.4 apresenta o modelo físico dos agentes utilizados no processo de execução de uma aplicação.

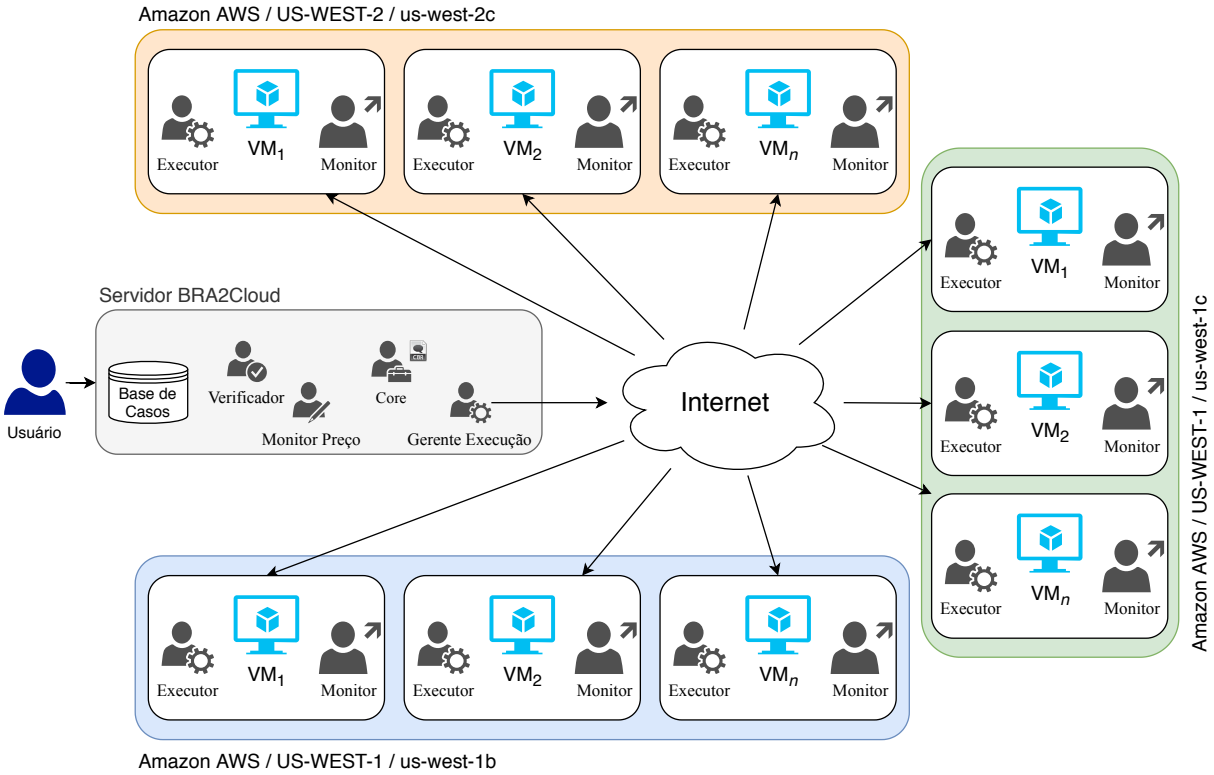


Figura 4.4: Arquitetura física na distribuição dos agentes.

O servidor BRA2Cloud consiste em um ambiente em que os agentes locais (Verificador, Monitor de Preço, Core e Gerente de Execução) executam em uma máquina centralizadora, iniciando um Container Global para o gerenciamento e execução de agentes. Além disso, a base de casos fica localizada neste servidor ou em algum *host* na mesma rede, de modo que o agente Core possa ter acesso para recuperar os casos similares, detalhados na Figura 4.8.

Observando a Figura 4.4, percebe-se que o modelo permite gerenciar VMs em diferentes regiões e zonas. Cada nova instância (VM) criada, um agente Executor e outro Monitor são executados remotamente para, respectivamente, executar e monitorar a instância durante a execução da aplicação. Portanto, n instâncias transientes serão criadas e gerenciadas pela arquitetura de agentes através da Internet.

4.2.2 Arquitetura de Software

A arquitetura é composta por agentes de software, que definem, executam e monitoram as execuções distribuídas nas instâncias transientes, com o objetivo de garantir a execução da aplicação. O detalhamento dos agentes e seus respectivos PAGE são apresentados na Tabela 4.4 e na Figura 4.5.

Tabela 4.4: Definições PAGE dos agentes.

Agente	**Percepções**	**Ações**	**Objetivos**	**Ambiente**
Verificador	Recebe os parâmetros da aplicação, dados e arquivos.	Analisa os dados e arquivos enviados; avalia os preços e disponibilidade das instâncias; define os tipos de instâncias.	Valida os dados informados; define o provedor e instâncias que correspondem com os requisitos das tarefas do usuário.	Parcialmente observável, determinístico, sequencial, estático e discreto.
Monitor de Preços	Obtém um conjunto de instâncias para monitorar com suas respectivas regiões e zonas; Identifica mudanças de preço nos tipos de instâncias.	Analisa o valor atual da instância; avalia as alterações nas instâncias; sinaliza mudança de preços.	Mantém o banco de dados de preços atualizado.	Observável, determinístico, sequencial, dinâmico e contínuo.
Core	Recebe do agente Verificador o provedor de computação em nuvem e suas respectivas definições de instâncias; recupera a base de casos.	Solicita a base de casos; aplica o modelo de similaridade; recupera casos similares para calcular a probabilidade de falha baseado no TAR.	Define um plano de execução tolerante a falhas para distribuir as tarefas e executá-las respeitando a técnica de TF previamente definida.	Parcialmente observável, determinístico, episódico, estático e discreto.
Gerente de Execução	Recebe o plano de execução.	Solicita a criação de n VMs; envia os arquivos necessários para a VM; cria um agente Executor em cada VM; avalia a execução da aplicação; finaliza a VM quando a execução finaliza.	Mantém os estados da VM e de sua execução; avalia a solução do caso proposto; recupera o estado do caso.	Observável, estocástico, sequencial, dinâmico e contínuo.
Executor	Recebe os arquivos da aplicação; monitora a execução na instância.	Executa a aplicação; avalia a execução da aplicação; envie sinais e dados após a conclusão da execução.	Mantém o estado de execução da tarefa; recupera os dados de execução.	Observável, estocástico, sequencial, dinâmico e contínuo.
Monitor	Obtém um conjunto de VMs em execução para monitorar.	Monitora a execução remota de tarefas; envia informações sobre uma falha na VM; revalida eventos para evitar falhas falsos positivas.	Fornece dados de status da execução da tarefa na VM; relata eventos de falha para o agente Recuperador.	Observável, determinístico, sequencial, dinâmico e contínuo.
Recuperador	Recebe o sinal de falha na execução da tarefa quando ocorre uma falha.	Realiza o processo de recuperação de acordo com o plano de execução tolerante a falhas.	Mantém o estado de execução da VM; garante o uso de técnicas de TF; avalia a solução de caso proposta; recupera o estado do caso.	Observável, determinístico, sequencial, dinâmico e contínuo.

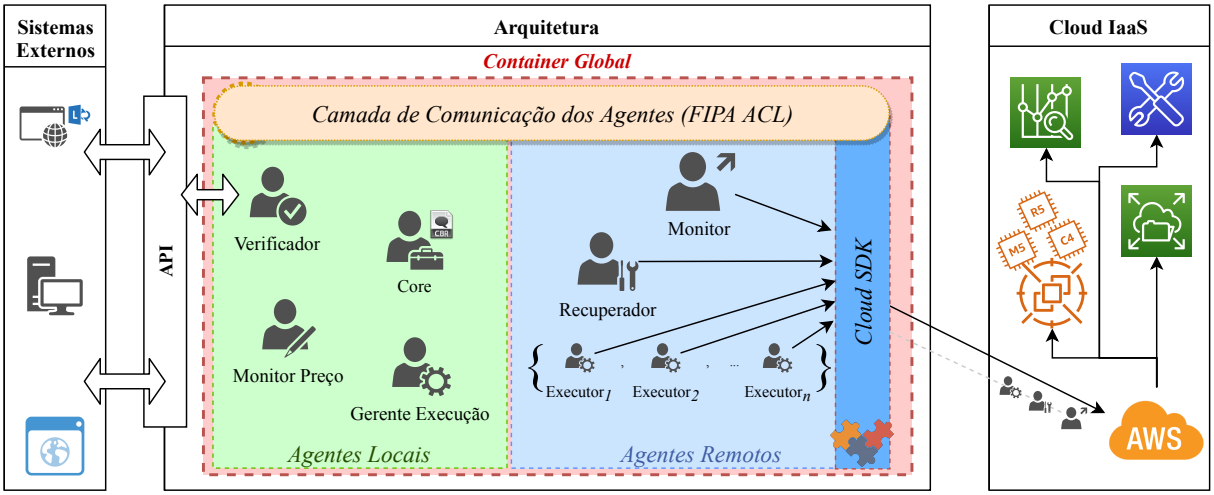


Figura 4.5: Arquitetura geral baseada em agentes.

Segue o detalhamento dos agentes presentes na Figura 4.5:

- Agente Verificador – valida os dados (parâmetros) fornecidos pelo usuário e escolhe qual instância transiente será utilizada baseado nas especificações dadas pelo usuário, o que inclui os arquivos da aplicação que será executada, o tempo estimado necessário para a execução da aplicação e a quantidade de recursos de CPU e memória necessários para a execução da aplicação, sendo opcional a escolha do tipo de instância que será executada;
- Agente Monitor de Preços – responsável por monitorar todas as alterações de preços presentes nas instâncias no ambiente de computação em nuvem. Em relação as instâncias Spot , cada alteração de preço reflete em um novo caso gerado a partir do agente Core, que adiciona um novo registro na base de casos baseado em um cálculo de tempo até a revogação;
- Agente Core – define um plano de execução a partir da escolha da técnica de TF mais apropriada e seus respectivos parâmetros. Para atingir altos níveis de acurácia, este agente utiliza as abordagens de predição do Tempo até a Revogação (TAR) apresentadas na Seção 4.3, que fornece elementos para calcular a taxa de sobrevivência, sendo uma informação mandatória para apoiar nas decisões dos agentes. Os comportamentos subsequentes dependem das definições deste agente;
- Agente Gerente de Execução – um gerente de execução nas VMs, responsável por criar n agentes Executores para atender aos requisitos da aplicação, i.e. se um usuário requer 50 execuções paralelas, um total de 50 VMs transientes serão criadas e suas execuções necessitam de gerenciamento;
- Agente Executor – realiza o processo de execução das tarefas e o monitoramento das execuções nas VM adquiridas, respeitando as definições previamente realizadas

pelos agentes Monitor de Preços e Core, i.e. o provedor de computação em nuvem, tipos de instâncias, plano de execução TF e seus respectivos parâmetros;

- Agente Monitor – um agente que monitora a VM para imediatamente informar quando houver uma falha. Além do processo de monitoramento realizado pelo agente Executor, o agente Monitor torna-se mais eficiente desde que seus comportamentos ocorrem como um observador externo e remoto;
- Agente Recuperador – acionado apenas quando há a presença de uma falha identificada pelo processo de monitoramento, este agente é instanciado quando uma falha ocorre. Este agente é informado através de uma mensagem enviada pela camada de comunicação e aplica o processo de recuperação para garantir a execução da aplicação. Assim como o agente Executor, este agente respeita as definições previamente estabelecidas em relação a técnica de TF e seus parâmetros.

4.2.3 Protocolos de Comunicação e Interação

Os agentes descritos na Seção 4.2.2 utilizam uma linguagem de comunicação denominada *Agent Comunication Language* (ACL) para trocar mensagens e permitir a interoperabilidade com diferentes tecnologias baseadas em agentes. Atualmente a linguagem FIPA ACL[2] define um conjunto com 22 performativas. Nesta proposta, nos limitamos as performativas apresentadas na Tabela 4.5.

Tabela 4.5: Performativas utilizadas nesta proposta.

Performativa	**Descrição**
ACL.inform	Uma das performativas mais importantes da FIPA ACL, ela representa o ato de comunicar alguma informação para outros agentes.
ACL.request	Também uma das performativas mais importantes da FIPA ACL, ela consiste em uma requisição a um agente para executar determinada ação.
ACL.request-when	Representa uma requisição condicionada, i.e. o agente inicia a execução de uma determinada ação quando a condição for verdadeira.
ACL.confirm	Representa uma confirmação de confiança da mensagem recebida.
ACL.disconfirm	Funciona como o inverso do *ACL.confirm*, que representa uma discordância da mensagem recebida.
ACL.agree	Comumente utilizada para responder um *ACL.request*, ela representa a aceitação de desempenhar uma determinada ação.
ACL.query-ref	Utilizada quando um agente pede alguma informação.
ACL.failure	Representa uma tentativa de executar uma dada ação que refletiu em uma falha.
ACL.subscribe	Representa a inscrição de interesse em um evento, em a ocorrência deste evento gera uma *ACL.inform* para comunicar os agentes interessados.

A integração com infraestruturas de nuvem externas ocorre por meio de módulos conectáveis (*plugins*) para dar suporte a outros provedores de nuvem. No escopo desta

[2]Disponível em `http://www.fipa.org/specs/fipa00037/SC00037J.html`

proposta, é utilizada a biblioteca JAVA AWS para a comunicação com o provedor Amazon. Além disso, uma interface de acesso externo é fornecida para permitir a integração e interoperabilidade com sistemas externos utilizando uma API que fornece Web Services de acordo com os padrões RESTful (Richardson and Ruby, 2008).

Um breve diagrama de sequência que reflete as interações desde a submissão de um BoT até a detecção de uma falha é apresentada na Figura 4.6. O diagrama demonstra um cenário em que uma falha é detectada, e o agente Monitor sinaliza o agente Recuperador, que irá recuperar o estado falho de execução da VM transiente de acordo com a técnica de TF definida anteriormente.

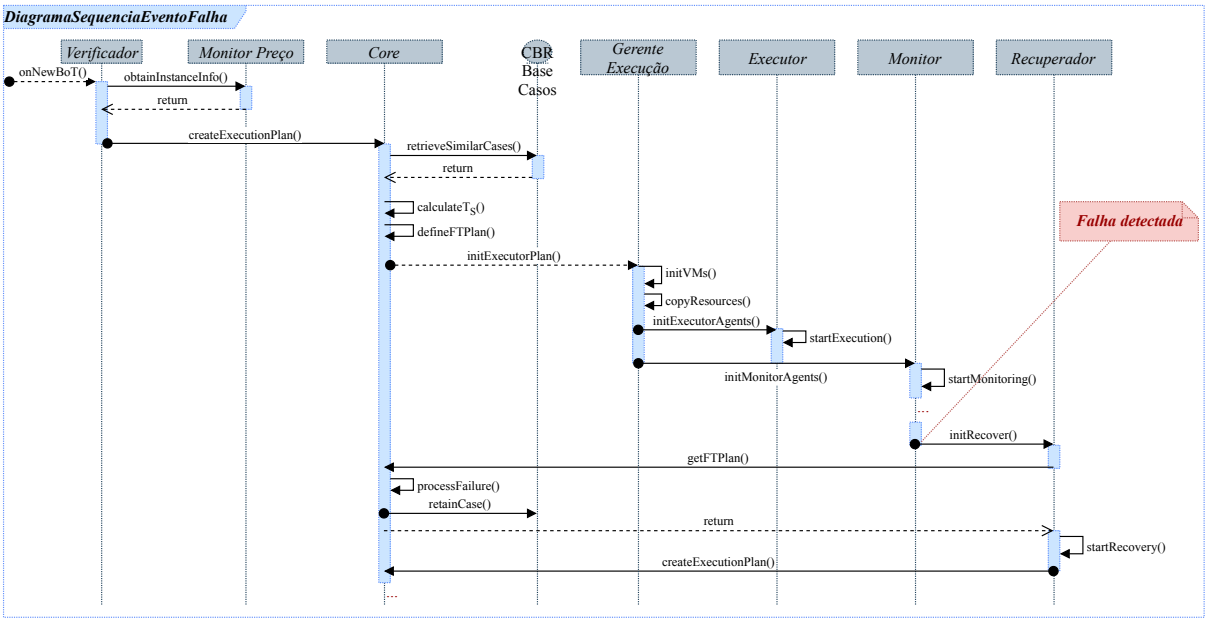


Figura 4.6: Distribuição dos agente e suas interações.

Observa-se na Figura 4.6 a existência dos agentes presentes na Figura 4.5 e suas interações. Os eventos que disparam estas interações são apresentados na Figura 4.7. Para um projeto que envolve agentes, a caracterização destes é necessária para o entendimento de seus objetivos e suas interações.

A caracterização de agentes se dá através da definição do PAGE - Percepção (*Percepts*), Ação (*Actions*), Objetivos (*Goals*) e Ambiente (*Environment*), apresentada na Tabela 4.4.

A definição do PAGE é importante para o completo entendimento dos agentes em relação aos seus objetivos, comportamentos e interações com o ambiente. As definições presentes nesta tabela representam a racionalidade de um agente, onde: a percepção descreve todos os possíveis dados fornecidos pelo ambiente em que o agente está inserido; as ações representam todos os possíveis comportamentos do agente para modificar o ambiente; os objetivos indicam as possíveis ações para um plano de execução; e o ambiente representa um local em que os agentes residem.

Para entender os elementos desta proposta em relação aos agentes, suas interações (*publish/subscribe*) e seus comportamentos, um complemento da arquitetura é apresentada na Figura 4.7.

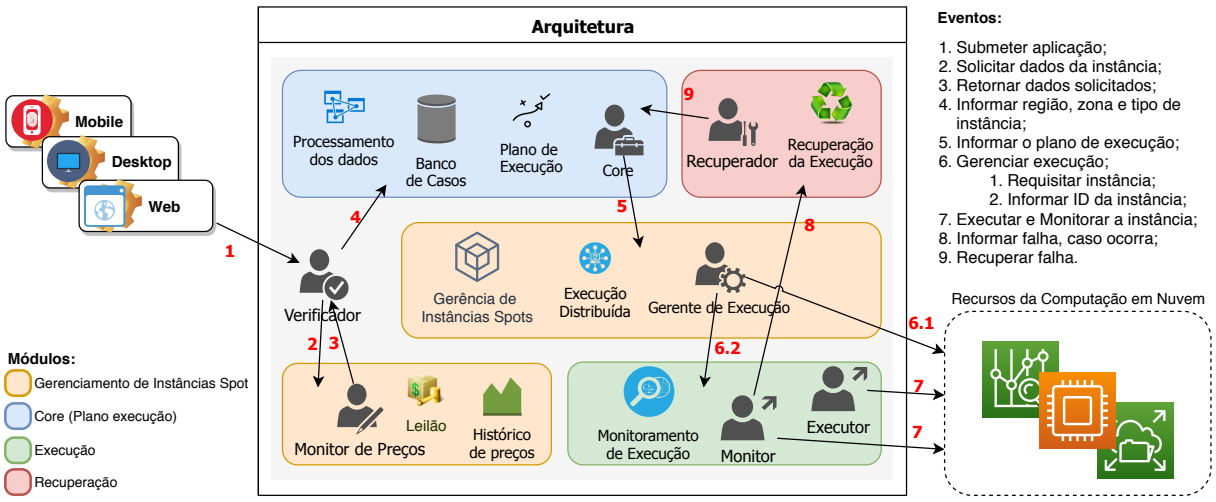


Figura 4.7: Interações e comportamentos dos agentes.

