UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Alocação de Máquinas Virtuais em Ambientes de Computação em Nuvem Baseada em Requisitos de Service Level Agreement

Versão Final Pós Defesa

Raysa da Luz Oliveira

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

(2º Ciclo de Estudos)

Orientador: Professor Doutor Mário Marques Freire

Covilhã, Julho de 2017

Dissertação elaborada no âmbito do Grupo *Multimedia Signal Processing* - Covilhã do Instituto de Telecomunicações por Raysa da Luz Oliveira, Licenciada em Tecnologia de Sistemas para Internet pelo Instituto Federal de Educação, Ciência e Tecnologia do Tocantins (Brasil), sob orientação do Doutor Mário Marques Freire, Investigador Sénior do Instituto de Telecomunicações e Professor Catedrático do Departamento de Informática da Universidade da Beira Interior, e submetida à Universidade da Beira Interior para apresentação e discussão em provas de mestrado.

Dedicatória

Primeiramente dedico essa conquista ao meu Deus que durante essa jornada e em toda a minha vida esteve sempre presente me dando saúde, sabedoria e muita força de vontade. A Ele toda a minha gratidão.

Aos meus pais, aos meus irmãos e toda a minha família, por serem meu alicerce. Por todo apoio emocional e financeiro. Saibam que essa vitória é por vocês.

Ao meu esposo Manoel, meu maior incentivador. Obrigada pela paciência e principalmente por acreditar em mim quando eu mesma deixei de acreditar. Você proporciona sentido às minhas realizações.

Aos meus amigos, que por muito tempo me mantive distante, meu imenso agradecimento por entender a minha ausência e torcer pela minha vitória.

Aos excelentes mestres que aqui tive a oportunidade de conhecer, em especial ao meu professor e orientador Mário Freire. E por último e não menos importante, gostaria de agradecer a esse país magnifico denominado Portugal. Obrigada pela experiência incrível.

> ”Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser mas graças a Deus não sou o que era antes.”
>
> Marthin Luther King

Agradecimentos

Gostaria de agradecer à Universidade da Beira Interior e ao Instituto de Telecomunicações por todo o incentivo à investigação e pelo apoio ao programa de mestrado; e ao *Czech National Grid Infrastructure MetaCentrum* pelo fornecimento do ficheiro de carga de trabalho *MetaCentrum*.

Resumo

A computação em nuvem teve um avanço considerável nos últimos anos, trazendo grandes benefícios incluindo escalabilidade, flexibilidade, acessibilidade global, melhor utilização de recursos e redução de custos, entre outros. Apesar de todos os benefícios, esta adesão e crescimento trás consigo grandes desafios como otimização do uso de recursos computacionais, redução de custos, garantia da qualidade de serviço (Quality of Service (QoS)), segurança, etc. As garantias da qualidade de serviço são estabelecidas através de *Service Level Agreement*s (SLAs), que são contratos estabelecidos entre o cliente e o fornecedor do serviço de computação em nuvem, visando especificar de forma mensurável as metas de nível de serviço a serem cumpridas, além dos papéis e responsabilidades das partes envolvidas. Este trabalho apresenta um estudo sobre cumprimento de SLAs por algoritmos de alocação de máquinas virtuais em ambientes de computação em nuvem. O trabalho tem em consideração métricas como disponibilidade, custo, tempo de conclusão de uma aplicação (*task completion time*) e nível de tolerância a faltas, avaliando o cumprimento de tais métricas em diferentes cenários.

O estudo é realizado utilizando o *framework* CloudSim Plus para modelação e execução de simulações de computação em nuvem. São introduzidos dois módulos no *framework* visando: (i) especificação de SLAs e templates de máquinas virtuais em formato *JavaScript Object Notation* (JSON), seguindo padrões do *Amazon Elastic Compute Cloud* (Amazon EC2); (ii) injeção de faltas aleatórias, permitindo avaliar como os SLAs são afetados perante o surgimento de faltas nos servidores.

Por fim, o trabalho apresenta uma proposta para automação da criação e alocação de máquinas virtuais, visando cumprir os SLAs e libertar o cliente da necessidade de especificar a quantidade mínima de máquinas virtuais para atendimento dos níveis de serviço exigidos. Mesmo com todo o nível de automação que os fornecedores de computação em nuvem possam oferecer, os resultados obtidos mostram que é possível melhorar a automação destes serviços, reduzindo a necessidade de intervenção do cliente e as violações de SLA devido a uma inadequada configuração de máquinas virtuais realizada pelo cliente.

Palavras-chave

Computação em Nuvem, Service Level Agreement, CloudSim Plus, Qualidade de Serviço, Simulação, Alocação de Máquinas virtuais

Abstract

Cloud computing has made considerable progress in recent years, bringing great benefits including scalability, flexibility, global accessibility, improved resource utilization and cost savings, among others. Despite all the benefits, this adhesion and growth carries with it great challenges such as optimization of the use of computational resources, reduction of costs, Quality of Service (QoS) assurance, security, etc. Guarantees are provided through Service Level Agreements (SLAs), which are agreements between the customer and the cloud computing service provider to measurably specify the service level goals to be fulfilled, as well as the roles and responsibilities of the parties involved. This work presents a study on compliance with service level agreements by algorithms for allocating virtual machines in cloud computing environments. The work takes into account metrics such as availability, cost, task completion time and level of fault tolerance, evaluating the compliance of such metrics in different scenarios.

The study is conducted using the CloudSim Plus framework for modeling and running cloud computing simulations. Two modules are introduced in the framework about: (i) specification of SLAs and virtual machine templates in JSON format, following Amazon Elastic Compute Cloud (Amazon EC2) standards; (ii) injection of random faults, allowing to evaluate how the SLAs are affected by the occurrence of faults in servers.

Finally, this work presents a proposal for automation of the creation and allocation of virtual machines, aiming to comply with the SLAs and free the client from the need to specify the minimum number of virtual machines to meet the required service levels. Even with all the automation level provided by cloud service providers, the obtained results show it is possible to further improve the automation of these services by reducing the need for customer intervention and SLA violations due to an inadequate configuration of virtual machines performed by the client.

Keywords

Cloud Computing, Service Level Agreement, CloudSim Plus, Quality of Service, Simulation, Virtual Machine Allocation

Índice

Lista de Figuras

Lista de Tabelas

Lista de Códigos

Lista de Algoritmos

Lista de Acrónimos

SLA Service Level Agreement

JSON JavaScript Object Notation

Amazon EC2 Amazon Elastic Compute Cloud

NIST National Institute of Standards and Technology

IaaS Infrastructure as a Service

PaaS Platform as a Service

SaaS Software as a Service

QoS Quality of Service

VM Virtual Machine

RR Round-Robin

ESCE Equally Spread Current Execution

AMLB Active Monitoring Load Balancer

TLB Throttled Load Balancing

GNU General Public License

MI Milhões de Instruções

WSLA Web Service Level Agreement

XML eXtensible Markup Language

SLAC Formal Service-Level-Agreement Language for Cloud Computing

SLM Service Level Management

SLO Service Level Objective

MTBF Mean Time Between Failures

MTTR Mean Time to Repair

PRNG Pseudo-Random Number Generator

WFD Worst Fit Decreasing

Capítulo 1

Introdução

1.1 Enquadramento da Dissertação

A computação em nuvem tem crescido cada vez mais, devido às vantagens oferecidas aos utilizadores. Segundo o quinto relatório Global *Cloud Index* da Cisco, a Cloud irá representar 83% do tráfego global de *data center* em 2019 (Cisco System, 2016).

Segundo o *National Institute of Standards and Technology* (NIST) a computação em nuvem é definida como (Mell e Grance, 2011):

> "Um modelo para permitir acesso ubíquo, cómodo, sob-demanda a uma *pool* compartilhada de recursos de computação configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente fornecidos e libertos com esforço mínimo de gestão ou interação do fornecedor de serviço, através da Internet".

A procura por serviços de computação em nuvem tem sido crescente, devido às suas inúmeras vantagens e comodidade. Ao alugar infraestrutura computacional na nuvem em vez de adquirir e manter tal infraestrutura, o cliente troca elevados custos de aquisição por baixos custos variáveis de utilização. Os serviços podem ser utilizados a partir de qualquer lugar do mundo com acesso à internet, em qualquer plataforma e a qualquer momento. O utilizador não tem a necessidade de saber onde os serviços estão alojados e como eles serão entregues. A computação em nuvem possui outras características importantes, tais como: rápida elasticidade, serviços sob demanda e *pool* de recursos entre outros.

Atualmente existem diversos modelos de serviços de computação em nuvem, conforme se descreve a seguir:

- ***Infrastructure as a Service*** (IaaS): a capacidade fornecida ao consumidor é a de providenciar o processamento, armazenamento, redes e outros recursos de computação fundamentais, onde o consumidor pode instalar e executar software arbitrário, que pode incluir sistemas operativos e aplicações. O consumidor não administra ou controla a infraestrutura da nuvem subjacente, mas possui controlo sobre sistemas operativos, armazenamento e aplicações instaladas (Mell e Grance, 2011);

- ***Platform as a Service*** (PaaS): a capacidade fornecida ao consumidor é a de instalar na nuvem infraestrutura criada pelo fornecedor e aplicações desenvolvidas pelo cliente. Tais aplicações podem ser desenvolvidas utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo fornecedor. O consumidor não faz a gestão ou controla a infraestrutura de nuvem subjacente, incluindo rede, servidores, sistemas operativos ou armazenamento, mas tem controlo sobre as aplicações instaladas e, possivelmente, as configurações do ambiente onde as aplicações são alojadas (Mell e Grance, 2011);

- ***Software as a Service*** (SaaS): a capacidade disponibilizada ao consumidor é usar as aplicações do fornecedor em execução numa infraestrutura de nuvem. Tais aplicações são acessíveis a partir de vários dispositivos clientes por meio de uma interface de *thin client*, como um navegador web ou uma interface de um programa. O consumidor não gere ou controla a infraestrutura da nuvem subjacente, incluindo rede, servidores, sistemas operativos, armazenamento ou mesmo recursos de aplicações individuais, com a possível exceção de configuração de definições de aplicações específicas do utilizador (Mell e Grance, 2011).

- ***Anything as a Service*** (XaaS): é uma abordagem que se refere à capacidade de se ter qualquer modelo como serviço (Fernandes et al., 2014). É um termo que abrange qualquer modelo, desde Data-as-a-Service (DaaS) (Vu et al., 2012), Routing-as-a-Service (RaaS) (Chen et al., 2014), Security-as-a-Service (SecaaS) (Chen et al., 2014), Malware-as-a-Service (MaaS) (Gutmann, 2007) até Windows-as-a-Service (WaaS) (Riggs, 2016).

Além dos modelos de serviços, a nuvem possui quatro modelos de implementação (Mell e Grance, 2011):

- **Nuvem Privada**: a infraestrutura de nuvem é fornecida para uso exclusivo de uma única organização e respetivos utilizadores, incluindo clientes e funcionários. Pode ser gerida e operada pela organização ou por uma terceira entidade e pode existir dentro ou fora das instalações;

- **Nuvem Pública**: é aberta para o público em geral, permitindo que qualquer pessoa ou empresa possa utilizar tais serviços;

- **Nuvem Comunitária**: a infraestrutura é fornecida para uso exclusivo de uma comunidade específica de organizações que têm preocupações compartilhadas;

- **Nuvem Híbrida**: é composta por duas ou mais infraestruturas de nuvem distintas (privadas, públicas ou comunitárias).

De entre os vários desafios da computação em nuvem, um dos maiores é proporcionar o melhor serviço para o utilizador (Ardagna et al., 2014; Wang et al., 2013; Gupta et al.,

2013). Os SLAs (Service Level Agreements ou Acordos de Nível de Serviço) existem como garantia da qualidade destes serviços para o utilizador.

O ITILv3 (ITI v3 Foundation, 2009) define SLA como um acordo entre um fornecedor de serviços de Tecnologias da Informação (TI) e um cliente. O SLA descreve o serviço de TI, documenta metas de nível de serviços e especifica as responsabilidades do fornecedor e do cliente.

Os SLAs contêm um conjunto definido de parâmetros de qualidade de serviço (Quality of Service (QoS)). Os parâmetros de QoS indicam, por exemplo, os níveis de desempenho, fiabilidade e disponibilidade oferecidos por uma aplicação, pela plataforma ou pela infraestrutura que a aloja (Ardagna et al., 2014). Estes parâmetros devem ser mensuráveis e podem ou não ser monitorizados durante a entrega dos serviços acordados no SLA. O ciclo de monitorização é bastante importante quando se refere à qualidade da entrega do serviço, pois é através da monitorização que se verifica o uso dos recursos em tempo de execução e se realiza a tomada de decisão para adequação de alocação de recursos. Tais medidas visam evitar ou corrigir problemas como ociosidade ou sobrecarga de recursos na nuvem. Nos SLAs são definidos os objetivos, as metas de qualidade de serviço e as punições.

De entre as métricas existentes, este trabalho considera um conjunto delas, tais como: custo do serviço na nuvem, disponibilidade, tolerância a faltas e tempo de conclusão de aplicações (tarefas). As métricas mais comuns são apresentadas no Capítulo 2. Num SLA cada uma dessas métricas possui valores mínimo e máximo esperados. Por exemplo, o cliente pode estabelecer que o custo por hora que ele espera pagar pelos serviços utilizados deva estar dentro de um determinado intervalo.

Esta investigação procura contribuir para que investigadores, clientes e empresas criem infraestruturas adequadas e precisas a partir de simulações, sem desperdícios de recursos, tempo e dinheiro, visto que, com o aumento da complexidade do sistema, o custo de encontrar soluções adequadas também aumenta.

Com o aumento da importância do sistema, o uso de SLAs torna-se necessário, pois é preciso um controlo mais rígido do desempenho dos serviços oferecidos. Por isso, esta dissertação é dedicada ao estudo do cumprimento de requisitos de SLAs através da alocação de máquinas virtuais.

O problema investigado nesta dissertação está descrito a seguir, juntamente com os objetivos do trabalho, as principais contribuições, as limitações do trabalho desenvolvido e a organização deste documento.

1.2 Definição do Problema e Objetivos da Investigação

O problema a investigar consiste no não cumprimento de SLAs por parte de fornecedores de computação em nuvem. Há fornecedores que poderão não se preocupar em

cumprir determinados SLAs, acabando por frustrar clientes. A importância do cumprimento destes acordos envolve garantia de qualidade, redução de custos, cumprimento de prazos e com isto trazendo confiança tanto para o cliente quanto para o fornecedor, pois a estipulação de tais acordos permite alinhar expectativas.

Este trabalho tem como objetivo principal fornecer um estudo do cumprimento de SLAs por algoritmos de alocação de máquinas virtuais em ambientes de computação em nuvem.

Como objetivos específicos a serem alcançados temos:

1. Definição e implementação de métricas de QoS no *framework* de simulação de computação em nuvem CloudSim Plus;
2. Definição e implementação no CloudSim Plus de um mecanismo de leitura de SLAs e garantia de cumprimento dos mesmos através de alocação de VMs;
3. Automação da criação e especificação da quantidade de máquinas virtuais através do custo e nível de tolerância especificado pelo cliente;
4. Especificação e implementação no CloudSim Plus de um módulo de injeção de faltas de servidores;
5. Teste dos módulos implementados para verificar o seu correto funcionamento;
6. Elaboração de experiências envolvendo alocação de máquinas virtuais, tendo em consideração os requisitos de SLA e injeção de faltas aleatórias.

1.3 Principais Contribuições

Este trabalho apresenta uma contribuição quando se trata da simulação em computação em nuvem implementada no CloudSim Plus, visto que este trabalho introduz um módulo de especificação de SLAs, permitindo avaliar o cumprimento de métricas específicas para diferentes algoritmos de alocação de recursos na nuvem. Outros trabalhos existentes ou utilizam um número reduzido de métricas, ou simplesmente não consideram estes acordos. Esta é uma falha em muitos trabalhos, pois os SLAs aumentam a eficiência dos serviços fornecidos e diminuem os riscos de possíveis faltas em aplicações.