Observando a Figura 4.7, após a submissão do usuário (1), o processo de interação dos agentes ocorre da seguinte maneira: (2) o agente *Verificador* utiliza a performativa *ACL.query-ref* para solicitar dados que atendam aos requisitos do usuário; (3) o agente *Monitor* avalia a disponibilidade da instância solicitada pelo usuário, retornando uma mensagem *ACL.confirm* com os dados da instância ou *ACL.disconfirm* no caso de indisponibilidade de uma instância que atenda aos requisitos do usuário; em (4), o agente *Verificador*, através do *ACL.inform*, disponibiliza o ambiente de execução, incluindo a região, zona e instância que deverá ser utilizada; (5) o agente *Core* informa o plano de execução utilizando *ACL.inform* e faz a requisição do ambiente de execução através do *ACL.request*, obtendo a resposta através do *ACL.agree*; em (6.1), uma instância é requisitada e no (6.2) o agente *Gerente de Execução* utiliza o *ACL.request-when*, indicando que a execução e monitoramento deve iniciar a partir da disponibilidade da instância solicitada; por fim, se uma falha ocorrer, o *ACL.failure* será utilizado para informar o agente *Recuperador*, que realiza o processo de recuperação e usa o *ACL.inform* para o agente *Core* iniciar novamente o processo de execução.

Considerando um sistema SIS_{bra} como um par de um agente AG_{bra} e um ambiente ENV_{bra}, $SIS_{bra} =< AG_{bra}, ENV_{bra} >$, no cenário desta proposta, cada agente possui um ambiente definido, e.g. o agente Monitor de Preços está limitado a observar apenas o ambiente de leilão enquanto que os agentes Executores e Recuperadores compartilham o mesmo ambiente.

Considere uma execução de um BoT que possui um conjunto de tarefas n:

- n instâncias transientes serão solicitados;
- Deve haver um agente Verificador global para validar os dados de entrada;
- É necessário um agente Core global para definir o plano de execução TF;
- Um Gerente de Execução é utilizado para gerenciar e cumprir o plano de execução anterior e criar n agentes Executores remotos que devem transferir os arquivos necessários, executar e monitorar a execução das tarefas distribuídas;
- São criados n agentes Monitores para observar a integridade das VMs; e
- r agentes Recuperadores temporários, sendo r um número inteiro positivo $r \leq n$, sendo criado apenas na ocorrência de uma falha.

4.3 Modelo de Raciocínio

Em Araujo Neto et al. (2018); Buyya et al. (2011), os autores demonstram que os padrões de alteração de preço podem ser observados e revelam que a disponibilidade de VMs varia de acordo com um conjunto de atributos: região, zona, tipo de instância e diferente horários dos dias da semana. O processo de raciocínio dos agentes é detalhado nesta seção.

4.3.1 Raciocínio Baseado em Casos

Padrões de mudanças de preços podem ser observados nas Figuras 4.2 e 4.3, que demonstram o volume de mudança de preços agrupados por dia da semana e hora do dia, respectivamente.

Baixos índices de mudanças de preço implicam em uma menor chance de falhas, caso contrário as chances de revogação aumentam. Portanto, nossa abordagem usa CBR para classificar casos, considerando os padrões de hora do dia e dia da semana. Os preços históricos e suas alterações são analisados para definir o plano de execução tolerante a falhas apropriado.

Comparado a abordagens existentes, nossa abordagem utiliza as observações nos padrões de mudança de preços e disponibilidade calculados e observados a partir do gráficos de dia da semana e hora do dia, conforme proposto em (Buyya et al., 2011; Javadi et al., 2013). CBR é um modelo adequado para resolver o problema apresentado neste trabalho.

Para incrementar o modelo de raciocínio, esse modelo utiliza as abordagens de equivalência e adaptação, que envolvem o entendimento de casos anteriores para auxiliar em decisões semelhantes e, no caso de falha, permitem sua adaptação.

O modelo de formalização do CBR desenvolvido neste trabalho inclui um elemento caso (δ), que corresponde ao contexto de uma situação real, representado como um conjunto finito de pares de chave/valor de tamanho n, composto por atributos (σ) e valores (υ):

$$\delta = \{< \sigma_0, \upsilon_0 >, < \sigma_1, \upsilon_1 >, < \sigma_2, \upsilon_2 >, ..., < \sigma_n, \upsilon_n >\} \tag{4.1}$$

Cada função de similaridade é usada não apenas para o processo de recuperação dos casos similares, mas também para adaptar novos casos para incrementar o banco de dados de casos, sendo determinada como a proximidade dos valores dos atributos e seus respectivos pesos, sendo definidos de acordo com o contexto do problema.

O contexto pode ser considerado como uma interpretação de um caso, em que apenas um subconjunto de atributos selecionado é considerado relevante em comparação com outros, com suas especificações e pesos definidos de acordo com as informações detalhadas na Tabela 4.6.

Tabela 4.6: Atributos do caso (δ) com as respectivas estratégias e pesos de suas funções de similaridade.

Atributo	**Detalhamento**	**Estratégia da Função**	**Peso**
instância	Um subconjunto de atributos indicando a instância utilizada, incluindo o nome da instância e seus recursos.	Constante	1 quando o atributo é igual, 0 quando contrário.
provedor	Representa o provedor de computação em nuvem.	Constante	1 quando o atributo é igual, 0 quando contrário.
região	Indica a região dentro do provedor selecionado.	Constante	1 quando o atributo é igual, 0 quando contrário.
zona	Atributo que indica a zona que está inserida na região.	Constante	1 quando o atributo é igual, 0 quando contrário.
dia-da-semana	Representa o dia da semana.	Euclideana	Um valor entre 0 e 1 de acordo com a proximidade do dia da semana, considerando valores entre 0 e 7.
hora-do-dia	Representa a hora do dia.	Euclideana	Um valor entre 0 e 1 de acordo com a proximidade da hora do dia, considerando valores entre 0 e 23.
estrategia-lance	Representa a estratégia de lance usada pelo usuário na definição do lance.	Constante	1 quando o atributo é igual, 0 quando contrário.
valor-lance	Representa o valor do lance dado pelo usuário, utilizado para adquirir a VM.	N/D	N/D.
valor-total	Indica o real valor total pago pelo usuário no uso da VM.	N/D	N/D.
time-inicio	Atributo que indica o instante (*timestamp*) da aquisição da VM.	N/D	N/D.
time-fim	Atributo que indica o instante (*timestamp*) da revogação da VM.	N/D	N/D.
TAR	Representa o TAR, considerando ***time-inicio*** e ***time-fim***.	N/D	N/D.
$\boldsymbol{TF_{params}}$	Um subconjunto de atributos contendo a técnica de TF utilizada com seus respectivos parâmetros e valores.	N/D	N/D.
censurado	Atributo que indica se o δ representa um evento de censura, usado pela análise de sobrevivência.	N/D	N/D.

Atributos com estratégia de função N/D (não definidas) indicam que eles não são usados em uma função de similaridade, sendo um dado importante usado pelo processo de raciocínio dos agentes. Um contexto Γ é definido como um conjunto finito de atributos de um subconjunto j (Ω) com restrições associadas ($C\alpha$) em $\Gamma = \{< \Omega_1, C\alpha_1 >, < \Omega_2, C\alpha_2 >, \cdots, < \Omega_j, C\alpha_j >\}$.

Um conjunto de casos (Δ) pode ser criado usando algoritmos que simulam um cenário de leilão com lances de usuário (U_{bid}), extraindo dados das alterações de preço do histórico (HIS_p). Esses casos, gerados a partir de dados reais extraídos de HIS_p, podem ser usados para calcular o tempo de sobrevivência esperado.

Conforme apresentado no Algoritmo 1, os casos podem ser gerados usando o resultado de uma função estimada $EST_{bid}(HIS_p, n)$, que calcula o preço da instância dada uma estratégia de lance na definição do U_{bid} e simulando os tempos de sobrevivência ao acompanhar os registros subsequentes de HIS_p, revogando a instância no momento em que o preço excede o U_{bid}.

Conforme detalhado no Algoritmo 1, o processo de CBR proposto utiliza as experiências anteriores (casos) presentes em um modelo de aprendizado contínuo, permitindo adaptações quando novos problemas são avaliados.

Além disso, a capacidade de equivalência e adaptação torna um modelo eficiente para resolver o problema apresentado neste artigo, que envolve a compreensão de casos anteriores para auxiliar em decisões semelhantes, permitindo a adaptação e o ajuste de casos recuperados para soluções similares.

Algorithm 1 Processo de geração de casos na base de casos.

```
1:  procedure GeradorCasos(reg, zona, tipoInst, initTim, finalTime)
2:      Inicialização do sistema
3:      parametros ← [reg, zona, tipoInst, initTim, finalTime]
4:      listaHistorico[] ← recuperarHistoricoDB(parametros)
5:      estrategiaLances[] ← recuperarEstrategiasLance()
6:      listaCasos[] ← newList()
7:      for estrategia in estrategiaLances do                    ▷ Processa cada estrategia
8:          for preco in listaHistorico index i do               ▷ Cada registro de preço
9:              timeBase ← preco.time
10:             precoBase ← preco.preco · estrategia
11:             censurado ← True
12:             for precoFuturo in listaHistorico do             ▷ Mudanças futuras
13:                 precoComparado ← listaHistorico[i + 1].preco|precoFuturo
14:                 if precoComparado > precoBase then           ▷ Localizado um valor acima
15:                     censurado ← False
16:                     min = calcMinutos(timeBase, listaHistorico[i + 1].time)
17:                     listaCasos.add(newCase(setOfAttrs))
18:             if censurado then                                ▷ Não foi localizado um valor acima do 'lance'
19:                 censurado ← True
20:                 min = calcMinutos(timeBase, listaHistorico[i + 1].time)
21:                 listaCasos.add(newCase(setOfAttrs))
22:         adicionarCasosNaBaseCasos(listaCasos)
23:     return listaCasos
```

O processo de geração de casos (Algoritmo 1) é composto por um conjunto de casos reais e seus respectivos atributos, conforme apresentado na Tabela 4.6. Ao combinar casos, alguns atributos são forçados a serem iguais.

Em um processo de similaridade, particiona-se θ em θ_E e θ_D, representando atributos que devem ser iguais e atributos que podem ser diferentes, respectivamente. Utilizando esta abordagem de particionamento dos atributos, a função de cálculo de similaridade na comparação dos casos é formalizada como segue:

$$SimFunc(\delta^{(1)}, \delta^{(2)}) = IGUAL \wedge PROXIMA \tag{4.2}$$

em que:

$$IGUAL \rightarrow FUNC(\gamma_\theta^{(1)} = \gamma_\theta^{(2)}) = \prod_{\theta \in \theta_E} A_I(\gamma_\theta^{(1)} = \gamma_\theta^{(2)})$$
$$PROXIMA \rightarrow FUNC(\gamma_\theta^{(1)} = \gamma_\theta^{(2)}) = \prod_{\theta \in \theta_D} K_\theta(|\gamma_\theta^{(1)} - \gamma_\theta^{(2)}|).$$

A Equação 4.2 possui uma função de ajuste (A_I) e uma função de *kernel* (K_θ). A função A_I resulta em 1 se seu argumento for verdadeiro e 0 em caso contrário, considerando a equidade dos argumentos. Além disso, a função K_θ é uma função analítica, analisando cada atributo em θ_D e assumindo o valor 1 quando $\gamma_\theta^{(1)} = \gamma_\theta^{(2)}$ e assumindo um valor tendenciando a 0 de acordo com a distância entre eles.

Considerando que a composição dos atributos (região, zona, tipo de instância, dia da semana e hora do dia) pode influenciar o tempo de revogação, as funções de similaridade implementadas no modelo CBR oferecem uma abordagem eficiente para resolver o problema colocado neste trabalho. Uma vez gerado um banco de dados de casos, é necessário recuperar casos semelhantes, conforme apresentado no Algoritmo 2.

Algorithm 2 Recuperando casos similares.

```
1:  procedure RecuperaCasosSimilares(reg, zona, tipoInst, dia, hora)
2:      parametros ← [reg, zona, tipoInst, dia, hora]
3:      modelo ← carregaModelo()          ▷ Carrega o modelo CBR, incluindo as funções de similaridades.
4:      modelo.casos ← carregaCasos(parametros)          ▷ Carrega a base de casos
5:      atributosCaso[] ← modelo.carregaAtributos()
6:      mapaCasosSimilares[] ← newList()
7:      for caso in modelo.casos do          ▷ Iterando em cada caso
8:          caso.peso ← 0
9:          for atributo in atributosCaso do          ▷ Iterando em cada atributo modelado
10:             tipoFuncao ← atributo.getTipoFuncao()
11:             if tipoFuncao eq modelo.funcao['HEAVISIDE'] then
12:                 caso.peso += (caso["atributo"] == atributo in parametros ? atributo.peso : 0)
13:             if tipoFuncao eq modelo.funcao['EUCLIDEAN'] then
14:                 caso.peso += calculaPeso(caso["atributo"], atributo, atributo.peso)
15:             if tipoFuncao is null || tipoFuncao eq modelo.funcao['UNKNOWN'] then
16:                 raise "Função não reconhecida (tipoFuncao)"
17:         mapaCasosSimilares.put(caso)
18:     filtrarListaPorPesoMax(mapaCasosSimilares) as Set
19:     return mapaCasosSimilares
```

O banco de casos é composto por atributos de casos (Tabela 4.6), organizado como um conjunto de tuplas (Equação 4.1) a serem exploradas pelo processo esquemático CBR

(ilustrado na Figura 4.8), composto por uma geração inicial de conhecimento (detalhada no Algoritmo 1), seguida por um conjunto de oito etapas sequenciais, como segue:

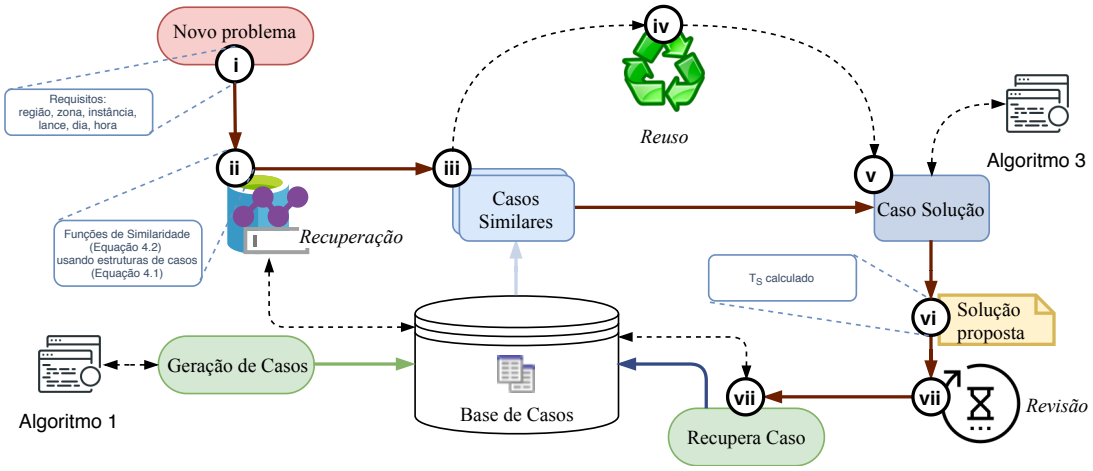


Figura 4.8: Visão geral esquemática do ciclo de CBR utilizado.

(i) Novo Problema – uma etapa que representa um novo problema (caso), enviado pelo usuário, que contém os atributos solicitados de região, zona, instância, estratégia de lance, dia da semana e hora do dia;

(ii) Recuperação – conforme apresentado no Algoritmo 2, esta etapa utiliza funções de similaridade usando a Equação 4.2 para buscar um conjunto de casos semelhantes de acordo com os atributos solicitados. A função usada inclui a soma ponderada de todas as semelhanças locais (semelhanças de atributo) resultando em uma medida global de similaridade. Na etapa de recuperação, a função de similaridade processa atributos de caso e usa pesos diferentes (Tabela 4.6) para aproximar ou maximar casos calculados;

(iii) Casos Similares – representa uma estrutura com um conjunto de casos semelhantes recuperados para serem reutilizados como referência no processamento do caso solução. Os casos possíveis são aqueles que combinam os mesmos atributos comparado a um novo problema;

(iv) Reuso – a adaptação e a reutilização dos casos recuperados são necessárias nesta etapa, sendo adequado construir uma solução proposta para o novo problema (caso) de acordo com casos observados anteriormente;

(v) Caso Solução - após a adaptação, um conjunto de casos é usado como entrada para um algoritmo que avalia os dados de cada caso, ignorando casos atípicos e calculando o tempo até a revogação, conforme apresentado no Algoritmo 3;

(vi) Solução proposta - como resultado da etapa de caso solução, os dados da solução proposta são compostos pelo tempo até a revogação, considerando o cenário de disponibilidade atual e a estratégia de lance adotada;

(vii) Revisão – a solução proposta é avaliada nesta etapa para confirmar sua eficiência em um cenário com casos de teste reais. Nesse cenário, o tempo até a revogação é observado;

(viii) Recupera caso – após a avaliação, é necessário explorar o caso resolvido e transformá-lo em um caso aprendido, adicionando sua solução ao banco de casos local. Se o tempo proposto até a revogação for alcançado, um caso de sucesso será adicionado ao banco de casos, seguido pelos parâmetros de TF definidos. Se o tempo até a revogação não for alcançado, as propriedades do caso são as mesmas, mas o tempo de execução atual estará presente em novos casos semelhantes obtidos na etapa de recuperação e na alteração de novas soluções propostas que usaram fator de recência, conforme apresentado na Seção 5.1.

Um sistema inteligente deve ser capaz de raciocinar e se adaptar a novas situações, entender as relações entre os fatos, descobrir significados e reconhecer as assertividades para aprender com base em sua experiência.

4.3.2 Análise de Sobrevivência

Analisando os casos semelhantes e baseado nas definições apresentadas na Seção 4.1, foi identificada uma disparidade nos casos em relação ao tempo em que uma instância permanece ativa, com casos em que há variação entre 1 e 2.761 horas. Casos raros em que o tempo de permanência de uma instância é alto indica que o caso foi gerado em um momento em que houve um valor elevado no ambiente de leilão.

Para apoiar o processo de raciocínio dos agentes, nossa abordagem usa um modelo estatístico para ignorar casos raros em um cenário em que o preço e a disponibilidade variam de acordo com a oferta e a demanda. Os modelos estatísticos fornecem mecanismos para extrair informações, às vezes sem dados completos, fornecendo suporte para organizar, analisar e apresentar dados processados que podem ser úteis nos processos de tomada de decisão.

Entre uma variedade de métodos estatísticos, esta proposta usa o modelo de Análise de Sobrevivência apresentado na Seção 2.4. No cenário desta proposta, uma falha de revogação e o tempo até uma falha, TAR, são tratados como os principais elementos. É utilizada uma técnica não paramétrica, que incorpora dados incompletos (censurados) para evitar uma estimativa viciada. Para isso, este método requer três elementos: (i)

dados de casos indexados no tempo; (ii) uma função estimadora $\hat{S}$ a ser aplicada aos dados; e (iii) o nível de confiança da abordagem de predição ($c \in (0, 1)$).

A ideia é estimar o maior tempo de sobrevivência (T_S) com baixa probabilidade de revogação. O Algoritmo 3 apresenta as etapas para obter o tempo de sobrevivência considerando os parâmetros do momento da submissão de uma aplicação.

Algorithm 3 Processando T_S

```
1: procedure ProcessaTempoSobrevivencia(regiao, zona, tipoInstancia, nivelConfianca, diaSemana, horaDia)
2:     casos[] ← recuperarCasosSimilares(regiao, zona, tipoInstancia, diaSemana, horaDia)
3:     if processaTamanhoCasos(casos) ≤ confidenceSize then
4:         raise Exceção de tamanho mínimo requerido.
5:     timeCasos[] ← Integer(sizeof(casos[]))
6:     censurados[] ← Boolean(sizeof(casos[]))          ▷ Requerido pelo estimador Kaplan Meier
7:     for caso in casos index i do
8:         timeCasos[i] ← caso
9:         censurados[i] ← [caso.isCensurado() == True]
10:    sortedIntervalos ← KaplanMeierEstimator.compute(casos, censurados)
11:    primeiroIntervalo ← sortedIntervalos[0]
12:    for intervalo in sortedIntervalos do              ▷ Buscar intervalo mais próximo
13:        if intervalo.cumulativeSurvival ≥ nivelConfianca then
14:            primeiroIntervalo ← intervalo
       return primeiroIntervaloo.valor
```

Como apresentado no Algoritmo 3, o T_S é extraído da curva de sobrevivência. Através de um estimador, uma função de sobrevivência torna-se necessária para identificar a probabilidade de sobrevivência (não-falha) até o tempo t. O estimador de Kaplan-Meier Meeker and Escobar (1998) é usado para produzir a curva de sobrevivência, que representa as probabilidades de sobrevivência de acordo com diferentes níveis de confiança (Equação 4.3).

$$\hat{S}(t) = \prod_{x \leftarrow t_x \leq t_n}^{n} 1 - \frac{FALHA_x}{SOBREVIVENCIA_x} \tag{4.3}$$

Na Equação 4.3, c_x é a representação de um limite superior de um pequeno intervalo de tempo; $FALHA_i$ é o número de falhas dentro de um intervalo; e $SOBREVIVENCIA_x$ representa o número de indivíduos sob risco no intervalo específico.