As principais contribuições resultantes do trabalho de investigação apresentado nesta dissertação são:

- Definição e implementação de um módulo de especificação de SLAs incluindo as métricas implementadas e leitor de SLAs;
- Especificação e introdução de um módulo de injeção de faltas no CloudSim Plus;

- Implementação do Algoritmo *Worst Fit Decreasing* e comparação do respetivo desempenho com o do algoritmo *Round-Robin* para mapeamento de aplicações para máquinas virtuais;
- Definição e implementação no CloudSim Plus da possibilidade de criação de máquinas virtuais de acordo com *templates* de instâncias do *Amazon Elastic Compute Cloud* (Amazon EC2);
- Automação da criação e definição da quantidade de máquinas virtuais para tolerar faltas em ambientes de computação em nuvem, procurando otimizar os serviços e abstendo do cliente tais tarefas.

Os módulos propostos nesta dissertação e implementados no CloudSim Plus são *open source* e estão disponíveis no repositório central do simulador CloudSim Plus (Silva Filho et al., 2016).

1.4 Limitações do Trabalho Desenvolvido

O presente trabalho contém algumas limitações, tais como:

- Falta um módulo para monitorização em tempo de execução de métricas de QoS;
- Não implementa outras faltas no módulo de injeção de faltas, tais como: faltas de aplicações, de rede, etc;
- Os testes não estão validados experimentalmente em infraestruturas reais de computação em nuvem;
- São consideradas apenas aplicações de processamento de tarefas.

1.5 Organização da Dissertação

O corpo desta dissertação é composto por cinco capítulos e as referências. Após este capítulo dedicado à introdução da dissertação, os restantes capítulos estão organizados da seguinte forma:

- **Capítulo 2**: apresenta uma visão geral sobre as ferramentas, tecnologias e fundamentos utilizados e uma revisão de trabalhos relacionados encontrados na literatura.
- **Capítulo 3** descreve a ferramenta e as métricas utilizadas, o modelo do SLA em formato JSON, os diagramas de classes e algoritmos das implementações realizadas e instaladas no núcleo do simulador CloudSim Plus.

- **Capitulo 4**: apresenta um conjunto de testes utilizando algoritmos para alocação de VMs com o objetivo de cumprir SLAs, testes com o injetor de faltas implementado e testes com a automação da criação e definição da quantidade de VMs para tolerar faltas através dos requisitos do cliente.
- **Capítulo 5**: apresenta as principais conclusões e direções para trabalhos futuros.

Capítulo 2

Background e Estado da Arte

2.1 Introdução

Neste capítulo são abordados alguns conceitos sobre vários elementos que formaram a base para a realização deste trabalho e também alguns trabalhos existentes na literatura, relacionados com a simulação de alocação de recursos na nuvem, considerando requisitos de SLA.

Este capítulo encontra-se organizado da seguinte forma. A secção 2.2 é dedicada ao balanceamento de carga e alocação de recursos computacionais, sendo discutido o funcionamento e algoritmos de balanceamento de carga. A secção 2.3 é dedicada à descrição do Service Level Agreement (SLA), abordando o conceito, a estrutura, a importância, o ciclo de vida e métricas de SLAs, linguagens de descrição de SLAs, service level management e service level objective. Na secção 2.4 são descritos trabalhos relacionados com o trabalho apresentado ao longo desta dissertação e na secção 2.5 são apresentadas as conclusões deste capítulo.

2.2 Balanceamento de Carga e Alocação de Recursos Computacionais

2.2.1 Balanceamento de Carga

O balanceamento de carga é um mecanismo utilizado para equilibrar o *workload* entre diferentes máquinas virtuais, evitando servidores ociosos ou sobrecarregados, procurando um equilíbrio para o melhor aproveitamento dos recursos. A aplicação deste mecanismo na nuvem deve ser realizada de forma distribuída, flexível e extensível (Zhang e Zhang, 2010), visando:

- Melhorar o desempenho e o tempo de resposta de aplicações;
- Conseguir uma utilização ideal de recursos computacionais;
- Reduzir consumo de energia;
- Reduzir violações de SLA;
- Evitar sobrecarga dos serviços alojados na nuvem.

A Figura 2.1 apresenta o funcionamento básico de um sistema de balanceamento de carga.

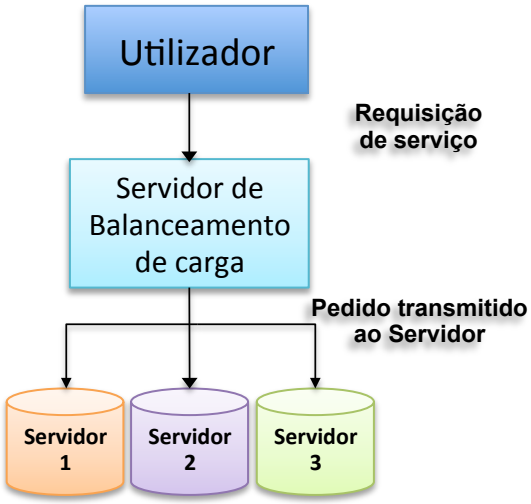

Figura 2.1: Balanceamento de Carga (Adaptado de (Science e Studies, 2014)).

Em primeiro lugar, o utilizador faz a requisição de um serviço. Cada requisição é interceptada e em seguida enviada a um servidor (nó) com melhor disponibilidade de recursos para atendê-la, procurando evitar que determinados nós fiquem sobrecarregados enquanto outros estão ociosos ou pouco utilizados. Num cenário de computação em nuvem, os serviços dos clientes são alojado em máquinas virtuais. Assim, cada máquina virtual representa um nó. O processo de balanceamento é feito através de nós redundantes, o que dá ao sistema um determinado nível de tolerância a faltas.

Os servidores são monitorados e mantidos pelos fornecedores de serviços em nuvem, fornecendo recursos computacionais aos clientes. Um fornecimento ineficiente de recursos pode levar ao aumento de custos do fornecedor, devido ao consumo excessivo de energia elétrica, desperdício de recursos computacionais, ou aumento de emissão de gases de efeito estufa. Esses problemas podem levar à violação de SLAs dos clientes (Barroso e Hölzle, 2007), pelo que a alocação ideal dos recursos em ambientes de computação em nuvem para prestação de serviço aos clientes é crucial. Existem mecanismos de fornecimento eficiente de alocação de máquinas virtuais considerando violações de SLA e minimização do consumo de energia ao mesmo tempo, como apresentado em (Lo et al., 2014; Calheiros et al., 2011; Kim et al., 2011).

Usualmente a migração de Máquinas Virtuais (Virtual Machines (VMs)) é classificada como regular *(regular migration)* ou ao vivo *(live migration)* (Lo et al., 2014). A migração regular move uma VM, interrompendo-a dentro do *host* de origem antes da migração, e em seguida copiando a VM para o *host* de destino, incluindo o estado da memória, reiniciando a VM no novo *host*. A migração ao vivo funciona de maneira similar, porém esta migração não interrompe a VM durante o processo de transição. A VM é copiada para o *host* de destino sem desligar a VM na origem. Quando termina a cópia dos dados da memória da VM para o *host* de destino, a VM é então finalizada na origem

e as conexões dos utilizadores encaminhadas para o novo *host*, sem haver quebra de conectividade.

2.2.2 Algoritmos de Balanceamento de Carga

Na literatura, foram publicados vários algoritmos para balanceamento de carga (*workloads*), de entre os quais se destacam os seguintes:

1. *Round Robin (RR)* (Nitika, Shaveta, 2012): organiza os servidores numa fila circular enviando cada requisição do utilizador para o próximo servidor na lista. Quando uma requisição é enviada para o último servidor, o algoritmo começará a percorrer a lista desde o início para as requisições seguintes.

2. *Equally Spread Current Execution (ESCE)* (Singh e Gangwar, 2014): seleciona a máquina virtual mais eficiente, de acordo com determinadas prioridades. O *workload* é distribuído de forma aleatória, verificando o respetivo tamanho e transferindo-a para uma máquina virtual que esteja com pouca carga.

3. *Active Monitoring Load Balancer (AMLB)* (Singh e Gangwar, 2014): acompanha o número de requisições a cada VM e seleciona a que estiver menos sobrecarregada. Essa política de balanceamento de carga tenta manter *workloads* semelhantes em todas as VMs disponíveis.

4. *Throttled Load Balancing (TLB)* (Zaouch, 2015): garante que apenas um número pré-definido de aplicações/tarefas são alocadas a uma única VM num referido instante. Se houver mais solicitação do que o número de VMs disponíveis, algumas das solicitações terão de ficar em fila de espera até a próxima VM ficar disponível.

5. *Greedy* (Malik et al., 2013): resolve o problema fazendo a escolha que parece melhor no momento específico. Muitos problemas de otimização podem ser resolvidos usando este algoritmo. Alguns problemas não têm solução eficiente, mas este algoritmo pode fornecer uma solução eficiente que é próxima da ótima.

2.3 Service Level Agreement (SLA)

2.3.1 Conceito, Estrutura e Importância dos SLAs

Um SLA é um documento que ajuda a definir a relação entre um cliente e um fornecedor, representando o nível da qualidade de serviço e as expectativas esperadas entre estas partes. Desta forma, um SLA é a garantia de que o fornecedor deste serviço oferece de facto, o nível de qualidade esperado pelo cliente. De acordo com (Zeginis e Plexousakis, 2010), o SLA é fundamental para os fornecedores de serviços configurarem e manterem

os compromissos com o consumidor do serviço. Tipicamente, os SLAs são definidos em texto simples, utilizando modelos ou *toolkits* (Bianco et al., 2008).

Um SLA adequadamente especificado representa cada serviço oferecido de acordo com dados tais como (Bianco et al., 2008):

- As métricas que serão coletadas;
- Quem irá coletar e como irá coletar;
- Ações a serem tomadas quando o serviço não é entregue no nível de qualidade especificado;
- Penalidades por falta de entrega do serviço no nível de qualidade especificado.

Existem diferentes tipos de SLAs e diferentes custos associados. Há clientes que necessitam de níveis elevados de disponibilidade e estão dispostos a pagar mais por isso, enquanto outros não.

A estrutura de um SLA pode conter os seguintes elementos (Zeginis e Plexousakis, 2010):

- **Objetivo**: descreve as razões por trás da criação do SLA;
- **Partes**: apresenta as partes envolvidas no SLA e respectivas funções, por exemplo, o fornecedor do serviço e o cliente;
- **Data de validade**: define o período de tempo que o SLA vai cobrir. Este é delimitado pelo início e término do prazo do contrato;
- **Plano**: define os serviços abrangidos no acordo;
- **Limitações**: define as medidas necessárias a serem tomadas para que os níveis de serviço requeridos sejam fornecidos;
- **Objetivos de nível de serviço**: define os níveis de serviço acordados entre cliente e o fornecedor do serviço. Normalmente inclui um conjunto de indicadores de nível de serviço como disponibilidade, desempenho e fiabilidade;
- **Penalidades**: define as penalidades a serem aplicadas no caso de o prestador de serviços apresentar um desempenho inferior ao esperado e acordado no SLA;
- **Termos de exclusão**: apresenta o que não será coberto no SLA;
- **Administração**: descreve os processos e os objetivos mensuráveis num SLA e define a autoridade organizacional para supervisioná-los.

Conforme apresentado acima, um dos elementos diz que para o contratante, por exemplo, é possível prever penalidades no caso de incumprimento de quaisquer serviços ou metas estabelecidas, o que pode tranquilizar o cliente em relação ao contrato acordado. Em contrapartida, o fornecedor também se protege contra quaisquer abusos ou

cobranças indevidas de serviços por conta do contratante. Deste modo, o fornecedor pode trabalhar sobre um roteiro preestabelecido. Isto permite, por exemplo, planear a contratação de pessoal especializado na medida adequada ou elaborar planos de ação contingenciais para conseguir atender às metas de serviço definidas (OpServices, 2015).

O SLA é um documento exigido em qualquer relação contratual de TI e deve ser revisto periodicamente para que tenha maior efetividade. É apenas com a revisão feita continuamente que o contratante pode ter a garantia de que a empresa de TI oferecerá suporte em todas as etapas do processo que, evidentemente, requerem cuidados e serviços diferenciados (OpServices, 2015).

Segundo (OpServices, 2015), os SLAs possuem inúmeras vantagens, tais como:

1. Promove garantias: uma certa segurança é passada tanto para o cliente quanto para o fornecedor, pois ambos contam com o apoio de uma proteção jurídica em contrato, de modo que os incumprimentos sejam evitados ou punidos;
2. Gera transparência: todas as responsabilidades são escritas e assinadas no contrato. Isto garante a transparência na relação entre as partes envolvidas;
3. Garante maior credibilidade: as garantias firmadas num documento aos clientes é parte de um estado de maturação necessário para a área de TI. As relações entre fornecedores e clientes passam a gerar mais credibilidade, para além da entrega ou prestação de bons resultados;
4. Favorece empresas de pequeno, médio e grande porte: o SLA é importante para que a qualidade na entrega de serviços seja garantida para qualquer cliente, especialmente pequenas e médias empresas. Estabelecer um SLA evita inconvenientes desnecessários, uma vez que, os SLAs costumam aumentar a qualidade de serviço percecionado pelo consumidor final, já que as empresas são melhor servidas e os fornecedores estabelecem e confirmam a sua credibilidade no mercado.

Porém, existem alguns problemas que muitas vezes levam ao não cumprimento dos níveis de serviço definidos no SLA, destacando-se os seguintes:

- Falta de comprometimento dos subfornecedores;
- Dificuldades na gestão do SLA;
- Ausência de novas tecnologias/serviços;
- Mau uso de recursos;
- Falta de proatividade.

2.3.2 Ciclo de Vida de SLAs

Os SLAs possuem um ciclo de vida definido em seis fases, conforme ilustrado na Figura 2.2 e detalhado a seguir (Bianco et al., 2008):

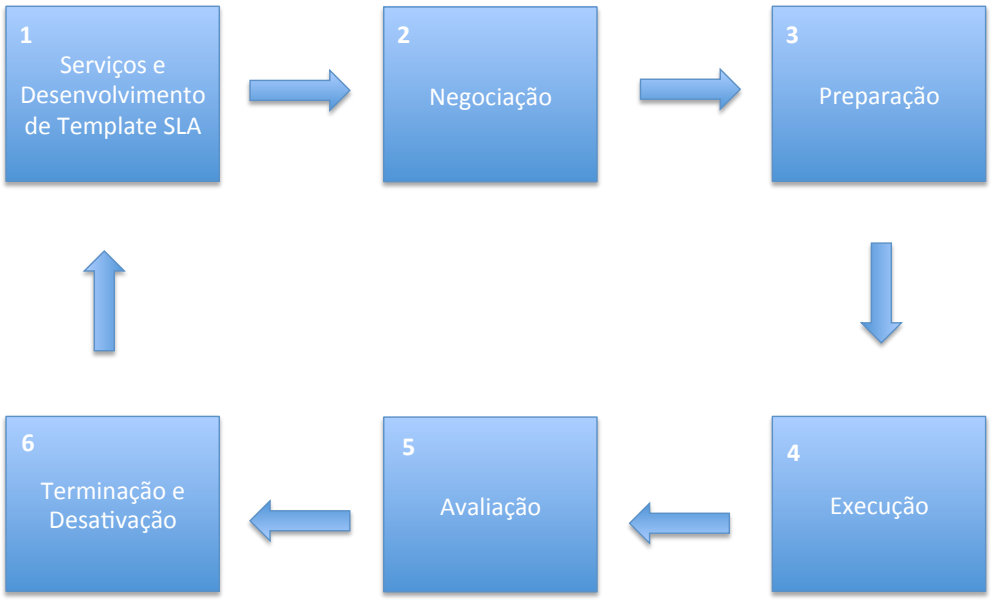

Figura 2.2: Ciclo de Vida de um SLA (Adaptado de (Bianco et al., 2008)).

1. **Serviços e Desenvolvimento de Template de SLA:** Nesta fase são identificadas as necessidades dos clientes, as características de serviços pretendidos, parâmetros que podem ser oferecidos, dado o ambiente de execução de serviço e a preparação do *template* padrão do SLA.