Exemplos de curvas de sobrevivência com os respectivos tempos extraídos de acordo com diferentes níveis de confiança podem ser observados na Figura 2.13. Se nenhuma falha ocorrer em um determinado intervalo, a curva de sobrevivência não diminui. Diminuir o nível de confiança significa tempos de sobrevivência improváveis. Melhores resultados foram alcançados ao usar o nível de confiança 98% em nossos experimentos.

Para produzir uma curva de sobrevivência, casos semelhantes foram recuperados da base de casos do CBR, que é usado como entrada pelo agente Executor para prever o TAR

de acordo com um nível de confiança que fornece segurança suficiente. O agente calcula o maior Tempo de Sobrevivência (T_S) pelo qual tem-se alta confiança (98%) de que nossa instância em execução não será revogada, considerando o dia da semana e hora do dia.

Este tempo é definido como $T_S = \arg\max_t\{t \in \mathbb{R} \mid P(TTR > t) \geq 0.98\}$ e pode ser calculado como o 98^o percentil extraído da sua curva de sobrevivência. A curva de sobrevivência para δ_i é calculada a partir dos casos similares conforme apresentado na Equação 4.4.

$$w_{ij} = \frac{SimCasos(\delta_i, \delta_j)}{\sum_{k \neq i} SimCasos(\delta_i, \delta_k)} \text{ for } j \neq i \tag{4.4}$$

Dado o tempo de execução de uma aplicação (T_A) e um tempo de sobrevivência calculado, pode-se utilizar regras de sequenciamento de produção Davis et al. (1977), que determina a ordem que as operações serão executadas, para dar suporte às decisões do agente, conforme apresentado na Figura 4.9 e detalhado no Algoritmo 4.

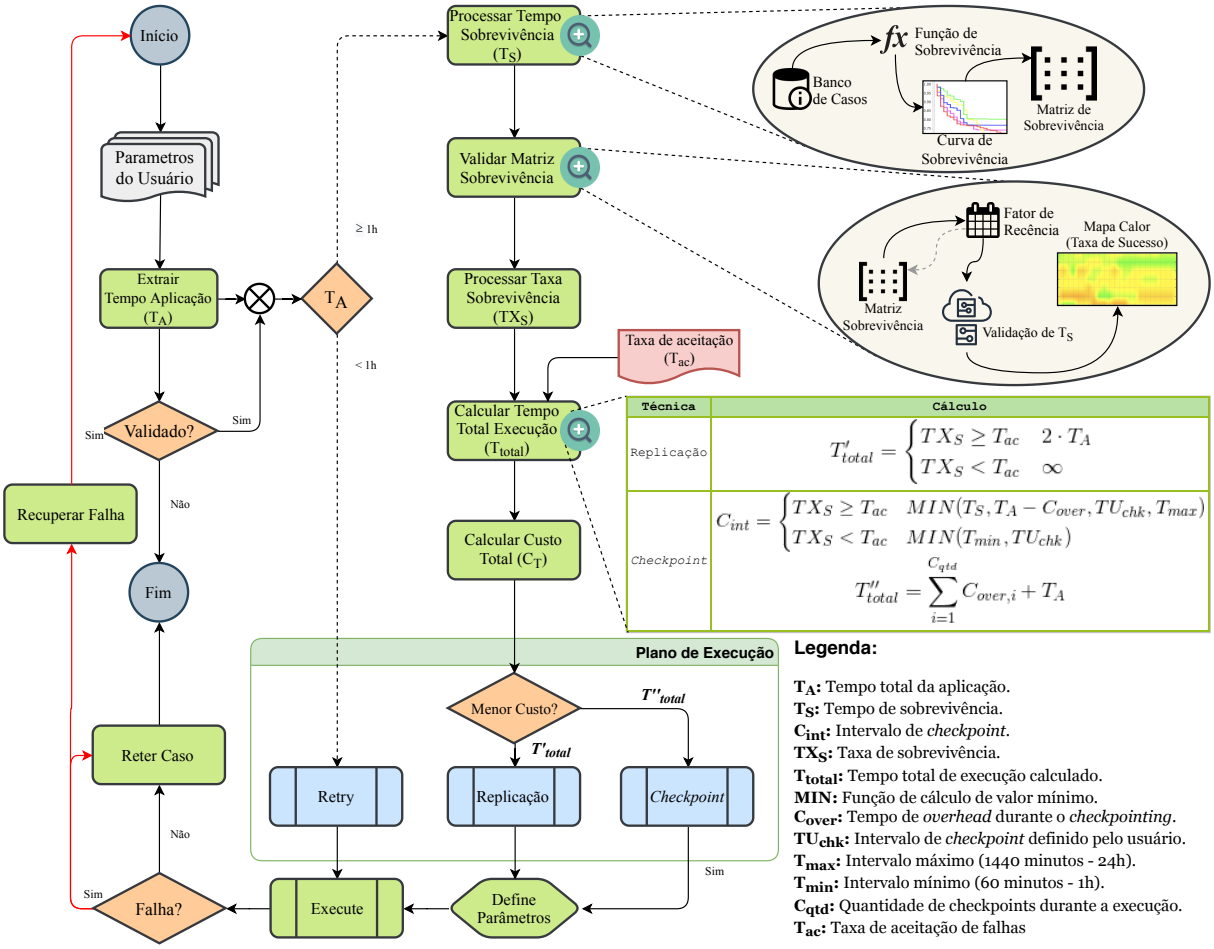


Figura 4.9: Processo de raciocínio do agente para definição do plano de execução.

Um parâmetro necessário (T_A) é um dado obrigatório e é usado na primeira etapa do processo: decidir qual técnica de TF será utilizada em uma nova submissão. Neste processo utiliza-se a definição presente em Voorsluys and Buyya (2012); Yi et al. (2010), que consideram longas execuções quando uma aplicação necessita de um tempo superior a uma hora (60 minutos), como maneira simples e intuitiva, porém eficaz, de lidar com o *trade-off* de custo/confiabilidade no uso de instâncias transientes, que possui ciclos de cobrança mínima neste intervalo de tempo.

O fluxo de decisão detalhado na Figura 4.9 e Algoritmo 4 considera a técnica de *retry* como o principal plano de execução a ser aplicado quando T_A for considerado curto, i.e. nenhum outro parâmetro torna-se necessário quando o tempo é inferior ou igual a 60 minutos ($T_A \leq 60$ minutos). Quando T_A é considerado longo ($T_A > 60$ minutos), a taxa de sobrevivência e seu sucesso são considerados parâmetros obrigatórios e são calculados para serem utilizados nos próximos passos.

Algorithm 4 Definindo o plano de execução apropriado.

```
1:  procedure DefinePlanoExecucao(tempoApp, regiao, zona, tipoInstancia, valorLance)
2:      Inicialização do Sistema
3:      dadosOK ← validaParametros(tempoApp, regiao, zona, tipoInstancia, valorLance)
4:      planoExecucaoMap ← initMap(tempoApp, regiao, zona, tipoInstancia, valorLance)
5:      if dadosOK then                                  ▷ Aplicação está apta para execução.
6:          if tempoApp > 60 then                                          ▷ Tempo em minutos
7:              diaSemana, horaDia ← recuperarTimeAtual()
8:              params ← [regiao, zona, instanceType, 0.98, diaSemana, horaDia]
9:              tempoSobrevivencia ← processarTempoSobrevivencia(params)          ▷ (Algoritmo 3)
10:             listaHistoricoPrecos ← recuperarHistoricoPrecos(90)              ▷ Tempo em dias
11:             taxaSobrevivencia ← 0
12:             for regPreco in listaHistoricoPrecos index i do                ▷ Cada registro de preço
13:                 if extrairInstante(regPreco) == diaSemana, horaDia then
14:                     precoAtual ← regPreco.precoAtual
15:                     erro ← False
16:                     instant ← regPreco.time
17:                     for instant increasing 60 do                           ▷ Tempo em minutos
18:                         if regPreco.preco > listaHistoricoPrecos[instant].preco then
19:                             erro ← True
20:                     if not erro then
21:                         taxaSobrevivencia + +
22:             tempoTotalReplicacao ← calcularTotalReplicacao()                      ▷ Equação 4.5
23:             intervaloCheckpoint ← calcularIntervaloCheckpoint()                   ▷ Equação 4.6
24:             tempoTotalCheckpoint ← calcularTotalCheckpoint()                      ▷ Equação 4.7
25:             planoExecucaoMap['TECNICA'] ← calcularPlano(tempoTotalReplicacao, tempoTotalCheckpoint)
26:             planoExecucaoMap['PARAMS'] ← calcularParametros(tempoTotalReplicacao, tempoTotalCheckpoint)
27:         else
28:             planoExecucaoMap['TECNICA'] ← modelo['RETRY']
29:     return PlanoExecucao(planoExecucaoMap)
```

Inicialmente, uma curva de sobrevivência é gerada considerando o instante atual, i.e. dia da semana e hora do dia no momento da submissão, e o TAR é extraído a partir do nível de confiança de 98%.

Cada relacionamento entre dia da semana e hora do dia possui sua própria curva de sobrevivência, associada a um tipo de instância transiente, sendo representada como uma Matriz de Tempo de Sobrevivência (MTS). A MTS representa uma matriz de tamanho

$7x24$ (7 dias da semana e 24 horas do dia) contendo os T_S processados para cada relação, conforme apresentado no Capítulo 5 e detalhado na Tabela 5.1.

Quando construída, a MTS é explorada para validar a taxa de sobrevivência (TX_S) considerando os valores de T_S presentes na MTS. Sucesso, neste caso, significa que o valor do T_S presente em cada relacionamento foi alcançado.

Exemplos de resultados desta validação são apresentados na Figura 5.7, que mostra o resultado através de um mapa de calor que representa o sucesso das taxas de sobrevivência estimadas.

Como próximo passo do processo, o tempo de execução total (T_{total}) previsto é calculado. Para replicação, usa-se a abordagem utilizada em Subramanya et al. (2015); Voorsluys (2014), que utilizam duas instâncias em replicações, considerando fator empírico no uso da TX_S, como observado na Equação 4.5, que calcula o tempo total da soma das duas replicações (T''_{total}) em instâncias distintas.

$$T'_{total} = \begin{cases} TX_S \geq 0.90 & 2 \cdot T_A \\ TX_S < 0.90 & \infty \end{cases} \tag{4.5}$$

Se o sucesso for alto ($TX_S \geq 90\%$), considera-se que o T_S seja alcançável e reflita em um cenário confiável. Para um cenário em que $TX_S < 90\%$, a definição do tempo como infinito (∞) reflete a não confiança da garantia de disponibilidade da VM durante o tempo T_A.

Para o cálculo do tempo total utilizando checkpoint, leva-se em consideração o uso da TX_S. Diante disso, um intervalo conveniente *checkpointing* é calculado da seguinte maneira: dado o tempo T_A; o tempo de *overhead* ao realizar o *checkpointing* C_{over}; o parâmetro de intervalo máximo (opcional) definido pelo usuário TU_{chk}; o tempo considerado longo, parametrizável pelo agente $T_{MAX} = 1440$ (24 horas); e o T_S, estimado pela MTS, um intervalo conveniente de *checkpointing* (C_{int}) pode ser obtido seguindo a Equação 4.6.

$$C_{int} = \begin{cases} TX_S \geq 0.90 & MIN(T_S,\ \ T_A - C_{over},\ \ TU_{chk},\ \ T_{max}) \\ TX_S < 0.90 & MIN(T_{min},\ \ TU_{chk}) \end{cases} \tag{4.6}$$

Semelhante à replicação, considera-se um cenário confiável quando $TX_S \geq 90\%$. Para níveis inferiores a 90%, o intervalo é definido a partir do tempo mínimo de cobrança (60 minutos) ou do tempo definido pelo usuário, de forma parametrizável.

O tempo total de execução (T''_{total}) é diretamente afetado pelo valor de C_{int}, utilizado para a definição da quantidade de *checkpoints* (C_{qtd}) que serão realizados durante a execução. Portanto, o T_{total} de execução do checkpoint é definido conforme Equação 4.7.

$$T''_{total} = \sum_{i=1}^{C_{qtd}} C_{over,i} + T_A \tag{4.7}$$

Para o cálculo do custo de replicação (CCS_{rep}), é verificada a existência e disponibilidade de outra instância do mesmo tipo em outra zona e na mesma região. Escolher uma combinação diferente de tipos de instâncias, incluindo zonas distintas para uma réplica é uma escolha óbvia, pois optando por n execuções replicadas na mesma região, zona e usando o mesmo tipo de instâncias, todas as n execuções falharão ao mesmo tempo se ocorrer uma falha.

A escolha da segunda instância é baseada na proximidade de seu preço ou zona, considerando um nível seguro da TX_S. No escopo deste trabalho, a estratégia de duas replicações é utilizada, conforme proposto em (Subramanya et al., 2015; Voorsluys and Buyya, 2012). O valor de CCS_{rep}, que inclui o uso de uma instância que está em outra zona, o valor do conjunto de instâncias é utilizado na definição do seu custo.

Para o cálculo do custo de checkpoint (CCS_{chk}), o valor é obtido através do produto do tempo total (Equação 4.7) e o preço da instância transiente, considerando a zona escolhida. Após o cálculo de previsão do CCS_{rep} e CCS_{chk}, o plano de execução é definido a partir de $MIN(CCS_{rep}, CCS_{chk})$, que calcula o mínimo valor entre eles.

Na definição dos parâmetros, no escopo deste trabalho, a replicação se dá através de duas execuções em zonas distintas na mesma região. Para o uso *checkpoint*, o parâmetro que indica o intervalo de *checkpointing* é definido através do resultado da Equação 4.6.

O principal objetivo desta abordagem baseada em agentes é observar o ambiente real, encontrar um plano de execução tolerante a falhas e definir seus parâmetros de forma que minimizem o tempo total de execução e reduzam custos do usuário. Para atingir essas metas, esta proposta usa a abordagem apresentada em Araujo Neto et al. (2018); Araujo Neto et al. (2018), que usa CBR, análise de sobrevivência e quantis da TAR para encontrar um tempo de sobrevivência e usá-lo como parte de uma estratégia para evitar e mitigar a ocorrência de falhas.

Capítulo 5

Experimentos

Neste capítulo a proposta de uso eficiente de instâncias transientes spot será examinada por meio de experimentos em ambiente simulado e real, envolvendo as seguintes técnicas de TF: reinício da tarefa (*retry*), *checkpoint/restore* e replicação (*replication*).

Na Seção 5.1 será avaliado o modelo de predição utilizado nos experimentos. Na Seção 5.2 serão apresentados os resultados realizados em ambiente simulado. Enquanto na Seção 5.3 serão apresentados os resultados em ambiente real.

5.1 Validação do Modelo de Predição

Para avaliar as decisões dos agentes foi utilizado o histórico de alterações de preços das instâncias *spot* recuperados durante esta pesquisa. Por meio destes registros e o Algoritmo 1 foi gerada uma base de casos com os TARs para ser utilizada como entrada no processamento dos T_S presentes na MTS.

Como apresentado na Seção 4.3, o T_S é um importante elemento utilizado para a definição do plano apropriado de execução tolerante a falhas, aplicando os parâmetros de acordo com o detalhamento apresentado na Tabela 4.6. Assume-se que o usuário pretende executar a aplicação no instante da submissão, sem o envolvimento de abordagens de agendamento de execução.

Para avaliar a assertividade nos T_S presentes na MTS em um ambiente simulado, foram utilizados 76 instâncias, incluindo cinco zonas diferentes (*us-west-1b, us-west-1c, us-west-2a, us-west-2b e us-west-2c*), das regiões *US-WEST-1* e *US-WEST-2*, durante o período Abril/2017 a Fevereiro/2019 (vide Figura 5.1). Apesar de existirem mais tipos de instâncias no ambiente do provedor da Amazon, optou-se por utilizar os tipos de instâncias nas quais havia registros de alteração de preço desde o início da amostra (Abril/2017).

O cenário completo de validação do T_S simulado ilustrado na Figura 5.1 inclui:

- 23 meses de alterações de preços reais, coletados da Amazon AWS durante o período de Abril/2017 a Fevereiro/2019, com aproximadamente 21 milhões de registros;
- 2 Regiões (*US-WEST-1* e *US-WEST-2*) em um cenário que inclui 26 semanas, simulando todas as relações entre dias da semana e hora do dia;
- 6 estratégias de lance: (i) valor atual da instância *spot*; (ii) média dos últimos 7 dias; (iii) mediana dos últimos 7 dias; (iv) média do último dia; (v) mediana do último dia; e (vi) valor da instância *on-demand*; e
- 76 instâncias totalizando 9.959.040 simulações.

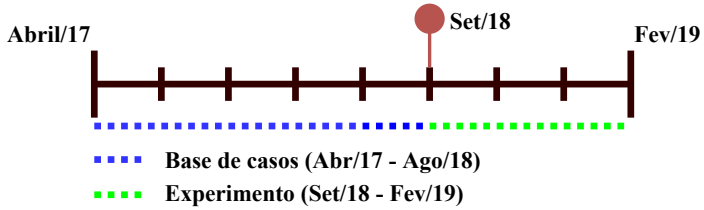


Figura 5.1: Cenário do experimento simulado (Setembro/2018 a Fevereiro/2019) na avaliação da MTS (Abril/2017 a Agosto/2018).

Para definir a faixa ideal de dados a ser usada na geração de casos e na estimativa dos T_S, foi calculada a taxa de sucesso em função do número de meses a serem incluídos. Dada a crescente taxa de sucesso à medida que mais observações são usadas, foram utilizados todos os dados coletados, conforme apresentado na Figura 5.2.

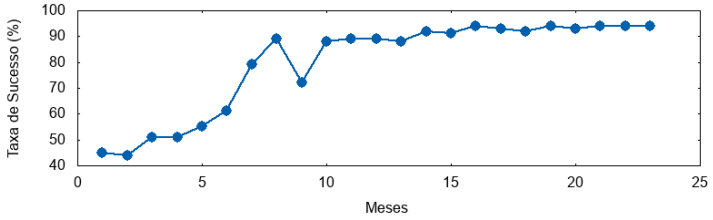


Figura 5.2: Taxa de sucesso (%) de acordo com a quantidade de dados coletados no experimento (meses).

Observa-se que há uma tendência geral de aumentar o sucesso à medida que o número de meses aumenta. Mas, esse resultado é esperado desde que o processo de geração de dados não tenha mudado significativamente ao longo do período analisado. Desta forma, a presença de mais dados reflete em uma melhor precisão da curva de sobrevivência, levando a melhores estimativas de T_S.

Observe também que há uma queda significativa na taxa de sucesso quando o período de nove meses foi utilizado. Isso ocorreu devido à introdução de novos tipos de instâncias com poucas observações na base de dados coletada, levando a estimativas ruins e a maiores falhas para essas instâncias, reduzindo a taxa de sucesso do modelo de predição.