2. **Negociação:** Nesta fase são negociados os valores específicos para os parâmetros de serviço definidos, os custos do serviço para o cliente e o custo para o fornecedor quando o SLA é violado, bem como a definição e a periodicidade dos relatórios a serem fornecidos ao cliente.

3. **Preparação:** O serviço (ou uma instância específica do mesmo) é preparado para ser utilizado pelo cliente. Esta fase pode exigir a reconfiguração dos recursos que suportam a execução do serviço, a fim de cumprir os parâmetros de SLA.

4. **Execução:** Esta fase inclui execução e monitorização do serviço para geração de relatórios em tempo real, validação da qualidade e deteção de violação do SLA em tempo real.

5. **Avaliação:** Esta fase é dividida em duas partes:

   (a) Avaliação do SLA e a QoS que é fornecido a cada cliente. A QoS, satisfação dos consumidores, melhorias e mudanças nos requisitos são revistas periodicamente para cada SLA.

(b) Avaliação do serviço em geral. Elementos que devem ser abrangidos nesta revisão são as QoS fornecidas a todos os clientes, a necessidade de rever metas de serviço e operações, a identificação de problemas de suporte de serviço, bem como a identificação da necessidade de diferentes níveis de serviço.

6. **Terminação e Desativação:** Esta etapa envolve o término do serviço por algumas razões tais como, vencimento/violação do contrato ou desativação de serviços descontinuados.

2.3.3 Métricas de SLAs

Um SLA descreve as características técnicas e não técnicas de um serviço, incluindo os requisitos de QoS e um conjunto de métricas. Cada parâmetro contém um nome, tipo, unidade e representa uma propriedade de um objeto do serviço. Um parâmetro de SLA refere-se a uma métrica, que, por sua vez, agrega uma ou mais métricas (Zeginis e Plexousakis, 2010).

Uma métrica é uma forma de medir o desempenho ou a eficiência de alguma característica de um serviço. Quando um SLA é estabelecido, é preciso definir, além dos tipos de serviço que serão prestados, as métricas que serão utilizadas para mensurar a respetitiva qualidade a ser entregue ao cliente. Existem inúmeras métricas de QoS para SLA, referentes à qualidade de rede, desempenho, eficiência, segurança, além de outros fatores tais como o custo. Algumas destas métricas estão descritas na Tabela 2.1.

Tabela 2.1: Métricas de QoS para SLAs.

Métrica	Descrição
Disponibilidade	Disponibilidade é a probabilidade de um sistema funcionar de forma contínua sem interrupções. Para calcular esta métrica, dividi-se o tempo de atividade pelo tempo de atividade mais o tempo de inatividade do sistema (Weibull, 2017).
Taxa de transferência	Quantidade de dados transferidos do emissor para o destinatário num determinado período de tempo (Catela et al., 2014).
Tempo de conclusão da aplicação	Quantidade calculada de tempo necessário para que qualquer aplicação específica seja completada (do início ao fim) (Begum e Prashanth, 2013).
Atraso	A Quantidade finita de tempo que um pacote leva para atingir o ponto final de recepção, depois de ser transmitido a partir do terminal de envio (José Mauricio Santos Pinheiro, 2008).
Jitter	Medida de variação do atraso entre os pacotes sucessivos de um fluxo de dados (Cisco Systems, 2006).
Custo total	Retorna a despesa real do uso de serviços em nuvem (Bardsiri e Hashemi, 2014).
% CPU	Média da utilização da CPU, em percentagem (TechNet - Microsoft, 2009).
Tempo de espera	Mede o tempo que um serviço espera até o momento de ser requerido.
Confidencialidade	É a propriedade de certas informações que não podem ser disponibilizadas ou divulgadas sem autorização. (Righi et al., 2004).
Flexibilidade	Retrata a facilidade com a qual um sistema ou componente pode ser modificado para utilização em aplicações ou ambientes diferentes daquelas para que foi especificamente concebido (Bardsiri e Hashemi, 2014).
Tolerância a faltas	Capacidade do serviço executar corretamente e de maneira uniforme, mesmo em condições de falta num nó arbitrário do sistema (Bardsiri e Hashemi, 2014).
Utilização de recursos	Número de horas de trabalho atribuídas a um recurso ou grupo de recursos, como uma percentagem da sua disponibilidade para um determinado período (Begum e Prashanth, 2013).
Fiabilidade	Capacidade de um sistema executar a sua função de forma consistente, sem degradação ou falta (Bardsiri e Hashemi, 2014).
MTBF	Tempo médio entre falhas (Di e Cappello, 2015).
MTTR	Tempo médio para reparo (Uriarte et al., 2014).
Tempo de resposta	Quantidade de tempo que o servidor de aplicações leva para apresentar os resultados de uma solicitação do utilizador (ORACLE, 2010).
Tempo de inatividade do serviço	Tempo em que o serviço esteve indisponível (Catela et al., 2014).

As métricas têm um papel importante quando se trata de tomar decisões para a seleção de serviços de computação em nuvem, bem como definir e aplicar os SLAs.

2.3.4 Linguagens de Especificação de SLAs

Existem algumas linguagens para especificação de SLAs reportadas na literatura. Tais linguagens permitem descrever regras de SLA de forma legível tanto para humanos como para computadores. Desta forma, elas contribuem para a automação de sistemas de monitorização de QoS. A seguir apresenta-se uma breve descrição das linguagens de SLA.

Web Service Level Agreement

A Web Service Level Agreement (WSLA) (Ludwig et al., 2002) é uma linguagem para especificação de SLAs baseada em *Web Services* e eXtensible Markup Language (XML), permitindo que clientes e prestadores de serviços possam definir uma grande variedade de SLAs, especificando os parâmetros e identificando a forma como estes serão medidos.

Além da linguagem, existe também o *framework* WSLA (Keller e Ludwig, 2003) proposto pela *IBM Research Division* (IBM, 2017), que especifica e monitoriza SLAs para *web services*. O *framework* mede e monitoriza os parâmetros de QoS, verificando o SLA e relatórios de violações das partes envolvidas no processo de gestão de SLAs. Embora a definição da WSLA seja direcionada para *Web Services*, ela é aplicável a qualquer cenário de gestão interdomínio, tais como processos de negócio, gestão de serviços ou de redes, sistemas e aplicações em geral.

Há um entendimento comum sobre a estrutura geral de um SLA, sendo a WSLA projetada para acomodar essa estrutura em três seções (Keller e Ludwig, 2003):

- **Partes envolvidas**: identifica todas as partes contratuais. Contém a identificação e as propriedades técnicas das partes;
- **Descrição de Serviço**: especifica as características do serviço, seus parâmetros e métricas. Esta informação é processada por um serviço de medição;
- **Obrigações**: estipula várias garantias e restrições que podem ser impostas aos parâmetros de SLA definidos.

O Código 2.1 apresenta uma visão geral da estrutura principal de um documento WSLA, omitindo-se os detalhes internos.

Código 2.1: Exemplo do Código da Estrutura Principal de um Documento WSLA.

```
<?xml version="1.0">
<wsla:SLA
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:wsla="http://www.ibm.com/wsla"
        name="StockquoteServiceLevelAgreement12345" >

        <Parties>
                ...
        </Parties>

        <ServiceDefinition>
                ...
        </ServiceDefinition>
```

```
        <Obligations>
                ...
        </Obligations>

</wsla:SLA>
```

SLAng

A SLAng (Lamanna et al., 2003) é uma linguagem para descrição de SLAs desenvolvida pela *University College London* e autodenominada "uma linguagem para definição de SLAs". Os objetivos da SLAng são (Lamanna et al., 2003):

- Fornecer um formato para a negociação de propriedades de QoS;
- Fornecer os meios para captura de propriedades de QoS de forma inequívoca para a inclusão nos acordos contratuais.

Os principais recursos da linguagem são (Lamanna et al., 2003):

- **Parametrização**: inclui um conjunto de parâmetros com valores que descrevem quantitativamente um serviço;
- **Composicionalidade**: um serviço pode ser o resultado de uma cooperação entre as diferentes entidades no domínio. Uma linguagem de SLA tem de permitir tal composição;
- **Validação**: antes de iniciar um SLA, os fornecedores devem ser capazes de verificar a sua sintaxe e consistência para validá-lo;
- **Monitorização**: as partes devem ser capazes de monitorizar automaticamente a extensão para que os níveis de serviço estabelecidos no acordo sejam efetivamente prestados pelos seus fornecedores;
- **Garantia**: uma vez que os níveis de serviço sejam acordados, *routers*, sistemas de gestão de base de dados, *middleware* e servidores *web* podem ser estendidos para cumprir os níveis de serviço de uma forma automática, usando técnicas como *caching*, replicação, *clustering* e *farming*.

A sintaxe SLAng é definida utilizando XML e pode ser integrada com linguagens de descrição de serviço existentes.

SLAC

Outra linguagem para especificação de SLAs em computação em nuvem é a Formal Service-Level-Agreement Language for Cloud Computing (SLAC) (Uriarte et al., 2014). O SLAC concentra-se em (Uriarte et al., 2014):

- Aspectos formais de SLAs;
- Apoio de acordos multi-parceiros;
- Aspectos comerciais e de serviços públicos;
- Gestão pro-ativa do acordo de SLA.

A base do SLAC é o *SLAC Management Framework*, desenvolvido para apoiar a especificação, avaliação e execução de SLAs em sistemas de nuvem utilizando a plataforma *OpenNebula* (OpenNebula, 2008).

O *framework* SLAC possui dois componentes principais:

- O *Scheduler Service*, que recebe e processa os pedidos dos clientes após a fase de negociação;
- O *SLA Evaluator*, que recebe o SLA escrito em SLAC, analisa-o e gera um conjunto de restrições correspondente à especificação, juntamente com a definição de serviço que são enviados para o *Scheduler* para poder fazer a instalação.

Tais componentes são integrados num sistema de monitorização que recupera os dados para a avaliação do SLA.

2.3.5 Service Level Management

A Gestão de Nível de Serviço (Service Level Management (SLM)), controla os SLAs, desde a sua negociação até à apropriada documentação de níveis de serviço que atendam as necessidades do negócio e à sua aplicação. O SLM tem como objetivo manter e melhorar a qualidade de serviço através de um ciclo contínuo que envolve o acordo, a monitorização e a reportagem dos níveis de serviço em termos de qualidade, quantidade e custo. É a principal área de prática para gestão e manutenção de QoS. Esta área de processo concentra-se na melhoria da QoS, revendo continuamente a qualidade dos serviços prestados por uma organização de TI (Bianco et al., 2008).

2.3.6 Service Level Objective

Dentro dos SLAs encontram-se os Service Level Objectives (SLOs) (Sturm et al., 2000), que são características mensuráveis específicas do SLA, como as especificadas na Tabela

2.1. Cada SLO corresponde a uma única característica de desempenho relevante para a prestação de um serviço. As partes acordadas determinam o que é um desempenho aceitável ou inaceitável dentro do negócio, determinando assim os SLOs. A definição de um bom desempenho para o serviço do cliente depende das próprias necessidades e prioridades do mesmo. Os objetivos dos SLOs devem ser realistas em conformidade com o orçamento do cliente, compreensíveis e mensuráveis.

Os SLOs devem indicar o que se espera do sistema em termos específicos para cada categoria. Alguns objetivos comuns podem conter:

- Média do tempo de resposta;
- Disponibilidade do sistema;
- Tempo de inatividade.

(Sturm et al., 2000), expressaram que os SLOs têm de ser:

- Atingíveis;
- Repetíveis;
- Mensuráveis;
- Compreensíveis;
- Significantes;
- Controláveis;
- Acordados pelas partes;
- Devem ter uma penalidade ou gratificação acessíveis.

Um SLO geralmente é definido em termos de cumprir um nível de serviço, dentro da métrica acordada, num determinado período de tempo, explicando como e onde ele deve ser medido. Por exemplo, “99,99% de disponibilidade em algum sistema por um período de cinco meses”.

2.4 Trabalhos Relacionados

(Rak et al., 2013) proposeram uma arquitetura de um *framework* para gestão de SLAs em ambientes de computação em nuvem, identificando as necessidades implícitas pela simulação a ser executada e sugerindo a adoção de um mecanismo de simulação que se encaixe com os requisitos propostos. Este *framework* explora as previsões de desempenho obtidos através de simulações em cada etapa do ciclo de vida do SLA.

A Figura 2.3 apresenta a arquitetura do *framework* proposto.

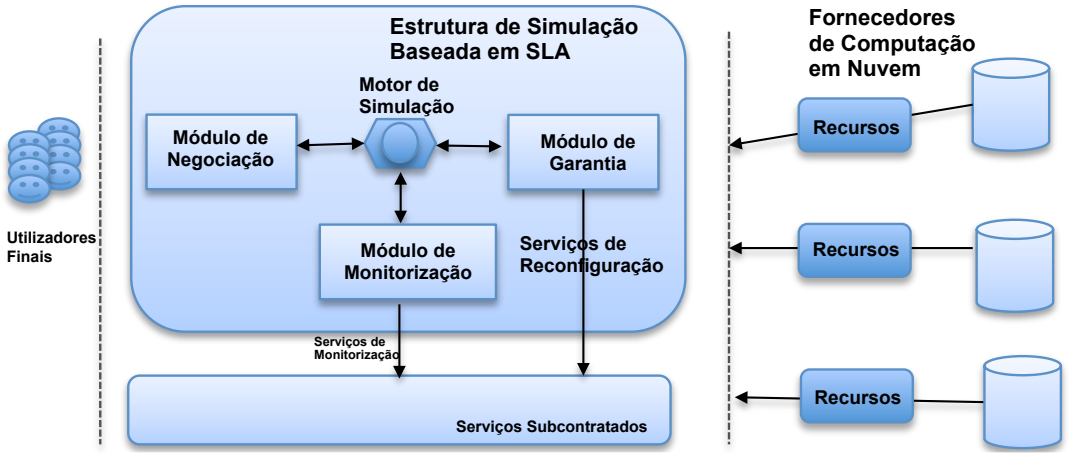

Figura 2.3: Framework de Simulação Baseada em SLA (Adaptado de (Rak et al., 2013)).

A arquitetura apresenta três módulos principais dedicados às fases de SLA:

- **Módulo de Negociação**: define acordos de SLA sobre um determinado nível de serviço, levando em conta tanto as necessidades do utilizador quanto as capacidades de fornecedores de serviços.
- **Módulo de Monitorização**: gera alertas quando existem condições que podem levar a violações do acordo e quando o SLA é de facto violado.
- **Módulo de Garantia**: realiza todas as ações necessárias para a concessão de SLAs. Pode ocorrer em duas fases diferentes do ciclo de vida do SLA:
  - quando um novo SLA é assinado;
  - durante o ciclo de vida do serviço oferecido.

Cada um destes módulos utiliza o mesmo mecanismo de simulação (motor de simulação) para tomar decisões sobre o SLA (a fim de aceitar ou recusar, gerar alertas e/ou tomar devidas medidas). Os autores propõem o uso de simulação em cada fase do ciclo de vida do SLA, que implica um elevado número de simulações simultâneas a serem realizadas. No entanto, tanto quanto é do nosso conhecimento, esta proposta ainda não foi implementada até a data de escrita desta dissertação.

(Di e Cappello, 2015) proposeram o GloudSim, um simulador de nuvem com base em *traces* de *data centers* da Google, produzidos a partir do log de milhares de aplicações e milhões de *jobs*/*tasks* em execução em mais de 12.000 *hosts* heterogeneos. O simulador foi implementado na linguagem de programação Java e é capaz de processar simultaneamente centenas de *jobs* em paralelo. Os investigadores podem observar claramente o número de execução de tarefas e também as tarefas que foram concluídas ao longo do tempo. Apesar do trabalho utilizar com sucesso *traces* para fazer simulação na nuvem, o simulador não possui um sistema de monitorização efetivo para controlar/verificar SLAs.

(Calheiros et al., 2011) proposeram uma abordagem adaptativa com foco na automação de tarefas de gestão de rotina, entrega flexível de recursos de TI virtualizados e aplicações quando e onde necessário, aumento e redução da capacidade do sistema sem exceder o fornecimento e sem ter um impacto inaceitável nas QoS alvo.

Tal técnica de fornecimento tenta cumprir as metas de QoS, que inclui o tempo de resposta e a taxa de rejeição de requisições dos serviços, evitando o excesso de fornecimento de recursos de TI e otimizando o uso destes.