A partir dos dados coletados foram gerados 356.187.177 casos usando o cenário detalhado na Figura 5.1. Um conjunto de casos foi definido aplicando a Equação 4.1 e o Algoritmo 1. Cada relacionamento entre o dia da semana e hora do dia possui sua própria curva de sobrevivência, sendo representada por uma MTS, conforme ilustrado na Tabela 5.1. A MTS apresenta o T_S em minutos, obtidos com a definição de 98^o de nível de confiança e extraídos de suas respectivas curvas de sobrevivência. A coluna vertical é composto por números que representam o dia da semana (i.e., 1 domingo, 2 segunda-feira, ..., 7 sábado), e a coluna horizontal representa a hora do dia (0, 1, 2, ..., 23).

Tabela 5.1: MTS gerada para a instância *r4.16xlarge*, com marcação na T_S extraído da curva de sobrevivência apresentada na Figura 5.3.

Dia da Semana	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	263	203	143	83	1251	1191	1176	1090	1042	970	891	831	460	723	670	610	576	694	636	576	514	456	395	315
2	280	569	349	256	404	363	313	255	195	135	92	276	215	155	96	71	585	524	465	402	344	284	233	173
3	114	338	348	309	253	193	157	159	119	80	888	820	664	700	649	794	756	674	614	588	516	480	420	353
4	343	304	224	184	215	175	117	167	158	100	192	390	333	295	238	245	187	130	74	62	394	334	297	223
5	163	184	247	195	500	472	425	434	374	314	182	117	**347**	133	126	742	687	627	572	512	447	387	599	539
6	461	293	223	323	238	206	172	252	410	350	317	415	355	276	235	192	135	284	424	364	300	252	188	128
7	109	619	558	498	446	391	333	289	229	159	113	800	494	685	852	795	735	680	612	555	500	438	378	318

Hora do Dia

Os valores presentes na MTS são os resultados de casos semelhantes recuperados pela função de similaridade em um banco de casos, conforme apresentados nas Equações 4.3 e 4.4, calculadas a partir dos melhores resultados experimentais para o nível de confiança variando entre 90%, 95% e 98%. Um exemplo de uma curva se sobrevivência que representa a marcação na Tabela 5.1 pode ser observada na Figura 5.3.

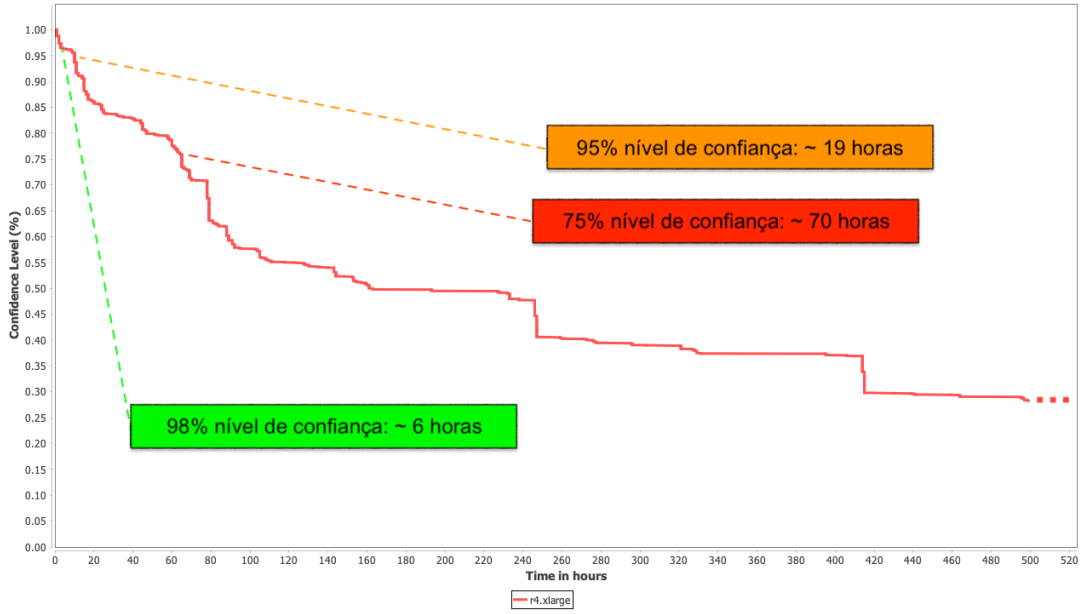


Figura 5.3: Curva de sobrevivência (quinta-feira às 12:00 GMT) da instância *r4.16xlarge* com seus respectivos valores de níveis de confiança.

Para avaliar os T_S da MTS, um conjunto de experimentos foi criado para comparar os valores presentes na MTS em cenários que envolvem o uso de dados reais de alterações de preços. Cada valor foi comparado considerando um ambiente de leilão, onde para cada T_S extraído da relação entre dia da semana e hora do dia, quando alcançável por meio da simulação tem seu valor incrementado em um, caso contrário o valor é acrescido de zero.

Este valor representa o TX_S (Figura 4.9) e uma nova matriz de mesmo tamanho é gerada com os resultados destes experimentos. Como exemplo, apresenta-se a instância *r4.16xlarge* com os resultados mostrados na Figura 5.4. As áreas verdes indicam que os valores de T_S foram alcançados em um cenário real de leilão. A taxa de sucesso alcançada na Figura 5.4 é de aproximadamente 81% com um desvio padrão de 1,56 pontos percentuais.

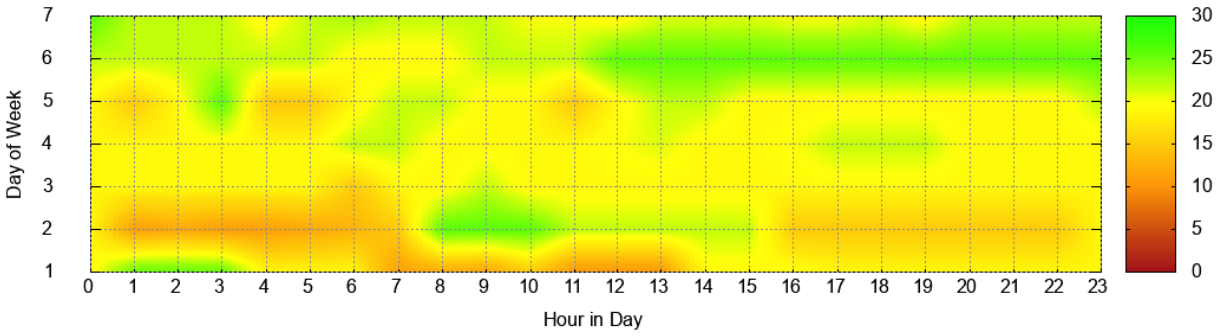


Figura 5.4: Mapa de calor representando os resultados de sucesso na comparação dos T_S na instância *r4.16xlarge*.

Semelhante à MTS, o Eixo Y é composto por números que representam o dia da semana (1 domingo, 2 segunda-feira, ..., 7 sábado) e o Eixo X representa a hora do dia (0, 1, 2, ..., 23). Considerando as simulações com as instâncias envolvidas, a taxa de sucesso sobe para aproximadamente 84%. Isso ocorre porque algumas instâncias com custos monetários elevados apresentam variações estáveis, permitindo longos T_S.

Analisando os resultados apresentados na Figura 5.4, observa-se duas lacunas em destaques: durante o domingo (1) entre às 4h e 13h, com uma média de 12 falhas em 26 tentativas; e na segunda-feira (2) entre às 0h e 7h, com uma média de 15 falhas em 26 tentativas. Isso demonstra que é necessário adotar outra estratégia para alcançar melhores taxas de sucesso e produzir valores de T_S mais confiáveis.

Desta forma, uma estratégia diferente foi incorporada como complemento à função que gera a curva de sobrevivência (Equação 4.3) para ajustar os valores da MTS, aumentar a precisão dos dados e obter melhores resultados. Essa estratégia atribui um peso maior aos casos gerados mais recentes (fator de recência), conforme apresentado na Figura 5.5.

Conforme ilustrado na Figura 5.5, nessa estratégia, o intervalo usado no experimento é reduzido para 16 meses, de Abril/2017 a Julho/2018 e um novo período de validação

(4 semanas) é adicionado para aumentar a acurácia de nossa abordagem ao considerar dados mais recentes. Experimentos mostram que aumentar o período de validação diminui o sucesso na sobrevivência e esse comportamento está relacionado à distância entre as variações de preço analisadas e a análise do instante atual. Distâncias maiores implicam em considerar dados mais distantes.

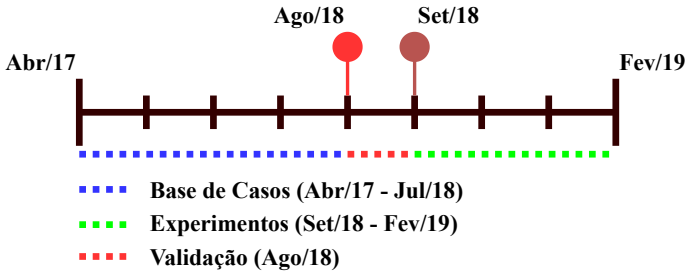


Figura 5.5: Cenário de experimento considerando fator de recência.

Para refinar os valores presentes na MTS e incluir um fator de recência para os T_S, as últimas 4 semanas (Agosto/2018) foram usadas para alcançar melhores T_S, criando um novo subconjunto de validação dos casos e calculando o tempo médio entre eles para reproduzir um novo cenário recente. Com esse fator de recência, uma nova MTS é gerada (MTS'), representando novos valores na relação entre dias da semana e horas do dia com T_S que consideram dados recentes. A MTS' gerada é apresentada na Tabela 5.2 com novos valores em comparação aos T_S presentes na Tabela 5.1.

Tabela 5.2: Uma nova MTS' considerando recência em seus T_S.

Dia da Semana \ Hora do Dia	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	133	99	134	60	354	391	416	500	292	190	291	283	246	272	199	209	436	236	196	233	114	233	198	123
2	80	99	89	110	121	102	121	121	98	87	92	67	103	100	96	81	135	200	189	290	145	111	122	173
3	121	121	98	87	92	67	103	100	96	81	80	92	67	103	93	88	113	99	231	234	190	222	199	173
4	92	67	103	93	88	113	99	231	158	78	99	121	98	87	92	67	103	74	62	94	112	132	133	99
5	103	100	96	81	135	200	189	290	145	111	82	60	60	61	69	118	227	211	133	132	166	139	200	177
6	103	93	88	113	99	231	234	190	222	130	188	133	99	100	111	109	88	84	79	118	227	211	133	91
6	69	118	227	211	133	132	166	139	109	391	123	289	294	152	300	277	231	421	312	222	131	244	145	144

Após processar novos T_S e atualizar os valores na MTS', os experimentos apresentaram melhores resultados e podem ser observados no mapa de calor na Figura 5.6.

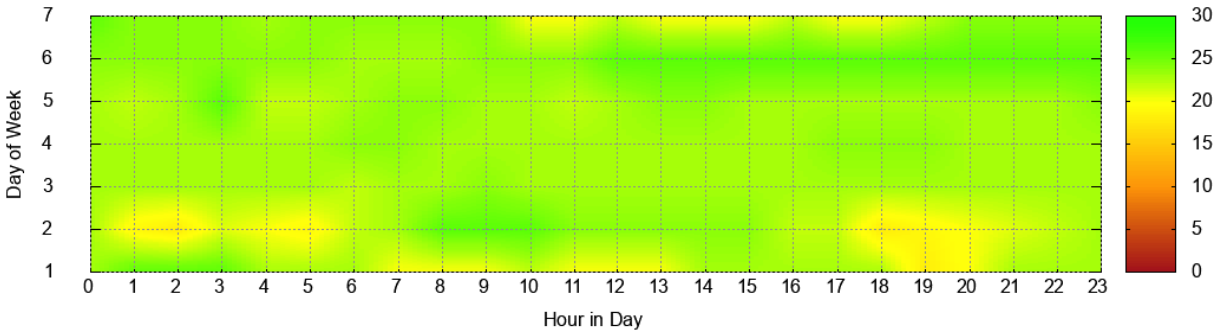
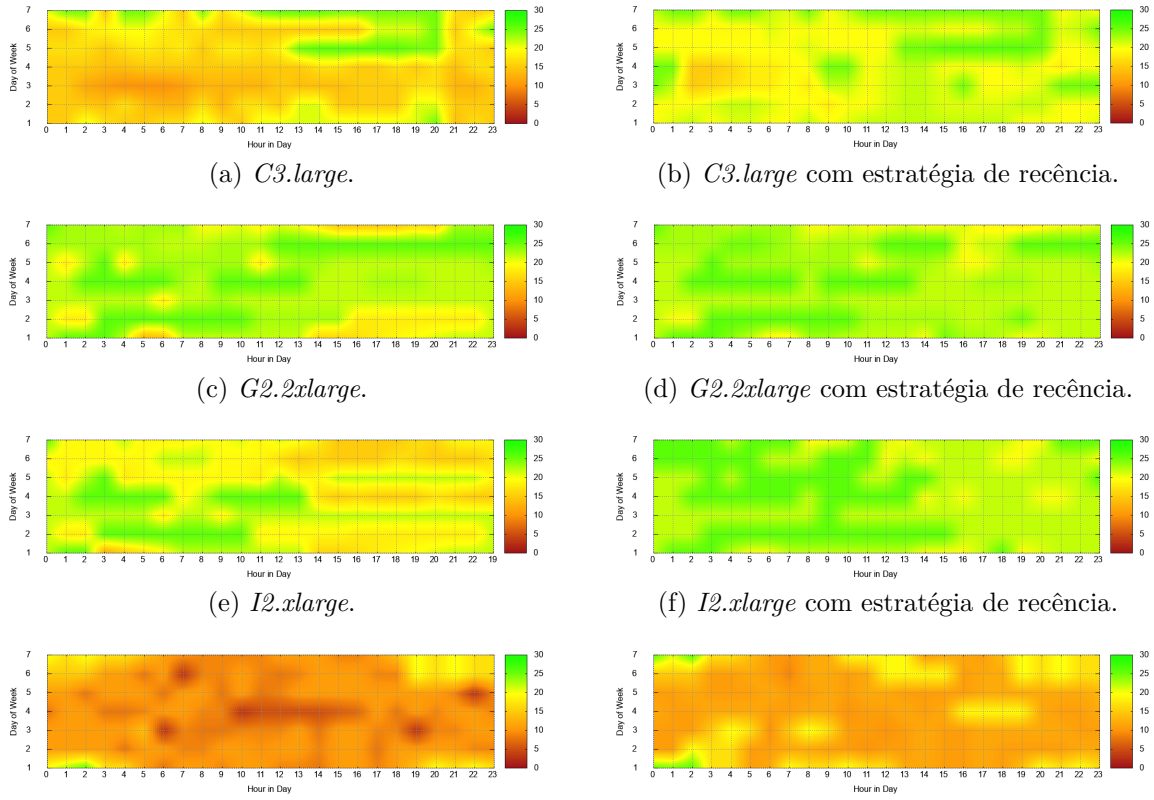


Figura 5.6: Mapa de calor na instância *r4.16xlarge* após a estratégia de recência.

Comparado aos resultados apresentados na Figura 5.4, foi alcançado um ganho de 10% após usar a estratégia de recência, atingindo uma taxa de sucesso de 91% nas mesmas condições de dados do experimento.

Considerando que o provedor da Amazon AWS fornece uma ampla seleção de tipos de instâncias *spot*, otimizadas para atender a diferentes casos de uso, oferecendo aos usuários a flexibilidade de escolher a combinação apropriada de recursos para suas aplicações, um conjunto de tipos de instâncias foi usado nos experimentos e um subconjunto é apresentado na Figura 5.7.



(a) *C3.large*. (b) *C3.large* com estratégia de recência.

(c) *G2.2xlarge*. (d) *G2.2xlarge* com estratégia de recência.

(e) *I2.xlarge*. (f) *I2.xlarge* com estratégia de recência.

(g) *M4.xlarge*. (h) *M4.xlarge* com estratégia de recência.

Figura 5.7: Resultados em mapa de calor considerando tipos de instância distintos.

Uma compilação com os resultados obtidos pelos experimentos apresentados na Figura 5.7 pode ser observado na Tabela 5.3.

Tabela 5.3: Taxa de sucesso (%) obtidas dos experimentos apresentados na Figura 5.7.

	(a & b)	(c & d)	(e & f)	(g & h)
Estratégia original	73%	88%	80%	32%
Estratégia usando Recência	89%	91%	91%	49%
Ganhos totais	16%	6%	11%	17%

Na Figura 5.7, um tipo de instância otimizada para computação foi usada em (a) e (b) (*C3.large*). Normalmente essa instância é procurada por usuários que possuem aplicações que necessitam de altas cargas de trabalho, oferecendo alto desempenho com boa relação custo-benefício e baixo preço, devida a sua alta capacidade computacional. Os resultados de uma instância de computação acelerada com recursos de GPU otimizados para aplicações com uso intenso de cálculos gráficos podem ser encontrados nos exemplos (c) e (d) (*G2.2xlarge*).

Os resultados mostrados em (e) e (f) representam uma instância otimizada para armazenamento com baixa latência e desempenho de Entrada/Saída aleatória muito alta (*I2.xlarge*). Por fim, (g) e (h) representam uma instância de uso geral que fornece um equilíbrio de computação com memória e recursos de rede (*M4.xlarge*).

Observando os resultados para (g) e (h), percebe-se que mesmo usando a estratégia de recência, apenas um pequeno ganho foi obtido, de 32% a 49%. Observando o seu comportamento no cenário deste experimento, esse tipo de instância possui mudanças consideráveis de preço no período utilizado, exigindo uma estratégia diferente que considera padrões mais recentes de mudança de preço. Comparado a Figura 5.7 (h), foram obtidos melhores resultados quando a estratégia de recência utilizou os últimos 7 dias como período de validação, conforme observado na Figura 5.8.

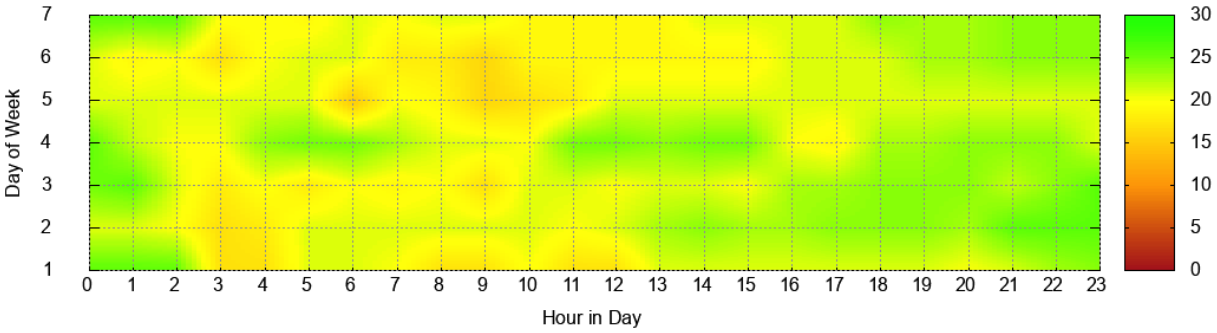


Figura 5.8: Novos resultados da instância *M4.xlarge* considerando o fator de recência nos últimos 7 dias.

5.2 Ambiente Simulado

Dado os resultados alcançados na validação do modelo de predição (Seção 5.1), assumiu-se que os T_S são confiáveis e podem ser extraídos da MTS para uso na definição da técnica de TF com seus respectivos parâmetros (vide Figura 4.9).

Nesta Seção, os experimentos envolvem o uso de *checkpoint/restore* e o cálculo de seus parâmetros, i.e. intervalo e quantidade de *checkpoints*, na comparação com outras abordagens da literatura. A importância do experimento de *checkpoint/restore* comparada

a replicação está relacionada a diferentes abordagens na definição dos seus parâmetros. Para a técnica de replicação está sendo utilizada a mesma abordagem proposta em Subramanya et al. (2015); Voorsluys and Buyya (2012), que consiste nas execuções em duas VMs distintas.