Foi feita a modelação do comportamento e do desempenho de diferentes tipos de aplicações e recursos de TI para transformar, de forma adaptativa, as solicitações de serviços do utilizador. (Calheiros et al., 2011) utilizaram o desempenho analítico (sistema de filas) e informações de *workload* para fornecer entrada inteligente sobre os requisitos do sistema para um fornecedor de aplicações.

De acordo com (Calheiros et al., 2011), as principais contribuições do trabalho são:

1. Desenvolvimento de uma técnica de fornecimento adaptativo, através de um algoritmo, com base no desempenho analítico e informações de *workload*. A técnica permite determinar e capturar a relação entre metas de QoS de aplicações e a alocação dinâmica de recursos de TI individuais;
2. Uma análise de dois *workloads*, conhecidos e específicos da aplicação, com o objetivo de demonstrar a utilidade da modelação do *workload* ao fornecer *feedback* para o fornecimento de ambiente de computação em nuvem;
3. Análise orientada por simulação, com base em modelos realistas de *workloads* em ambiente de produção.

De acordo com a arquitetura proposta, apresentada na Figura 2.4, a **Camada SaaS** contém um mecanismo de **Controlo de Admissão** com base no número de requisições em cada instância das aplicações. Se todas as instâncias de aplicações virtualizadas tiverem **k** pedidos em suas filas, os novos pedidos são rejeitados, uma vez que podem violar o tempo de resposta. As solicitações aceites são encaminhadas para a **Camada de PaaS** do fornecedor, que implementa o sistema proposto.

Existem alguns componentes críticos para o funcionamento do sistema (Calheiros et al., 2011):

- **Fornecedor de Aplicações**: recebe os pedidos aceites e fornece as máquinas virtuais e instâncias de aplicações baseados na entrada do **Analisador de Carga de Trabalho** e do **Módulo de Previsão de Carga e Modelador de Desempenho**. Os pedidos aceites são encaminhados para as **Instâncias de Aplicações Virtualizadas**, que são capazes de processar o pedido utilizando o algoritmo *round-robin*.
- **Analisador de Carga de Trabalho**: gera estimativas de demandas futuras para a aplicação. Esta informação é passada para o **Módulo de Previsão de Carga e Modelador de Desempenho**;

- **Módulo de Previsão de Carga e Modelador de Desempenho**: decide o número de instâncias de aplicações virtualizadas necessárias para cumprir as metas de QoS.

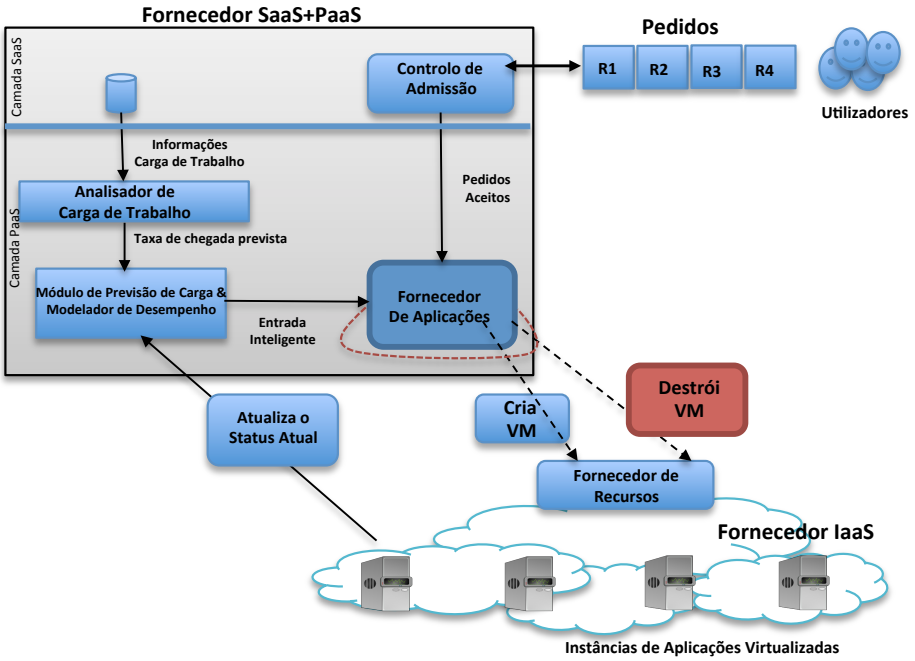

Figura 2.4: Mecanismo para Fornecimento Adaptativo de Máquinas Virtuais (Adaptado de (Calheiros et al., 2011)).

Este mecanismo é executado continuamente para assegurar que as metas de fornecimento sejam cumpridas em todos os momentos. Foram definidos alguns objetivos de projeto para esta abordagem de fornecimento (Calheiros et al., 2011):

- **Automação**: todas as decisões relacionadas com o fornecimento devem ser feitas automaticamente;
- **Adaptação**: o **Fornecedor de aplicações** deve adaptar-se às mudanças na intensidade do *workload*;
- **Garantia de boa execução**: a alocação de recursos no sistema pode ser dinamicamente variada para garantir o cumprimento de metas de QoS.

O CloudSim (Calheiros et al., 2009) foi utilizado para a modelação do ambiente de computação em nuvem. Tal trabalho contém poucos parâmetros de QoS, além da falta de uma gestão de SLA com a finalidade de gerir eficazmente as violações de qualidade de serviço.

(Aslanpour e Dashti, 2016) proposeram uma estrutura para alocação automática de recursos utilizando a combinação de métodos reativos e proativos através de simulação. Os recursos são fornecidos através do comportamento das VMs e do SLA, onde os parâmetros utilizados para a tomada de decisão estão diretamente ligados ao uso da CPU e tempo de resposta. A proposta foi implementada utilizando o framework de simulação

CloudSim. Tal trabalho utiliza apenas duas métricas e o código fonte não foi disponibilizado, além de o trabalho utilizar apenas a política de alocação definida por omissão no CloudSim.

(Nita et al., 2014) apresentaram um módulo de injeção de faltas. Para a implementação foi utilizado o simulador CloudSim, porém os autores optaram por integrar o módulo com o *CloudReports*, uma ferramenta gráfica para o CloudSim. O módulo segue o modelo baseado em eventos e insere faltas no CloudSim com base em distribuições estatísticas. Contudo, o injetor de faltas não se encontra disponível no repositório do *CloudReports*.

2.5 Conclusões

Para a construção de um acordo de SLA é necessário utilizar uma linguagem que o expresse de forma clara e objetiva, evitando dúvidas e duplas interpretações. Neste capítulo foram apresentadas diversas linguagens que são utilizadas para definir SLAs. Estas linguagens compreendem todo o acordo entre cliente e fornecedor, desde a parametrização à garantia dos serviços. Estas linguagens não foram usadas no trabalho, tendo-se optado pelo uso do formato que o *Amazon CloudWatch* utiliza por ser mais simples de gerir e ler.

Os trabalhos relacionados listados neste capítulo não possuem muitas métricas para avaliação do desempenho do sistema, além de alguns trabalhos não estabelecerem contratos de níveis de serviço. Esta dissertação é focada nas métricas de qualidade de serviço visto que é bastante importante gerir e controlar serviços de TI, sempre visando a satisfação do utilizador.

Este capítulo forneceu uma base sobre o assunto, enfocando os pontos mais importantes sobre a tecnologia e afins.

Capítulo 3

Modelação e Implementação

3.1 Introdução

Neste capítulo são apresentadas de forma detalhada a ferramenta utilizada para a realização das simulações, as métricas que foram utilizadas e a descrição das implementações do módulo instalado no núcleo do simulador CloudSim Plus com diagramas de classes e algoritmos.

Este capítulo encontra-se organizado da seguinte forma. A secção 3.2 apresenta uma perspetiva geral sobre o *Framework* de Simulação CloudSim Plus. A secção 3.3 aborda as métricas Utilizadas nos SLAs, a secção 3.4 descreve o formato para construção de SLAs e a secção 3.5 descreve as implementações realizadas no CloudSim Plus. A secção 3.6 apresenta as conclusões deste capítulo.