Em relação ao uso da técnica de *checkpoint/restore*, comumente utilizada quando T_A é considerado longo, e considerando uma aplicação de uso intensivo de CPU, dados ou memória, que requer um tempo T_A, são comparadas diferentes abordagens na literatura em relação ao tempo total de execução (Figura 5.9), definição do intervalo de *checkpointing* (Equação 4.6), que reflete no número total de *checkpoint* e no custo total do usuário (Figura 5.10).

Os cenários dos experimentos no ambiente simulado estão distribuídos da seguinte forma: em (1) e (2) os intervalos são definidos usando uma abordagem fixa $C_{int} = 60$ minutos (Voorsluys and Buyya, 2012; Yi et al., 2010; Zhou et al., 2017); em (3) é definido o valor fixo de $C_{int} = 60 - C_{over}$, considerando os custos indiretos de *overhead* no momento em que o estado de execução é salvo (Voorsluys and Buyya, 2012); (4_a), (4_b) e (4_c) são utilizados intervalos fixos de $C_{int} = 10$, $C_{int} = 15$ e $C_{int} = 20$ minutos, respectivamente (Subramanya et al., 2015); em (5_a) e (5_b), os autores consideram cada alteração de preço como um gatilho para um novo *checkpoint* (Yi et al., 2012), usando 96 e 145, respectivamente, recuperados por registros reais do banco de dados de alteração de preço. Em caso de exceder o limite do *checkpoint*, o valor máximo é utilizado; por fim, (6) utiliza $C_{int} = 30$ como um intervalo fixo (Zhou et al., 2017).

Em (7), são apresentados os resultados alcançados utilizando a abordagem deste trabalho, considerando diferentes cenários de T_S: (7_a) $T_S = T_A$; (7_b) com metade do tempo total $T_S = T_A/2$; (7_c) avaliando um novo tempo após o *checkpointing* usando o mesmo $T_S = T_A/2$; (7_d) um valor inferior e próximo ao tempo total $T_S = T_A - C_{over}$; (7_e) um valor superior e próximo do tempo total $T_S = T_A + C_{over}$; e por fim, (7_f) com um tempo consideravelmente superior $T_S = 2 \cdot T_A$ (o dobro).

Como explicado na Seção 4.3, o número de *checkpoints* e o tempo de *overhead* afetam o tempo total de execução de uma aplicação (Equação 4.7). Considerando um cenário em que $T_A = 900$ minutos, $C_{int} = T_A$ e diferentes tempos de C_{over}, observa-se os seguintes resultados em relação ao tempo total de execução (Figura 5.9), total de *checkpoints* e custo total (Figura 5.10) comparado a Subramanya et al. (2015); Voorsluys and Buyya (2012,?); Yi et al. (2012, 2010); Zhou et al. (2017).

Considerando os diversos cenários apresentados nas Figuras 5.9 e 5.10, observa-se que: nos casos $7_a, 7_c, 7_d, 7_e, 7_f$, apenas um *checkpoint* ($C_{qtd} = 1$) foi necessário; e em 7_b, um total de dois *checkpoints* ($C_{qtd} = 2$). Em ambos os casos, o cálculo do intervalo e do tempo total de execução se dá por meio das Equações 4.6 e 4.7, respectivamente.

Observando as Figuras 5.9 e 5.10, que consideram aplicações de longa execução, comparados a Subramanya et al. (2015); Voorsluys and Buyya (2012,?); Yi et al. (2012, 2010); Zhou et al. (2017), a abordagem desta proposta alcança um ganho de até 44,30% quando $C_{over} = 5$ (6_b), 49,44% com $C_{over} = 10$ (4_c, 6_a e 5_b), e o ganho pode chegar até 74,16% na redução do tempo quando $C_{over} = 30$ (4_c). A cobrança está diretamente relacionada ao tempo de consumo dos recursos, portanto a diminuição dos tempos reflete nos custos do usuário.

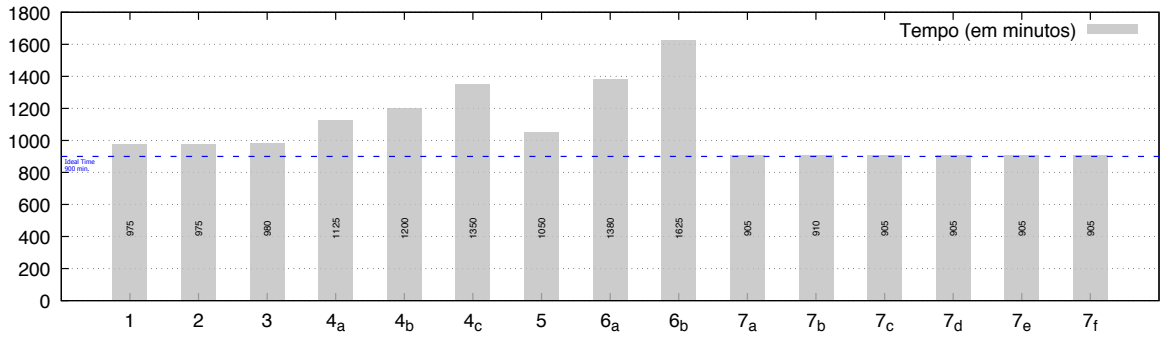
Para representar tarefas com tempos mais curtos, diferentes valores de T_A e C_{over} foram usados para comparar as abordagens em diferentes cenários e os resultados são apresentados na Figura 5.11.

Resultados mais completos destes experimentos, compostos por diferentes cenários de T_A e C_{over}, podem ser observados nas Tabelas 5.4, 5.5 e 5.6, que mostram valores para a quantidade de *checkpoints* calculados, o tempo total de execução e custo total, respectivamente. Observa-se que melhores resultados são obtidos nas estratégias desta proposta em relação a quantidade de *checkpoints*, tempo total e custo total.

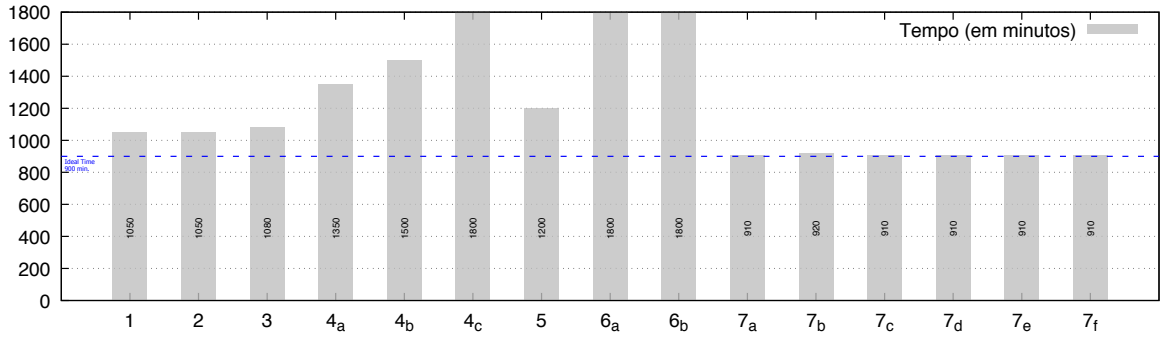
Tabela 5.4: Resultados de quantidades de *checkpoints* calculados para diferentes cenários.

Estratégia	$T_A = 300$			$T_A = 400$			$T_A = 900$			$T_A = 1260$			$T_A = 1440$		
	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$
1	5	5	5	6	6	6	15	15	15	21	21	21	24	24	24
2	5	5	5	6	6	6	15	15	15	21	21	21	24	24	24
3	5	6	10	7	8	13	16	18	30	22	25	42	26	28	48
4_a	15	15	15	20	20	20	45	45	45	63	63	63	72	72	72
4_b	20	20	20	26	26	26	60	60	60	84	84	84	96	96	96
4_c	30	30	30	40	40	40	90	90	90	126	126	126	144	144	144
5	10	10	10	13	13	13	30	30	30	42	42	42	48	48	48
6_a	60	30	10	80	40	13	96	90	30	96	96	42	96	96	48
6_b	60	30	10	80	40	13	145	90	30	145	126	42	145	144	48
7_a	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
7_b	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
7_c	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
7_d	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
7_e	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
7_f	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**

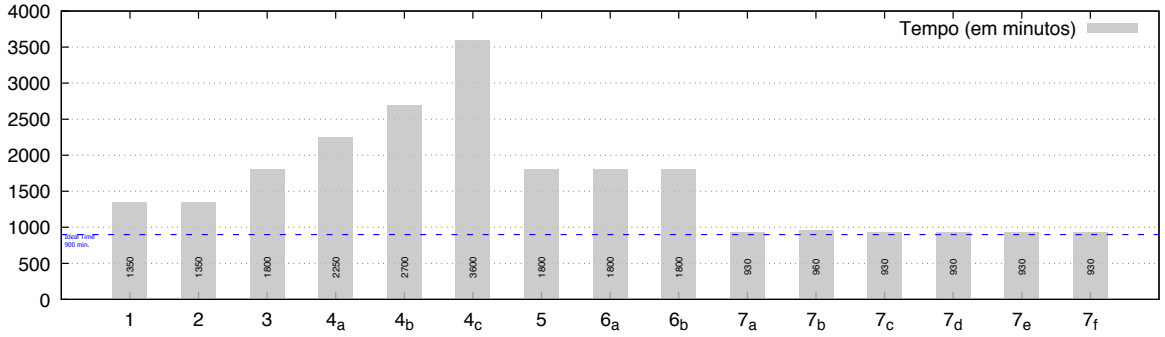
A Tabela 5.4 apresenta resultados de quantidade de *checkpoints* calculados, refletindo diretamente nos valores apresentados nas Tabelas 5.5 e 5.6.



(a) Considerando $C_{over} = 5$.

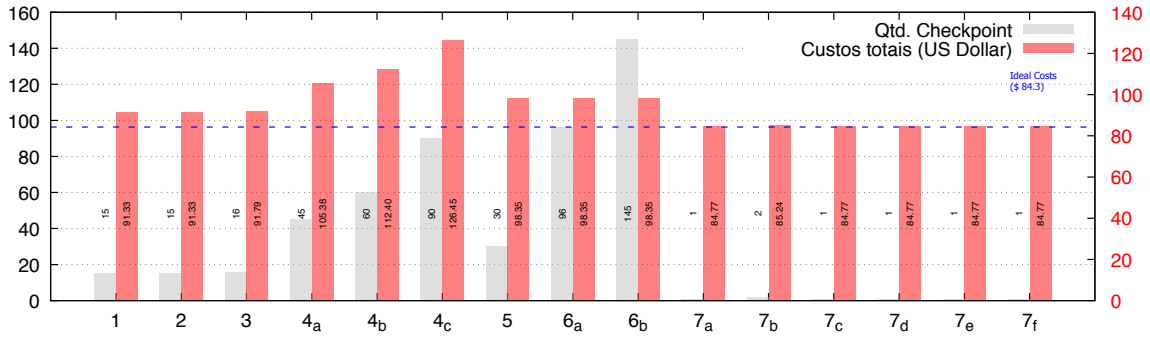


(b) Considerando $C_{over} = 10$.

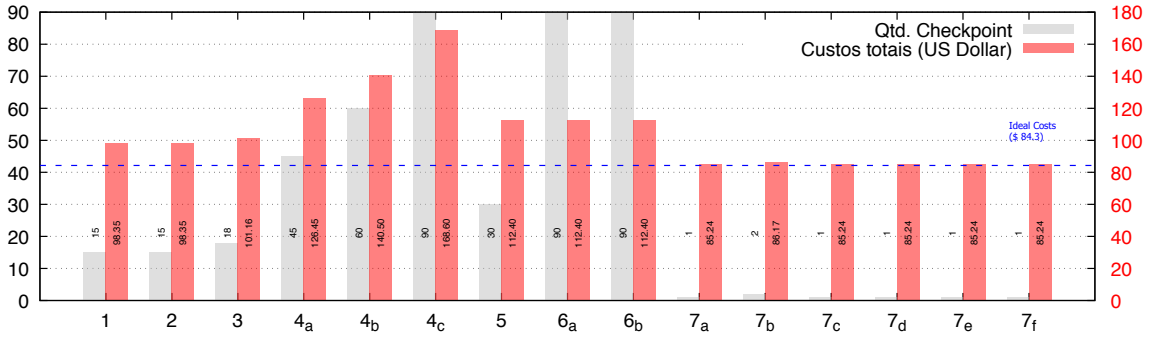


(c) Considerando $C_{over} = 30$.

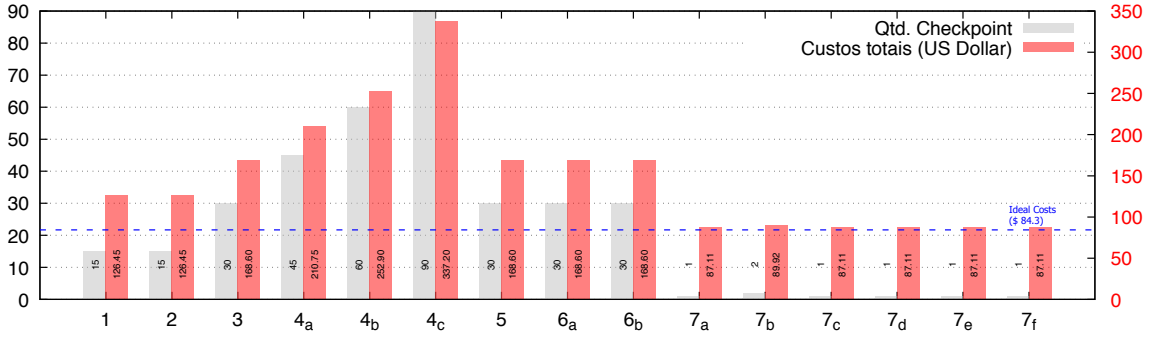
Figura 5.9: Comparativo na redução no tempo total de execução.



(a) Considerando $C_{over} = 5$.

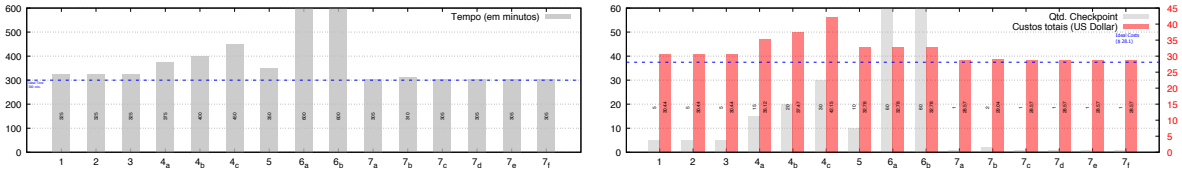


(b) Considerando $C_{over} = 10$.

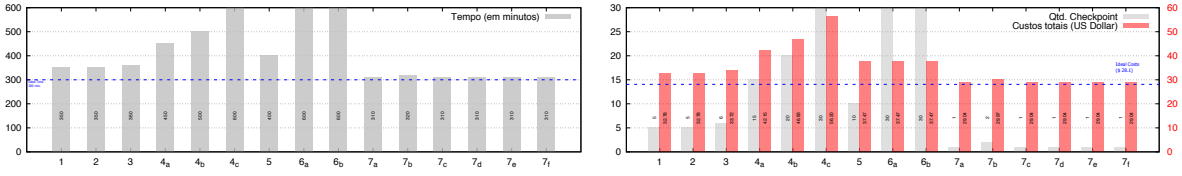


(c) Considerando $C_{over} = 30$.

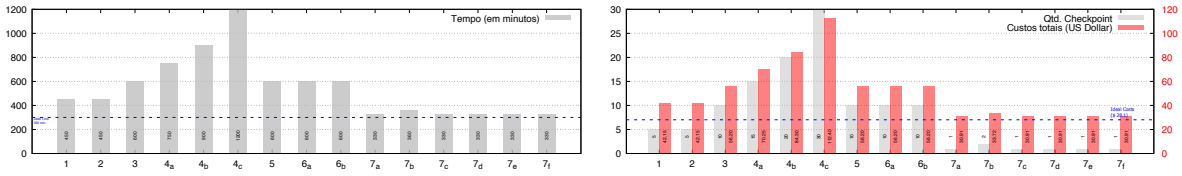
Figura 5.10: Comparativo na redução em quantidade de *checkpoints* e custo total.



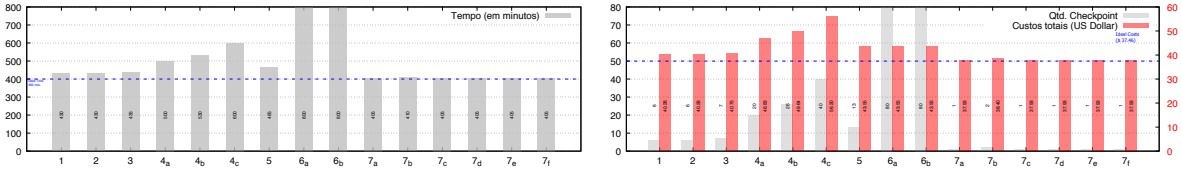
(a) $T_A = 300$ e $C_{over} = 5$



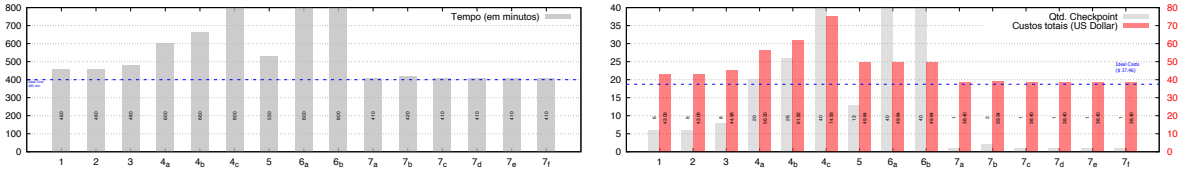
(b) $T_A = 300$ e $C_{over} = 10$



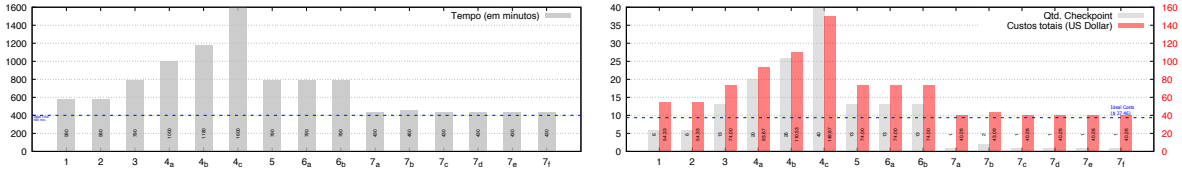
(c) $T_A = 300$ e $C_{over} = 30$



(d) $T_A = 400$ e $C_{over} = 5$



(e) $T_A = 400$ e $C_{over} = 10$



(f) $T_A = 400$ e $C_{over} = 30$

Figura 5.11: Resultados com diferentes cenários para T_A e C_{over}.

Tabela 5.5: Resultados de tempo total consumido para diferentes cenários.

Estratégia	$T_A = 300$			$T_A = 400$			$T_A = 900$			$T_A = 1260$			$T_A = 1440$		
	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$
1	325	350	450	430	460	580	975	1050	1350	1365	1470	1890	1560	1680	2160
2	325	350	450	430	460	580	975	1050	1350	1365	1470	1890	1560	1680	2160
3	325	360	600	435	480	790	980	1080	1800	1370	1510	2520	1570	1720	2880
4_a	375	450	750	500	600	1000	1125	1350	2250	1575	1890	3150	1800	2160	3600
4_b	400	500	900	530	660	1180	1200	1500	2700	1680	2100	3780	1920	2400	4320
4_c	450	600	1200	600	800	1600	1350	1800	3600	1890	2520	5040	2160	2880	5760
5	350	400	600	465	530	790	1050	1200	1800	1470	1680	2520	1680	1920	2880
6_a	600	600	600	800	800	790	1380	1800	1800	1740	2220	2520	1920	2400	2880
6_b	600	600	600	800	800	790	1625	1800	1800	1985	2520	2520	2165	2880	2880
7_a	**305**	**310**	**330**	**405**	**410**	**430**	**905**	**910**	**930**	**1265**	**1270**	**1290**	**1445**	**1450**	**1470**
7_b	310	320	360	410	420	460	910	920	960	1270	1280	1320	1450	1460	1500
7_c	**305**	**310**	**330**	**405**	**410**	**430**	**905**	**910**	**930**	**1265**	**1270**	**1290**	**1445**	**1450**	**1470**
7_d	**305**	**310**	**330**	**405**	**410**	**430**	**905**	**910**	**930**	**1265**	**1270**	**1290**	**1445**	**1450**	**1470**
7_e	**305**	**310**	**330**	**405**	**410**	**430**	**905**	**910**	**930**	**1265**	**1270**	**1290**	**1445**	**1450**	**1470**
7_f	**305**	**310**	**330**	**405**	**410**	**430**	**905**	**910**	**930**	**1265**	**1270**	**1290**	**1445**	**1450**	**1470**

Tabela 5.6: Resultados de custo total consumido (em dólar US) para diferentes cenários.

Estratégia	$T_A = 300$			$T_A = 400$			$T_A = 900$			$T_A = 1260$			$T_A = 1440$		
	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$	$C_{over} = 5$	$C_{over} = 10$	$C_{over} = 30$
1	30.44	32.78	42.15	40.28	43.09	54.33	91.33	98.35	126.45	127.86	137.69	177.03	146.12	157.36	202.32
2	30.44	32.78	42.15	40.28	43.09	54.33	91.33	98.35	126.45	127.86	137.69	177.03	146.12	157.36	202.32
3	30.44	33.72	56.20	40.74	44.96	74.00	91.79	101.16	168.60	128.32	141.44	236.04	147.06	161.11	269.76
4_a	35.12	42.15	70.25	46.83	56.20	93.67	105.38	126.45	210.75	147.53	177.03	295.05	168.60	202.32	337.20
4_b	37.47	46.83	84.30	49.64	61.82	110.53	112.40	140.50	252.90	157.36	196.70	354.06	179.84	224.80	404.64
4_c	42.15	56.20	112.40	56.20	74.93	149.87	126.45	168.60	337.20	177.03	236.04	472.08	202.32	269.76	539.52
5	32.78	37.47	56.20	43.55	49.64	74.00	98.35	112.40	168.60	137.69	157.36	236.04	157.36	179.84	269.76
6_a	56.20	56.20	56.20	74.93	74.93	74.00	129.26	168.60	168.60	162.98	207.94	236.04	179.84	224.80	269.76
6_b	56.20	56.20	56.20	74.93	74.93	74.00	152.21	168.60	168.60	185.93	236.04	236.04	202.79	269.76	269.76
7_a	**28.57**	**29.04**	**30.91**	**37.94**	**38.40**	**40.28**	**84.77**	**85.24**	**87.11**	**118.49**	**118.96**	**120.83**	**135.35**	**135.82**	**137.69**
7_b	29.04	29.97	33.72	38.40	39.34	43.09	85.24	86.17	89.92	118.96	119.89	123.64	135.82	136.75	140.50
7_c	**28.57**	**29.04**	**30.91**	**37.94**	**38.40**	**40.28**	**84.77**	**85.24**	**87.11**	**118.49**	**118.96**	**120.83**	**135.35**	**135.82**	**137.69**
7_d	**28.57**	**29.04**	**30.91**	**37.94**	**38.40**	**40.28**	**84.77**	**85.24**	**87.11**	**118.49**	**118.96**	**120.83**	**135.35**	**135.82**	**137.69**
7_e	**28.57**	**29.04**	**30.91**	**37.94**	**38.40**	**40.28**	**84.77**	**85.24**	**87.11**	**118.49**	**118.96**	**120.83**	**135.35**	**135.82**	**137.69**
7_f	**28.57**	**29.04**	**30.91**	**37.94**	**38.40**	**40.28**	**84.77**	**85.24**	**87.11**	**118.49**	**118.96**	**120.83**	**135.35**	**135.82**	**137.69**

Embora comportamentos diferentes sejam observados, em geral, as decisões dos agentes fazem uso do T_S para definir o intervalo de *checkpoint*, sendo esta a melhor escolha em comparação com outras estratégias, independentemente de representar uma aplicação de uso intensivo de CPU ou memória. Para o experimento simulado, foram utilizadas as definições da instância de GPU otimizada para computação (*p2.16xlarge*), que possui o valor de $ 5,62 dólares na região *US-WEST-2*.

O tempo total de execução da aplicação reflete no custo total. Desta forma, as estratégias desta proposta oferece meios para diminuir esse tempo e ajudar a reduzir os custos monetários, na perspectiva dos usuários da computação em nuvem.

5.3 Ambiente Real

Para validar a arquitetura proposta em ambiente real foi utilizada uma aplicação que consiste em um simulador ambiental baseado em agentes para uso e cobertura da terra chamado MASE-BDI (Coelho et al., 2016). O simulador MASE-BDI é uma extensão do sistema ***M**ulti-**A**gent **S**ystem for **E**nvironmental Simulation* (MASE)[1] (Ralha et al., 2013), em que o comportamento dos agentes funciona com base no modelo de crenças, desejos e intenções (*Belief–Desire–Intention* – BDI) (Bratman, 1987).

Portanto, o MASE-BDI consiste em uma aplicação não-determinística, que faz uso intensivo de memória e CPU, fornecendo elementos para ser utilizado como um caso de validação desafiador para aplicar em plataformas de computação em nuvem, conforme apresentado em Ralha et al. (2019). Além disso, execuções em formas de BoTs podem ser aplicados no MASE-BDI, em um cenário de execuções isoladas e paralelas.

Para executar o MASE-BDI em um ambiente real, faz-se necessário uma arquitetura distribuída com uma abordagem híbrida de distribuição entre o servidor e as VMs, incluindo um servidor local (para gerenciar as execuções distribuídas) e os recursos da nuvem (VMs para executar as tarefas).

O servidor local pode ser operado com a quantidade mínima de recursos. Os recursos do servidor remoto são alocados de acordo com os requisitos do usuário e das necessidades da aplicação. O servidor utilizado consiste em uma VM *on-demand* do tipo *m3.medium*, localizada na zona *us-west-1c*, com a seguinte configuração: Ubuntu 16.04.6 LTS instalado em uma arquitetura de 64 bits composta por uma vCPU (Intel Xeon CPU E5-2670 2.50GHz) e 3,75 GB de memória RAM.

Para a criação dos agentes, a plataforma de código aberto JADE[2] foi utilizada (versão 4.5.0), que permite a criação de agentes com comportamentos dinâmico e distribuído. A tecnologia utilizada neste experimento para manutenção do banco de casos é o myCBR [3] (versão 3.1), uma ferramenta compacta que facilita a modelagem e execução de aplicações que utilizam CBR, com foco no processo de recuperação baseada em funções de similaridade (Stahl and Roth-Berghofer, 2008).

Para a execução do aplicativo MASE-BDI, foi utilizado o Docker[4], uma ferramenta para gerenciamento e execução de contêineres multi-plataforma, que permite um controle transparente e concorrente por meio de *checkpoints* bloqueados e isolados (Goldschmidt et al., 2018; Netto et al., 2017).

[1] Disponível em `http://mase.cic.unb.br/new/`
[2] *JAVA Agent DEvelopment Framework*, disponível em `https://jade.tilab.com`
[3] Disponível em `http://mycbr-project.org`
[4] Disponível em `http://docker.io`

A quantidade de registros inseridos no banco de casos gerados é de 8.633.376, inseridos usando o Algoritmo 1 considerando os tipos de instâncias *c3.large*, *m3.medium* e *c5.large*, localizados na região *US-WEST-1* em duas zonas (*us-west-1b* e *us-west-1c*). A Tabela 5.7 apresenta a quantidade de casos gerados e o total de alterações para cada um dos três tipos de instâncias.

Tabela 5.7: Distribuição dos casos nas instâncias *c3.large*, *m3.medium* e *c5.large*.

Instância	**Qtd. alterações**	**Qtd. casos gerados**
c3.large	28.919	3.400.334
m3.medium	17.382	3.271.539
c5.large	2.119	1.961.503

O MASE-BDI foi configurado para utilizar 200 agentes (100 pecuaristas e 100 agricultores) em múltiplos ciclos de simulação. Os recursos necessários para executar o MASE-BDI incluem pelo menos 600 MB de armazenamento, 2 GB de memória e 1 CPU. Mas, caso seja incluído mais recursos de CPU e memória, o tempo total de execução será diminuído.

5.3.1 Cenários de Execução

Com a finalidade de executar experimentos em diferentes cenários foram utilizadas três tipos de instâncias na região *US-WEST-1*, a saber:

- *c3.large* representa um tipo de instância otimizada para computação que possui 3,75 GB de memória RAM, 2 vCPUs (Intel(R) Xeon(R) E5-2680 2.80GHz) e 2 discos virtuais SSD com 16 GB de armazenamento;
- *m3.medium* representa um tipo de instância utilizada para uso geral que possui 3,75 GB de memória, 1 vCPU (Intel(R) Xeon(R) E5-2670 2.50GHz) e 1 disco virtual SSD com 16 GB de armazenamento;
- *c5.large* representa a nova geração de instância otimizada para computação que possui 4 GB de memória RAM, 2 vCPUs (Intel(R) Xeon(R) Platinum 3.90GHz) e 1 disco virtual otimizado para EBS (*Amazon Elastic Block Store*), que fornece um armazenamento de blocos de alta performance com *throughput* dedicado de 500 a 4000 megabits por segundo (Mbps).

Todas as configurações são compostas por uma arquitetura de 64 bits e desempenho moderado de tráfego na rede, com um tempo médio de 38 segundos para copiar os dados iniciais da aplicação, considerando cópias individuais para cada instância transiente. No cenário do experimento, as cópias paralelas das 10 execuções duram aproximadamente 3 minutos.

Para a transferência dos estados de execução (dados salvos durante o *checkpointing*), considerando uma abordagem de cópia dos dados bloqueada, i.e. a execução aguarda a transmissão completa da transferência em uma operação atômica, o tempo individual médio é de 4,7 minutos. Considerando cópias simultâneas para o mesmo servidor, o tempo médio das transferências nas 10 instâncias aumenta para 35,5 minutos.

Para a definição dos valores aplicados aos lances foi adotada a estratégia que utiliza a média dos últimos 7 dias, estratégia comumente adotada para evitar falhas de revogação na instância (Subramanya et al., 2015; Voorsluys, 2014). Nos casos em que o valor médio dos últimos 7 dias seja inferior ao preço atual da instância, adota-se a estratégia de definição do lance do usuário ser igual ao preço atual da instância.

Esse caso pode ocorrer quando há muitas variações de diminuição de preço comparada aos registros de aumento. Apesar de ser interessante a diminuição do preço para modificações futuras, i.e. incrementa o TAR, o histórico recente de diminuição de preço prejudica esta estratégia. Na Tabela 5.8 são apresentados os cenários utilizados nos experimentos do ambiente real.

Tabela 5.8: Cenários dos experimentos em um ambiente real.

#	Instância	Qtd.	DDS	HDD	T_A	T_S	TX_S	Zona	Preço base
1	***c3.large***	10	1	9	576	413	0,88	us-west-1b	0.0279
2	***c3.large***	10	3	10	576	351	0,92	us-west-1b	0.0283
3	***m3.medium***	10	2	16	844,8	389	0,88	us-west-1c	0.0077
4	***m3.medium***	10	6	22	844,8	970	0,69	us-west-1c	0.0077
5	***c5.large***	10	7	2	1384	85	0,96	us-west-2c	0.0678
		10	7	2	1384	230	0,96	us-west-2d	0.0232
6	***c5.large***	10	1	2	1384	98	0,92	us-west-2c	0.0678
		10	1	2	1384	168	0,96	us-west-2d	0.0232

Legenda:
DDS: **D**ia **D**a **S**emana; **HDD**: **H**ora **D**o **D**ia; T_A (em minutos);
TX_S calculado (em minutos); **Preço base** em dólar US por hora.

O tempo estimado para a execução do MASE-BDI com definições para 200 ciclos usando *c3.large* é de 576 minutos (~9,5 horas), enquanto que o cenário com 50 ciclos utilizando *m3.medium* tem o tempo médio de 844,8 minutos (~14 horas). Em ambos os cenários considera-se a configuração que utiliza 200 agentes durante a simulação. Os valores dos tempos de execuções em diferentes cenários são apresentados na Figura 5.12, extraídos através dos limites de execução, sendo utilizada a sua mediana.

Considerando que o MASE-BDI é uma aplicação que utiliza muito recurso de CPU, nos cenários 5 e 6 foi utilizada a instância *c5.large*, um tipo de configuração explorado por usuários que se beneficiam de processadores de alto desempenho. Portanto, nos cenários 5 e 6 (Tabela 5.8), os parâmetros de execução foram modificados para 500 agentes em 200 ciclos de execuções. Com estas mudanças, o tempo necessário para a execução é de 1384 minutos (~23 horas).

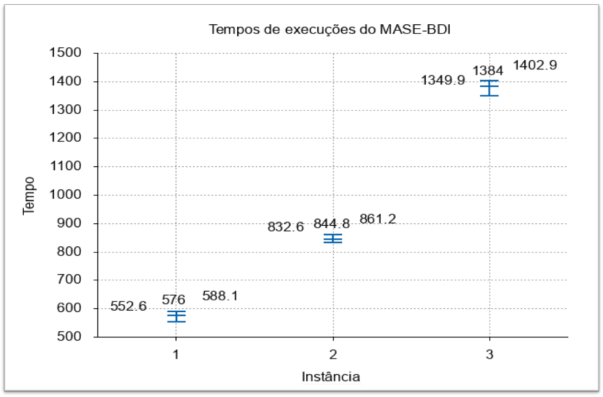


Figura 5.12: Tempos nas execuções do MASE-BDI com indicação da mediana.

Para os experimentos 1, 2, 3 e 4, o plano de execução definido foi o de *checkpoint/restore*, de acordo com o fluxo definido na Figura 4.9, nos seguintes cenários:

- Cenário 1 - instância *c3.large*, o valor calculado em $T'_{total} = 0,5356$, considerando duas execuções da instância, que possuem o mesmo valor mínimo em outras regiões. O calculo de T''_{total} considera $C_{int} = T_S$, portanto, sendo necessário apenas um *checkpoint* durante a execução ($T''_{total} = 0,2843$). Neste cenário, o plano de execução definido foi: apenas um *checkpoint* será executado no $C_{int} = 413$ (T_S), acrescentando apenas um $C_{over} = 35,5$ minutos. Caso haja uma falha, o estado da execução é recuperado a partir do único estado salvo em um novo cenário em que o plano de execução será calculado a partir do tempo restante de execução;

- Cenário 2 - semelhante ao Cenário 1, o valor calculado em $T'_{total} = 0,5433$, considerando duas execuções da instância, que possuem o mesmo valor mínimo em outras regiões. O calculo de T''_{total} considera $C_{int} = T_S$, portanto, sendo necessário apenas um *checkpoint* durante a execução ($T''_{total} = 0,2884$). Neste caso, o plano de execução conveniente também considera apenas um *checkpoint* no $C_{int} = 351$ (T_S), adicionando o tempo de um $C_{over} = 35,5$ minutos. Caso haja uma falha, o estado da execução é recuperado a partir do único estado salvo em um novo cenário em que o plano de execução será calculado a partir do tempo restante de execução;

- Cenário 3 - instância *m3.medium*, o valor de $T'_{total} = 0,2168$, considerando duas execuções da instância, que possuem o mesmo valor mínimo em outras regiões. O calculo de T''_{total} considera $C_{int} = T_S$, portanto, sendo necessário dois *checkpoint* durante a execução ($T''_{total} = 0,1175$). Neste caso, o plano de execução conveniente considera dois *checkpoints* a cada $C_{int} = 351$ (T_S), adicionando o tempo de dois $C_{over} = 35,5$ minutos. Diferente dos cenários 1 e 2, caso haja uma falha, o último

estado de execução é recuperado e um novo plano será calculado a partir do tempo restante de execução;

- Cenário 4 - semelhante ao Cenário 3, o valor de $TX_S = 0,69$ é inferior ao definido em T_{ac}, o que indica que o T_S foi alcançável 18 das 26 últimos experimentos neste mesmo cenário. Diante disso, o valor de $T'_{total} = \infty$, e o calculo de T''_{total} considera o tempo mínimo definido no modelo ($T_{min} = 60$ minutos). Para este cenário serão necessários 14 *checkpoints* durante a execução ($T''_{total} = 0,1721$). Neste caso, mesmo com uma quantidade considerável de *checkpoints*, o plano de execução conveniente considera o uso de *checkpoint*. Caso haja uma falha, o último estado de execução é recuperado e um novo plano será calculado a partir do tempo restante de execução.

Seguindo as definições do fluxo definido na Figura 4.9, para os experimentos 5 e 6, o plano de execução definido foi o de replicação ativa, de acordo nos seguintes cenários:

- Cenário 5 - instância *c5.large*. No instante da execução, considerando a parametrização do usuário que define a execução das tarefas na zona *us-west-2c*, a instância possui o preço 0,0678, porém na zona *us-west-2d*, o mesmo tipo de instância possui um preço inferior (0,0232). Com o valor de $TX_S = 0,96$ considerado alto, o cálculo de T'_{total} considera o uso das duas instâncias em zonas distintas, com o valor $T'_{total} = 2,0990$. O cálculo de T''_{total} considera C_{int} =T_S (85 minutos). Com $T_A = 1384$, são necessários 16 *checkpoints* durante a execução. Desta forma, $T''_{total} = 2,2057$. Como o valor de $T'_{total} < T''_{total}$, o plano de execução conveniente para este cenário é a replicação em duas instâncias distribuídas em zonas distintas. Para este caso, em um ambiente diferente, a instância na zona *us-west-2d* possui T_S= 230 e $TX_S = 0,96$.

- Cenário 6 - semelhante ao Cenário 5, utiliza a instância *c5.large* e a parametrização do usuário define a execução na zona *us-west-2c*, a replicação mostra-se menos custosa. No instante da execução, a instância possui o preço 0,0678 na zona definida, porém na zona *us-west-2d*, o mesmo tipo de instância possui um preço inferior (0,0232). Com o valor de $TX_S = 0,96$ considerado alto, o cálculo de T'_{total} considera o uso das duas instâncias em zonas distintas, com o valor $T'_{total} = 2,0990$. O cálculo de T''_{total} considera C_{int} =T_S (98 minutos). Com $T_A = 1384$, são necessários 14 *checkpoints* durante a execução, totalizando $T''_{total} = 2,1255$. Como o valor de $T'_{total} < T''_{total}$, o plano de execução conveniente para este cenário também é a replicação em duas instâncias distribuídas em zonas distintas. Para este caso, diferentes valores são observados na instância que está localizada na zona *us-west-2d*, que possui o T_S= 168 e $TX_S = 0,96$.

5.3.2 Análise Comparativa

É importante mencionar que os valores T'_{total} e T''_{total} são estimativas calculadas pelo modelo para o tempo e o custo total de execução de cada técnica de TF, sendo essenciais para as decisões e os comportamentos dos agentes, considerando as variáveis envolvidas na Figura 4.9. O reflexo real dos tempos totais e custos, que representam resultados de um ambiente real de execução, são apresentadas na Tabela 5.10 e Figura 5.13.

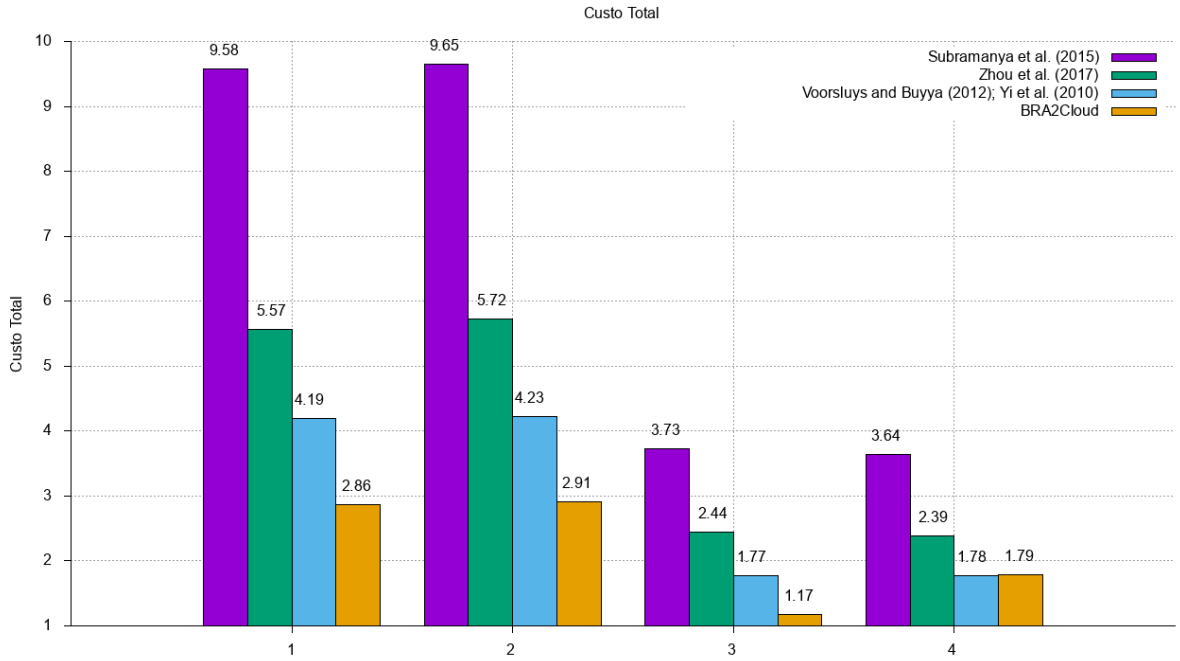


Figura 5.13: Comparação dos resultados dos cenários apresentados na Tabela 5.8.

Considerando que a estratégia proposta para o plano de replicação é semelhante ao utilizado pela literatura, i.e. o uso de replicação em duas instâncias em zonas distintas, esta proposta foi avaliada em relação ao intervalo de *checkpoints* em comparação com Subramanya et al. (2015) (15 minutos), Zhou et al. (2017) (30 minutos), Voorsluys and Buyya (2012); Yi et al. (2010) (60 minutos), semelhante ao apresentado na Seção 5.2.

Um ponto importante a ser mencionado é o fato de que atribuições diferentes para os T_S ocorrem em todos os cenários desses experimento. Desta forma, os intervalos de *checkpoints* definidos usando a abordagem proposta tornam-se dinâmicos a partir da Equação 4.6 para a definição dos parâmetros apropriados. Além disso, para garantir o tempo de execução nos experimentos comparativos e obter os resultados, a estratégia utilizada para o lance do usuário foi o preço da instância *on-demand*.

Devido a segurança e a confiabilidade no uso de instâncias *on-demand*, i.e. que possuem garantia de seus recursos e disponibilidade de acordo com a SLA, não é necessário o uso de técnicas de TF, porém o seu preço é consideravelmente superior aos praticados

pelas instâncias *spot*. Os cálculos dos valores praticados no cenários destes experimentos considerando o uso de instâncias *on-demand* podem ser observados na Tabela 5.9.

Na comparação com instâncias *on-demand*, que possui o preço fixo de USD 0.0850 na região *US-WEST-1*, no pior caso (Cenário 4), o ganho ao utilizar instâncias *spot* usando esta abordagem é de 83,48%, considerando o mesmo tempo de aplicação (Tabela 5.10).

Por padrão, uma conta não comercial da AWS possui um limite de 20 instâncias *spot* simultâneas por região. Portanto, para a criação dos cenários necessários neste experimento, foram utilizadas contas diferentes para execuções simultâneas.

Tabela 5.9: Custos calculados no uso de instâncias *on-demand*.

Cenário	**Instância**	**Região**	**Zona**	**Preço**	**Preço Total**	**%**
1	*c3.large*	*US-WEST-1*	*us-west-1b*	0,12	11,52	75,17 %
2	*c3.large*	*US-WEST-1*	*us-west-1b*	0,12	11,52	74,73 %
3	*m3.medium*	*US-WEST-1*	*us-west-1c*	0,077	10,84	89,20 %
4	*m3.medium*	*US-WEST-1*	*us-west-1c*	0,077	10,84	83,48 %

Legenda:
% refere-se a diferença comparada a [*****];
Preço refere-se ao valor atual de uma instância *on-demand*.
Preços em dólar US por hora.

Durante os experimentos, 200 instâncias foram solicitadas: 160 para o experimento usando *checkpoint/restore* (40 por cada cenário); e 40 usando replicação (20 por cada cenário).

Ao final da execução dos experimentos em todos os cenários, aproximadamente 28.000 simulações foram realizadas no MASE-BDI, divididas entre 16.000 e 4.000 entre as instâncias *c3.large* e *m3.medium*. Aproximadamente 8.000 simulações foram executadas na instância *c5.large*. Este valor aproximado ocorre devido ao cancelamento de execução da segunda replicação quando uma delas finaliza a sua execução. Como previsto, não ocorreram falhas de revogação em nenhuma execução durante os cenários deste experimento.

Tabela 5.10: Resultados dos experimentos em ambiente real.

Referência	Intervalo	Qtd.	T_A	Tempo Total	Custos Instância	Custos Total	%
			Cenário 1				
(Subramanya et al., 2015)	15	38	576	2062	0,9581	9,58	70,12 %
(Zhou et al., 2017)	30	19	576	1199	0,5575	5,57	48,62 %
(Voorsluys and Buyya, 2012; Yi et al., 2010)	60	9	576	902	0,4194	4,19	31,70 %
[*]	413	1	576	616	0,286	2,86	-
			Cenário 2				
(Subramanya et al., 2015)	15	38	576	2047	0,965	9,65	69,76 %
(Zhou et al., 2017)	30	19	576	1213	0,5721	5,72	48,96 %
(Voorsluys and Buyya, 2012; Yi et al., 2010)	60	9	576	897	0,4230	4,23	30,99 %
[*]	351	1	576	619	0,2912	2,91	-
			Cenário 3				
(Subramanya et al., 2015)	15	56	844,8	2912	0,373	3,73	68,44 %
(Zhou et al., 2017)	30	28	844,8	1909	0,2449	2,44	51,85 %
(Voorsluys and Buyya, 2012; Yi et al., 2010)	60	14	844,8	1386	0,1778	1,77	33,69 %
[*]	389	2	844,8	919	0,117	1,17	-
			Cenário 4				
(Subramanya et al., 2015)	15	56	844,8	2843	0,3648	3,64	50,68 %
(Zhou et al., 2017)	30	28	844,8	1869	0,2398	2,39	24,98 %
(Voorsluys and Buyya, 2012; Yi et al., 2010)	60	14	844,8	1391	0,1785	1,78	-0,79 %
[*]	60	14	844,8	1402	0,1791	1,79	-

Legenda:
[*] representa a abordagem utilizada nesta proposta;
% refere-se a diferença comparada a **[*]** considerando o mesmo cenário;
Custos em dólar US por hora.

Capítulo 6

Conclusões

O cenário de execução com uso de servidores transientes na computação em nuvem é dinâmico e não confiável. Desta forma, a escolha da técnica de TF apropriada, com seus respectivos parâmetros, pode viabilizar um ambiente resiliente com uso de recursos de maneira eficiente para reduzir custos segundo a perspectiva do usuário.

Este trabalho se concentra em um elemento importante para o provimento de um ambiente resiliente: comportamentos tolerantes a falhas. Conforme abordado em Sharma et al. (2017), os autores enfatizam os desafios associados a aplicação de abordagens de TF em servidores transientes, focando a necessidade de um plano de TF apropriado para explorar completamente os seus recursos.

Neste trabalho foi apresentada uma arquitetura baseada em agentes autônomos para escolha dinâmica da técnica mais adequada de TF na execução de BoT, provendo um ambiente resiliente de execução em nuvem computacional com uso de instâncias transientes. Os agentes decidem através de um modelo que utiliza o raciocínio baseado em casos, apoiado pela análise de sobrevivência, qual estratégia de TF utilizar de acordo com o cenário de execução, visando a redução de custos para o usuário. Desta forma, o objetivo geral proposto na Seção 1.3 foi totalmente alcançado. Os resultados experimentais foram comparados às abordagens presentes na literatura conforme apresentado no Capítulo 5.

Considerando os objetivos secundários apresentados na Seção 1.3, os mesmos foram alcançados da seguinte forma:

- (i) as técnicas de TF (*retry*, *checkpoint/restore* e *replication*) foram avaliadas considerando diferentes cenários, incluindo ambiente simulado e real conforme apresentado no Capítulo 5, Seções 5.2 e 5.3 com resultados nas Tabelas 5.5, 5.4, 5.6 e 5.10.

- (ii) e (iii), como detalhado na Seção 5.3, o simulador MASE-BDI foi utilizado como estudo de caso nos experimentos em ambiente real, sem a necessidade de adaptação

das tarefas (i.e., utilizando um BoT existente), provendo desta forma um ambiente transparente para a aplicação das técnicas de TF. Foram realizadas comparações com as propostas da literatura, como apresentada na Tabela 5.10.

Os resultados experimentais demonstram que o modelo de predição alcança altos níveis de acurácia (Figura 5.6), gerando tempos de sobrevivências estimados com baixas probabilidades de revogação, obtidos através da Equação 4.3, para a geração de curvas de sobrevivência (Figura 2.13) e MTS (Tabela 5.2). Na Seção 5.1 foi feita uma validação desta abordagem de predição e os resultados chegaram a 91% de sucesso (Figura 5.7) utilizando uma estratégia de recência para considerar mudanças de preços recentes (Figura 5.5).

Comparado as abordagens existentes na literatura, os resultados dos experimentos em ambiente simulado e real demonstram que a arquitetura desenvolvida alcança melhores resultados, com um ganho de até 70,12% na redução dos custos obtido através de experimentos em um ambiente real (Tabela 5.10), com uma diferença de 4,04% quando comparado ao ambiente simulado (Tabela 5.6).

Finalmente, pode-se destacar como contribuições deste trabalho:

- Uma arquitetura baseada em agentes autônomos com comportamentos dinâmicos baseados no cenário de execução;
- Uma abordagem para predição do tempo de sobrevivência de instâncias transientes, que utiliza CBR, suportado pela análise de sobrevivência;
- Um modelo multi-estratégico de TF, que define as técnicas e seus respectivos parâmetros de maneira não estática;
- Um *framework* de execução, chamado de BRA2Cloud, que permite a execução de BoT sem a necessidade de adaptação das tarefas;
- Um ambiente resiliente para instância transiente que permite a redução dos custos em uma perspectiva do usuário de computação em nuvem.

As técnicas de *retry*, *checkpoint/restore* e *replicação* foram utilizadas por permitirem a diminuição dos custos de execução. Até onde tem-se conhecimento, não existe trabalho publicado que combine o uso dos elementos desta tese para o provimento de um ambiente resiliente que utiliza exclusivamente servidores transientes na computação em nuvem.

6.1 Publicações

Durante o desenvolvimento da pesquisa foram feitas várias publicações científicas, as quais serão apresentadas em ordem cronológica crescente conforme itens na sequência.

Em 2018 foram publicados cinco artigos em conferências nacionais e internacionais. Primeiramente com foco no modelo de predição de falhas de revogação foram publicados dois artigos na área de sistemas distribuídos e *cloud*: (1) *Workshop* em *Clouds* e Aplicações (WCGA), evento coligado ao Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC); e no (2) *11th International Conference in Cloud Computing (CLOUD 2018)*.

Com foco no modelo de agentes foram publicados dois artigos na área de Inteligência Artificial e computação de alto desempenho: (3) *7th Brazilian Conference on Intelligent Systems (BRACIS)*; e (4) *30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*.

Um artigo mais completo da abordagem resiliente para uso de instâncias transientes foi publicado no *IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2018)* (5).

1. José Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: Uma Abordagem Baseada em Aprendizagem de Máquina para Predição de Falhas de Revogação em Instâncias Transientes, XVI *Workshop* em *Clouds* e Aplicações (WCGA), XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC), 2018, Campos do Jordão, SP, Brasil, p. 1-12. (Estrato B4 Qualis Capes na Ciência da Computação);

2. Jose Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: *A prediction approach to define checkpoint intervals in spot instances.* Em Cloud Computing - CLOUD 2018 - *11th International Conference*, *Held as Part of the Services Conference Federation*, SCF 2018, Seattle, WA, USA, 2018, Proceedings, páginas 84–93;

3. Jose Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: *An agent-based fog computing architecture for resilience on amazon EC2 spot instances.* Em *7th Brazilian Conference on Intelligent Systems*, BRACIS 2018, São Paulo, Brasil, 2018, páginas 360–365. (Estrato A4 Qualis Capes na Ciência da Computação);

4. Jose Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: *A fault-tolerant agent-based architecture for transient servers in fog computing.* Em *30th International Symposium on Computer Architecture and High Performance Computing*, SBAC-PAD 2018, Lyon, France, 2018, páginas 282–289. (Estrato A4 Qualis Capes na Ciência da Computação);

5. Jose Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: *A resilient agent-based architecture for efficient usage of transient servers in cloud computing.* Em 2018 *IEEE International Conference on Cloud Computing Technology and Science*, CloudCom 2018, Nicosia, Cyprus, 2018, páginas 218–225. (Estrato A2 Qualis Capes na Ciência da Computação).

Em 2019 foram publicados dois artigos em periódicos internacionais com foco na arquitetura multi-provedor tolerante a falhas (1); e outro detalhando a arquitetura baseada em agentes (2).

1. Jose Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: *Mults: A multi-cloud fault-tolerant architecture to manage transient servers in cloud computing. Journal of Systems Architecture*, 101:101651, 2019. (Estrato B1 Qualis Capes na Ciência da Computação);

2. Jose Pergentino de Araújo Neto, Donald M. Pianto e Célia Ghedini Ralha: *Towards increasing reliability of amazon EC2 spot instances with a fault-tolerant multi-agent architecture. Multiagent and Grid Systems*, 15(3):259–287, 2019. (Estrato B2 Qualis Capes na Ciência da Computação).

Acreditamos que os resultados desta pesquisa está sendo reconhecida pela comunidade acadêmica, tendo em vista que o autor desta tese foi convidado como *Renowned Speaker* na *International Conference on Cloud Computing and Virtualization*[1] (ICCV 2020), a ser realizada em Londres, Inglaterra, para apresentar sua proposta arquitetural resiliente para uso de instâncias transientes na computação em nuvem.

6.2 Trabalhos Futuros

Como trabalhos futuros pode-se validar a arquitetura proposta com dados reais de outros provedores de nuvem. Atualmente, apenas o provedor Amazon oferece dados públicos que podem ser utilizados para prever falhas de acordo com cada estratégia de lance. Embora o provedor Google tenha um repositório de dados[2] que descreve vários registros no gerenciamento de disponibilização em seus *clusters*, esse repositório não é atualizado frequentemente. No entanto, para utilizar outros provedores seria necessário gerar uma base de casos através dos próprios recursos do provedor, mas o custo para gerar dados suficientemente úteis seria alto.

[1]Disponível em `https://cloudcomputing.annualcongress.com`
[2]Disponível em `https://github.com/google/cluster-data`

Para aumentar a resiliência da abordagem proposta e considerando diferentes tipos de aplicações, uma combinação de planos de TF pode ser usada, por exemplo, um plano de execução que utiliza a replicação com maiores intervalos de *checkpointing* ou migração de tarefas (*job-migration*) quando ocorre uma mudança considerável de preço.

Durante as validações e os experimentos realizados, foram identificados casos em que nossa abordagem produziu resultados insuficientes nas validações do T_S, apesar de que estes baixos índices induzem a um plano conservador de execução. Para estas instâncias, o processo de geração de dados recentes aparentemente possui uma forte influência, com variações de preço superior ao observado em outras instâncias. Para capturar essas mudanças, implementamos um fator de recência que considera dados de observações mais recentes para aumentar as taxas de sucesso.

A janela de tempo usada na abordagem de recência foi definida empiricamente através de experimentos utilizando uma instância que possui registros consideráveis de alteração de preço. Foi utilizado uma janela de validação que incluía as últimas quatro semanas, porém algumas instâncias requerem uma variação no período desta janela. Entendemos que será necessário estudar como definir dinamicamente esta janela considerando outros tipos de instâncias. Além disso, uma análise de sazonalidade pode ser explorada, utilizando eventos periódicos, e.g. aumento de demanda durante as festas de final de ano.

Uma abordagem de múltiplos provedores pode ser usada, estendendo o número de modelos de preços de acordo com cada provedor. É importante destacar que já iniciamos a pesquisa desta abordagem *multi-cloud*, chamada de MULTS[3], mas que só utiliza uma técnica de TF (*checkpoint/restore*).

Por fim, a arquitetura proposta pode ser modificada, retirando os agentes e usando uma abordagem distribuída utilizando recursos nativos de comunicação por mensagens. Desta forma, questões relacionadas a performance e uso de recursos podem ser comparados com a abordagem proposta.

[3] *A **Mult**i-cloud Fault-Tolerant Architecture for **T**ransient **S**ervers*, disponível em `http://mults.cic.unb.br`

Referências

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI communications*, 7(1):39–59. 32

Allison, P. D. (2010). *Survival analysis using SAS: a practical guide*. Sas Institute. 41

Amazon (2019). Amazon web services (aws). 1

Anarado, I. and Andreopoulos, Y. (2016). Core Failure Mitigation in Integer Sum-of-Product Computations on Cloud Computing Systems. *IEEE Transactions on Multimedia*, 18(4):789–801. 66

Araujo Neto, J. P., Pianto, D. M., and Ghedini Ralha, C. (2018). A resilient agent-based architecture for efficient usage of transient servers in cloud computing. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 218–225. 89

Araujo Neto, J. P., Pianto, D. M., and Ralha, C. G. (2018). A Prediction Approach to Define Checkpoint Intervals in Spot Instances. In *Proceedings of the 11th International Conference on Cloud Computing, CLOUD 2018, SCF 2018*, volume 10967, pages 84–93, Seattle, WA, USA. Springer. 79, 89

Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., and Rabkin, A. (2010a). A view of cloud computing. *Communications of the ACM*, 53(4):50. 1, 8

Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., and Rabkin, A. (2010b). Above the Clouds: A Berkeley View of Cloud Computing. *Communications of the ACM*, 53(4):50. 12

Avizienis, A. (1978). Fault-tolerance: The survival attribute of digital systems. *Proceedings of the IEEE*, 66(10):1109–1125. 16

Barborak, M., Dahbura, A., and Malek, M. (1993). The consensus problem in fault-tolerant computing. *ACM Computing Surveys (CSur)*, 25(2):171–220. 17

Barr, A. and Feigenbaum, E. A. (1981). The handbook of artificial intelligence, vol. 2. *Wm. Kaufman, Inc., Los Altos CA. Contributions from over*, 100. 28

Bellifemine, F., Poggi, A., and Rimassa, G. (1999). Jade–a fipa-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London. 39

Ben-Yehuda, O. A., Ben-Yehuda, M., Schuster, A., and Tsafrir, D. (2011). Deconstructing Amazon EC2 Spot Instance Pricing. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, volume 1, pages 304–311. IEEE. 14

Bian, J., Weng, C., Du, J., and Li, M. (2008). A QoS-Aware and Fault-Tolerant Workflow Composition for Grid. In *2008 Seventh International Conference on Grid and Cooperative Computing*, pages 510–516. IEEE. 27

Binnig, C., Salama, A., Zamanian, E., El-Hindi, M., Feil, S., and Ziegler, T. (2015). Spotgres - parallel data analytics on Spot Instances. In *2015 31st IEEE International Conference on Data Engineering Workshops*, volume 2015-June, pages 14–21. IEEE. 48, 49, 50, 57, 66

Bratman, M. (1987). *Intention, plans, and practical reason*. Harvard University Press, Cambridge, MA. 103

Brenner, W., Zarnekow, R., and Wittig, H. (2012). *Intelligent software agents: foundations and applications*. Springer Science & Business Media. 34

Buyya, R., Javadi, B., and Thulasiramy, R. K. (2011). Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 219–228. IEEE. 48, 51, 52, 66, 79

Buyya, R., Qu, C., Calheiros, and N., R. (2016). A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications*, 65:167–180. 3

Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616. 1, 8, 10, 62

Chandy, K. M. and Lamport, L. (1985). Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems (TOCS)*, 3(1):63–75. 23

Chapelle, O., Scholkopf, B., and Zien, Eds., A. (2009). Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3):542–542. 31

Cirne, W., Brasileiro, F., Paranhos, D., Góes, L. F. W., and Voorsluys, W. (2007). On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing*, 33(3):213–234. 28

Cirne, W., Brasileiro, F., Sauvé, J., Andrade, N., Paranhos, D., Santos-neto, E., Medeiros, R., and Gr, F. C. (2003). Grid computing for bag of tasks applications. In *In Proc. of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment*. 2

Cirne, W., Di, S., and Kondo, D. (2012). Characterization and Comparison of Cloud versus Grid Workloads. In *2012 IEEE International Conference on Cluster Computing*, pages 230–238. IEEE. 60

Coelho, C. G. C., Abreu, C. G., Ramos, R. M., Mendes, A. H. D., Teodoro, G., and Ralha, C. G. (2016). MASE-BDI: agent-based simulator for environmental land change with efficient and parallel auto-tuning. *Appl. Intell.*, 45(3):904–922. 103

Colosimo, E. and Giolo, S. (2014). *Análise de sobrevivência aplicada*, volume 2. Blucher. 41

Columbus, L. (2016). Roundup of cloud computing forecasts and market estimates, 2016. *Forbes Magazine*. 13

Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition. 22

Cox, D. R. (1984). *Analysis of survival data*. Routledge, 1st edition edition. 42

Davis, R., Buchanan, B., and Shortliffe, E. (1977). Production rules as a representation for a knowledge-based consultation program. *Artificial Intelligence*, 8(1):15 – 45. 86

De la Prieta, F., Rodríguez, S., Bajo, J., and Corchado, J. M. (2014). +cloud: A virtual organization of multiagent system for resource allocation into a cloud computing environment. In Nguyen, N. T., Kowalczyk, R., Corchado, J. M., and Bajo, J., editors, *Transactions on Computational Collective Intelligence XV*, pages 164–181. Springer Berlin Heidelberg, Berlin, Heidelberg. 59

Demuth, H. B., Beale, M. H., De Jess, O., and Hagan, M. T. (2014). *Neural Network Design*. Martin Hagan, USA, 2nd edition. 32

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In Prieditis, A. and Russell, S., editors, *Machine Learning Proceedings 1995*, pages 194 – 202. Morgan Kaufmann, San Francisco (CA). 31

Efron, B. (1988). Logistic regression, survival analysis, and the kaplan-meier curve. *Journal of the American statistical Association*, 83(402):414–425. 41

Elnozahy, E. N. M., Alvisi, L., Wang, Y.-M., and Johnson, D. B. (2002). A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, 34(3):375–408. 21, 22, 23, 24, 27

Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2):114–131. 39

Fedoruk, A. and Deters, R. (2002). Improving fault-tolerance by replicating agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems, part 2 - AAMAS '02*, page 737, New York, NY, USA. ACM Press. 26

Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading. 37

Foster, I., Zhao, Y., Raicu, I., and Lu, S. (2008). Cloud Computing and Grid Computing 360-Degree Compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. IEEE. 1, 7

Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., and Stoica, I. (2009). Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13):2009. 8, 10, 12

Gärtner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26. 28

Georgeff, M., Pell, B., Pollack, M., Tambe, M., and Wooldridge, M. (1999). The Belief-Desire-Intention Model of Agency. In *Intelligent Agents V: Agents Theories, Architectures, and Languages. 5th International Workshop, ATAL'98.*, pages 1–10. Springer. 38

Gokhale, S., Lyu, M., and Trivedi, K. (1998). Reliability simulation of fault-tolerant software and systems. *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*, pages 1–21. 27, 40

Goldschmidt, T., Hauck-Stattelmann, S., Malakuti, S., and Grüner, S. (2018). Container-based architecture for flexible industrial control applications. *Journal of Systems Architecture*, 84:28 – 36. 103

Gong, Y., He, B., and Zhou, A. C. (2015). Monetary cost optimizations for MPI-based HPC applications on Amazon clouds. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '15*, pages 1–12, New York, New York, USA. ACM Press. 3, 15, 60

Google (2019). Google cloud platform (gcp). 1

Guerraoui, R. and Schiper, A. (1996). Fault-tolerance by replication in distributed systems. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 1088, pages 38–57. Springer. 27, 28

Guimaraes, F. P., Célestin, P., Batista, D. M., Rodrigues, G. N., and de Melo, A. C. M. A. (2014). A Framework for Adaptive Fault-Tolerant Execution of Workflows in the Grid: Empirical and Theoretical Analysis. *Journal of Grid Computing*, 12(1):127–151. 27

Guo, W., Chen, K., Wu, Y., and Zheng, W. (2015). Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15*, pages 191–202, New York, New York, USA. ACM Press. 3

Hägg, S. (1997). A sentinel approach to fault handling in multi-agent systems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1:181–195. 59

He, X., Shenoy, P., Sitaraman, R., and Irwin, D. (2015). Cutting the Cost of Hosting Online Services Using Cloud Spot Markets. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15*, pages 207–218, New York, New York, USA. ACM Press. 3, 48, 49, 50, 53, 60

Hwang, S. and Kesselman, C. (2003). A Flexible Framework for Fault Tolerance in the Grid. *Journal of Grid Computing*, 1(3):251–272. 27

Iosup, A., Ostermann, S., Yigitbasi, M. N., Prodan, R., Fahringer, T., and Epema, D. H. J. (2011). Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):931–945. 2

Iosup, A., Sonmez, O., Anoep, S., and Epema, D. (2008). The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th international symposium on High performance distributed computing - HPDC '08*, page 97, New York, New York, USA. ACM Press. 2

Irwin, D., Sharma, P., and Shenoy, P. (2017). Portfolio-driven Resource Management for Transient Cloud Servers. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1):5:1—-5:23. 3, 4, 62, 69

Isermann, R. (2006). *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer, Berlin. 16

Jalote, P. (1994). *Fault tolerance in distributed systems*. Prentice-Hall, Inc. 22

Jangjaimon, I. and Tzeng, N.-F. (2015). Effective Cost Reduction for Elastic Clouds under Spot Instance Pricing Through Adaptive Checkpointing. *IEEE Transactions on Computers*, 64(2):396–409. 48, 49, 50, 51, 60, 66

Javadi, B., Thulasiram, R. K., and Buyya, R. (2013). Characterizing spot price dynamics in public cloud environments. *Future Gener. Comput. Syst.*, 29(4):988–999. 79

Kaminka, G. A. (2000). *Execution Monitoring in Multi-Agent Environments*. PhD thesis, Computer Science Department—University of Southern California. 36

Kendrick, P., Criado, N., Hussain, A., and Randles, M. (2018). A self-organising multi-agent system for decentralised forensic investigations. *Expert Systems with Applications*, 102:12 – 26. 34

Khan, A., Xifeng Yan, Shu Tao, and Anerousis, N. (2012). Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*, pages 1287–1294. IEEE. 1

Kishore, J., Goel, M., and Khanna, P. (2010). Understanding survival analysis: Kaplan-Meier estimate. *International Journal of Ayurveda Research*, 1(4):274. 41, 42

Kitchenham, B. (2004). Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26. 43

Kolodner, J. (2014). *Case-based reasoning*. Morgan Kaufmann. 32, 33

Korb, K. B. and Nicholson, A. E. (2010). *Bayesian artificial intelligence*. CRC press. 31

Kotsiantis, S. B., Zaharakis, I., and Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24. 30

Kulikowski, C. A. and Weiss, S. M. (1991). Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems. *Machine Learning. San Francisco: Morgan Kaufmann Publishers. ISBN*, pages 1–55860. 30

Laprie, J.-C. (1985). Dependable computing and fault-tolerance. *Digest of Papers FTCS-15*, pages 2–11. 16

Lee, K. and Son, M. (2017). DeepSpotCloud: Leveraging Cross-Region GPU Spot Instances for Deep Learning. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 98–105. IEEE. 49, 50, 57

Lee, P. A. and Anderson, T. (1990). *Fault Tolerance: Principles and Practice*, volume 3 of *Dependable Computing and Fault-Tolerant Systems*. Springer, Vienna. 16, 17, 18, 20, 25, 27

Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. (2009). What's inside the Cloud? An architectural map of the Cloud landscape. In *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, volume , May 23„ pages 23–31. IEEE. 9

Li, Z., Kihl, M., and Robertsson, A. (2015). On a feedback control-based mechanism of bidding for cloud spot service. In *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, pages 290–297. IEEE. 48, 50, 52

Lopes, A. L. M., Fracolli, L. A., et al. (2008). Revisão sistemática de literatura e metassíntese qualitativa: considerações sobre sua aplicação na pesquisa em enfermagem. *Texto e Contexto-Enfermagem*, 17(4):771–778. 47

Lu, S., Li, X., Wang, L., Kasim, H., Palit, H., Hung, T., Legara, E. F. T., and Lee, G. (2013). A Dynamic Hybrid Resource Provisioning Approach for Running Large-Scale Computational Applications on Cloud Spot and On-Demand Instances. In *2013 International Conference on Parallel and Distributed Systems*, pages 657–662. IEEE. 49, 50, 54

Luger, G. and Stubblefield, W. (1993). Artificial intelligence—structures and strategies for complex problem solving. *Cummings, Redwood City, CA*. 28, 31

Marzullo, K. (1993). Consistent global states of distributed systems: Fundamental concepts and mechanisms. *Distributed Systems, Ed. S. Mullender, Addison-Wesley*, pages 55–96. 23

Meeker, W. Q. and Escobar, L. A. (1998). *Statistical Methods for Reliability Data*. Wiley, New York. 85

Meeker, W. Q. and Escobar, L. A. (2014). *Statistical methods for reliability data*. John Wiley & Sons. 41

Mell, P., Grance, T., et al. (2011). The NIST definition of cloud computing. *Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg*. 1, 7, 10, 12

Michel, F., Ferber, J., Drogoul, A., et al. (2009). Multi-agent systems and simulation: a survey from the agents community's perspective. *Multi-Agent Systems: Simulation and Applications, Computational Analysis, Synthesis, and Design of Dynamic Systems*, pages 3–52. 39

Microsoft (2019). Microsoft azure. 1

Miller Jr, R. G. (2011). *Survival analysis*, volume 66. John Wiley & Sons. 41

Moody, A., Bronevetsky, G., Mohror, K., and de Supinski, B. R. (2010). Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE. 51

Moore, R. C. (1984). A formal theory of knowledge and action. Technical report, DTIC Document. 36

Moreno-Vozmediano, R., Montero, R. S., and Llorente, I. M. (2013). Key Challenges in Cloud Computing: Enabling the Future Internet of Services. *IEEE Internet Computing*, 17(4):18–25. 15

Müller, J.-P. and Ferber, J. (1996). Influences and reaction: a model of situated multiagent systems. In *Proceedings of Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 72–79. 37

Nelson, W. (1990). Accelerated life testing: statistical models, data analysis and test plans. *Accelerated life testing: statistical models data analysis and test plants*. 41

Netto, H. V., Lung, L. C., Correia, M., Luiz, A. F., and de Souza, L. M. S. (2017). State machine replication in containers managed by kubernetes. *Journal of Systems Architecture*, 73:53 – 59. Special Issue on Reliable Software Technologies for Dependable Distributed Systems. 103

Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The Eucalyptus Open-Source Cloud-Computing System. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131. IEEE. 10

Nwana, H. S. (1996). Software agents: an overview. *The Knowledge Engineering Review*, 11(3):205–244. 35

Oprescu, A.-M. and Kielmann, T. (2010). Bag-of-Tasks Scheduling under Budget Constraints. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 351–359. IEEE. 2

Pedreira, O., Piattini, M., Luaces, M. R., and Brisaboa, N. R. (2007). A systematic review of software process tailoring. *ACM SIGSOFT Software Engineering Notes*, 32(3):1. 44, 45

Pendharkar, P. C. and Cusatis, P. (2018). Trading financial indices with reinforcement learning agents. *Expert Systems with Applications*, 103:1–13. 34

Platon, E., Mamei, M., Sabouret, N., Honiden, S., and Parunak, H. V. D. (2007). Mechanisms for environments in multi-agent systems: Survey and opportunities. *Autonomous Agents and Multi-Agent Systems*, 14(1):31–47. 39

Platon, E., Sabouret, N., and Honiden, S. (2008). An architecture for exception management in multiagent systems. *International Journal of Agent-Oriented Software Engineering*, 2(3):267. 59

Poola, D., Ramamohanarao, K., and Buyya, R. (2016). Enhancing Reliability of Workflow Execution Using Task Replication and Spot Instances. *ACM Transactions on Autonomous and Adaptive Systems*, 10(4):1–21. 49, 50, 56

Qu, C., Calheiros, R. N., and Buyya, R. (2016). A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications*, 65:167–180. 3

Ralha, C. G., Abreu, C. G., Coelho, C. G. C., Zaghetto, A., Macchiavello, B., and Machado, R. B. (2013). A multi-agent model system for land-use change simulation. *Environmental Modelling and Software*, 42:30–46. 103

Ralha, C. G., Mendes, A. H. D., Laranjeira, L. A., Araújo, A. P. F., and Melo, A. C. M. A. (2019). Multiagent system for dynamic resource provisioning in cloud computing platforms. *Future Generation Comp. Syst.*, 94:80–96. 103

Randell, B. (1975). System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232. 24

Rao, A. S., Georgeff, M. P., et al. (1995). Bdi agents: From theory to practice. In *ICMAS*, volume 95, pages 312–319. 34, 38

Rezende, S. O. (2003). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda. 31

Richardson, L. and Ruby, S. (2008). *RESTful web services*. "O'Reilly Media, Inc.". 77

Rowley, H., Baluja, S., and Kanade, T. (1998). Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38. 32

Russell, S. and Norvig, P. (2010). *Artificial intelligence: a modern approach*. Pearson, Upper Saddle River, NJ, 3 edition. 28, 30, 31, 34, 35, 36, 37, 69, 70

Sharma, P., Lee, S., Guo, T., Irwin, D., and Shenoy, P. (2017). Managing Risk in a Derivative IaaS Cloud. *IEEE Transactions on Parallel and Distributed Systems*, 9219(c):1–1. 49, 50, 58, 60, 111

Shastri, S., Rizk, A., and Irwin, D. (2016). Transient guarantees: Maximizing the value of idle cloud capacity. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 992–1002. 2, 13, 63

Shieber, S. M. (1994). Lessons from a Restricted Turing Test. *Communications of the ACM*, 37(6):70–78. 29

Siddiqui, U., Tahir, G. A., Rehman, A. U., Ali, Z., Rasool, R. U., and Bloodsworth, P. (2012). Elastic jade: Dynamically scalable multi agents using cloud resources. In *2012 Second International Conference on Cloud and Green Computing*, pages 167–172. 59

Simon, H. (1983). Search and Reasoning in problem solving. *Artificial Intelligence*, 21(1-2):7–29. 29

Sotomayor, B., Montero, R. S., Llorente, I. M., and Foster, I. (2009). Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13(5):14–22. 10, 12

Souchon, F., Dony, C., Urtado, C., and Vauttier, S. (2003). Improving exception handling in multi-agent systems. In *International Workshop on Software Engineering for Large-Scale Multi-agent Systems*, pages 167–188. Springer. 59

Stahl, A. and Roth-Berghofer, T. R. (2008). Rapid prototyping of cbr applications with the open source tool mycbr. In Althoff, K.-D., Bergmann, R., Minor, M., and Hanft, A., editors, *Advances in Case-Based Reasoning*, pages 615–629, Berlin, Heidelberg. Springer Berlin Heidelberg. 103

Stanković, R., Štula, M., and Maras, J. (2017). Evaluating fault tolerance approaches in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 31(1):151–177. 17

Subramanya, S., Guo, T., Sharma, P., Irwin, D., and Shenoy, P. (2015). SpotOn: A Batch Computing Service for the Spot Market. In *Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15*, pages 329–341, New York, New York, USA. ACM Press. 48, 49, 50, 56, 60, 88, 89, 97, 98, 105, 108, 110

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press. 30, 31

Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 2. MIT press Cambridge. 31

Taesoon Park, Ilsoo Byun, Hyunjoo Kim, and Yeom, H. (2002). The performance of checkpointing and replication schemes for fault tolerant mobile agent systems. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 256–261. IEEE Comput. Soc. 21

Tanenbaum, A. S. and Van Steen, M. (2007a). *Distributed systems*. Prentice-Hall. 15, 16, 22

Tanenbaum, A. S. and Van Steen, M. (2007b). *Distributed systems: principles and paradigms*. Prentice-Hall. 17

Tang, X., Yang, X., Liao, G., and Zhu, X. (2016). A shared cache-aware Task scheduling strategy for multi-core systems. *Journal of Intelligent & Fuzzy Systems*, 31(2):1079–1088. 2

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, 59:433–460. 29

Turing, A. M. (2009). Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer. 29

Vaquero, L. M., Rodero-Merino, L., Caceres, J., and Lindner, M. (2008). A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55. 10

Velleman, J. D. and Bratman, M. E. (1991). Intention, Plans, and Practical Reason. *The Philosophical Review*, 100(2):277. 38

Von Neumann, J., Burks, A. W., et al. (1966). Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14. 32

Voorsluys, W. (2014). *Resource provisioning in spot market-based cloud computing environments*. PhD thesis, University of Melbourne. 49, 50, 55, 60, 66, 88, 105

Voorsluys, W. and Buyya, R. (2012). Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 542–549. IEEE. 12, 49, 53, 55, 66, 87, 89, 97, 98, 108, 110

Weyns, D., Parunak, H. V. D., Michel, F., Holvoet, T., and Ferber, J. (2004). Environments for multiagent systems state-of-the-art and research challenges. In *International Workshop on Environments for Multi-Agent Systems*, pages 1–47. Springer Berlin Heidelberg. 36

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons. 34

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice. *Knowledge engineering review*, 10(2). 34, 35, 38, 39

Xu, P. and Deters, R. (2004). MAS and fault-management. In *Applications and the Internet, 2004. Proceedings. 2004 International Symposium on*, pages 283–286. IEEE. 59

Yang, Y., Peng, X., and Wan, X. (2015). Security-aware data replica selection strategy for Bag-of-Tasks application in cloud computing. *Journal of High Speed Networks*, 21(4):299–311. 2

Yi, S., Andrzejak, A., and Kondo, D. (2012). Monetary Cost-Aware Checkpointing and Migration on Amazon Cloud Spot Instances. *IEEE Transactions on Services Computing*, 5(4):512–524. 49, 50, 52, 66, 97, 98

Yi, S., Kondo, D., and Andrzejak, A. (2010). Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 236–243. IEEE. 87, 97, 98, 108, 110

Youseff, L., Butrico, M., and Da Silva, D. (2008a). Toward a Unified Ontology of Cloud Computing. In *2008 Grid Computing Environments Workshop*, pages 1–10. IEEE. 9

Youseff, L., Butrico, M., and Da Silva, D. (2008b). Toward a Unified Ontology of Cloud Computing. In *2008 Grid Computing Environments Workshop*, pages 1–10. IEEE. 9

Zhou, J., Zhang, Y., and Wong, W.-F. (2017). Fault Tolerant Stencil Computation on Cloud-based GPU Spot Instances. *IEEE Transactions on Cloud Computing*, 7161(c):1–1. 49, 50, 58, 60, 97, 98, 108, 110