3.2 O Framework de Simulação CloudSim Plus

O CloudSim Plus (Silva Filho et al., 2016; Silva Filho et al., 2017) é um *framework* para simulação de computação em nuvem, desenvolvido em Java e licenciado através da *General Public License* (GNU)(GPLv3). O simulador pode ser utilizado para realização de testes baseados em cenários e configurações específicas de ambientes físicos de fornecedores de computação em nuvem, permitindo a gestão de todo o ciclo de vida de VMs e aplicações. O ciclo de vida de uma máquina virtual inclui as ou algumas das seguintes etapas: a criação, configuração, clonagem, migração, inicialização, reinicialização e destruição (Khajeh-Hosseini et al., 2012) .

O CloudSim Plus oferece diversos recursos como suporte para modelação e simulação de:

1. *Data centers* de computação na nuvem de grande escala;
2. *Hosts* de servidores virtualizados, com políticas personalizáveis para fornecimento de recursos do *host* para máquinas virtuais;
3. Recursos computacionais cientes do consumo de energia;
4. Topologias de *data centers* e aplicações de transmissão de mensagens;
5. Inserção dinâmica de elementos de simulação;

6. Políticas para fornecimento de *hosts* para máquinas virtuais e alocação granular de recursos de tais *hosts* para as respetivas VMs.

O CloudSim Plus permite modelar ***data centers*, VMs, aplicações** (também designadas por ***cloudlets*), *brokers*** e ***hosts***. O *framework* permite ainda modelar diversas características destas entidades, como capacidade computacional de *data centers*, *hosts*, requisitos de VMs e aplicações. É possível definir os custos associados a cada tipo de recurso computacional (custo de processamento, memória, etc) e podem ser implementadas e avaliadas diversas políticas, como por exemplo políticas de seleção de *hosts* para migração de VMs.

Um ***host*** implementa as características básicas de uma máquina física dentro de um *data center*, apresentando parâmetros como: capacidade de fornecimento de memória RAM e largura de banda, capacidade de armazenamento e cores de CPUs.

Um ***broker*** é responsável por tomar decisões em nome de um determinado cliente, como quantas máquinas virtuais serão alocadas para determinado cliente, para qual *data center* enviar tais VMs e em quais VMs as aplicações vão ser executadas. As **VMs** são alocadas dentro de *hosts*. Cada VM possui um proprietário (broker), com características como número de cores de CPUs, quantidade de RAM, espaço de armazenamento e largura de banda.

Cada ***cloudlet*** criada representa uma aplicação a ser executada por uma VM com características reais tais como: tamanho (dado em Milhões de Instruções (MI)) e cores requiridos.

Diagrama de Pacotes do *Framework* CloudSim Plus: Na Figura 3.1 são apresentados todos os pacotes do *framework* CloudSim Plus. Os pacotes destacados na cor verde são os especificados e implementados com este trabalho.

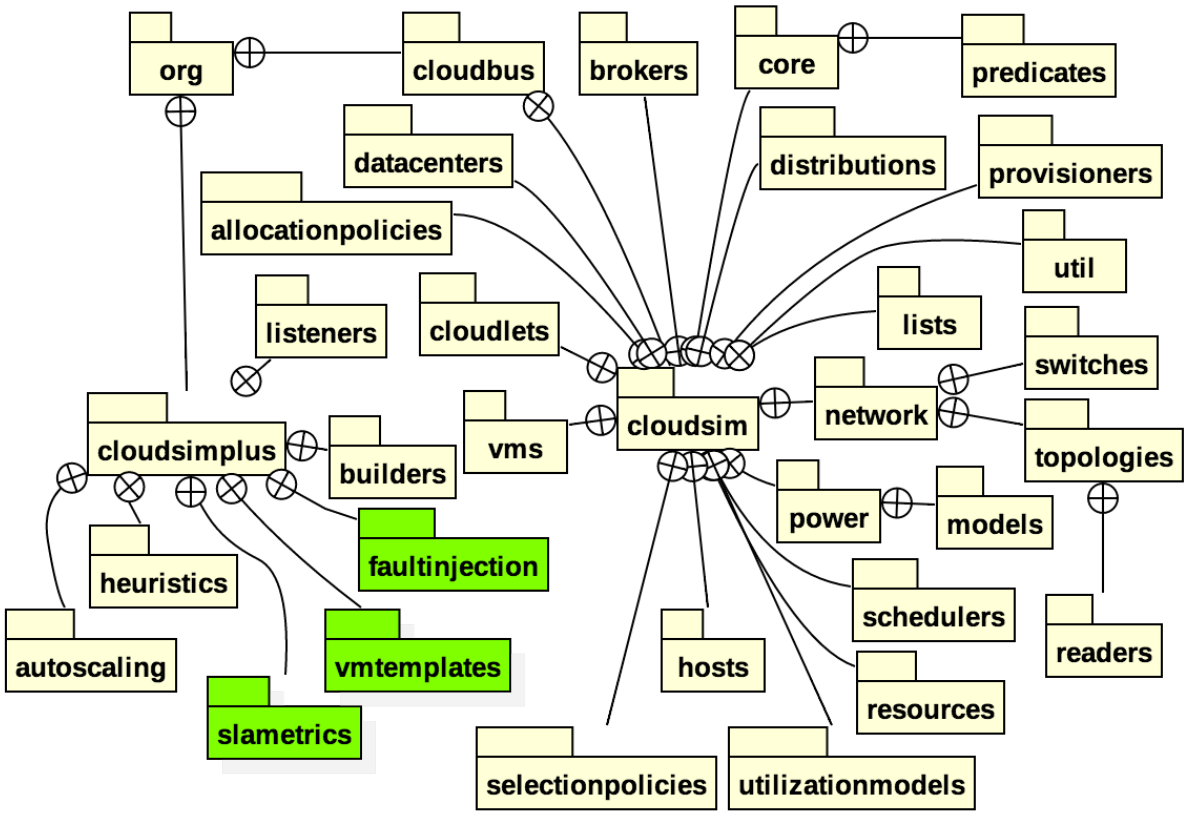

Figura 3.1: Diagrama de pacotes do CloudSim Plus, ilustrando a inclusão dos módulos implementados assinalados a cor verde.

O pacote `faultinjection` possui classes para gerar faltas aleatórias de hosts e também para criar clones quando falharem VMs vinculadas a determinados clientes. O pacote `slametrics` possui classes para leitura de SLAs em formato JSON. Por último, o pacote `vmtemplates` possui classes para leitura de templates de VMs do Amazom EC2. As classes de tais pacote são apresentadas nas subsecções seguintes.

3.3 Métricas Utilizadas nos SLAs

No presente estudo, foram implementadas algumas das métricas descritas na Tabela 2.1, tais como:

Custo: Representa o custo da alocação de VMs na nuvem (Amazon, 2016).

$$custoTotal = precoHora * totalHoras; \tag{3.1}$$

O custo total é definido pelo valor cobrado por hora de uso da VM, multiplicado pelo

tempo utilizado. Para as experiências foram utilizados os valores de custo definidos pelo serviço *Amazon EC2*, de acordo com a demanda do cliente.

Tempo de Conclusão da Tarefa: Esta é uma métrica de usabilidade onde a duração total da tarefa representa a medida de eficiência e produtividade do sistema (Morse, 2017).

$$tempoConclusaoTarefa = T_F - T_S \tag{3.2}$$

O T_F é o tempo em que uma aplicação termina sua execução e o T_S é o tempo que a aplicação é submetida. Ou seja, o tempo em que a aplicação inicia menos o tempo em que a aplicação finaliza a execução.

Mean Time Between Failures (MTBF): O MTBF (Opservices, 2015), em português tempo médio entre faltas, é um indicador de desempenho que representa a média de tempo decorrido entre uma falta e a próxima vez que a falta poderá ocorrer. A fórmula para encontrar o MTBF é:

$$MTBF = \frac{tempoAtividade}{totalFaltas} \tag{3.3}$$

É o tempo total de funcionamento correto num período, dividido pela quantidade de faltas.

Mean Time to Repair (MTTR): O MTTR (Opservices, 2015), em português tempo médio para reparo, mede o tempo previsto até o reparo ou recuperação do sistema após uma falta.

$$MTTR = \frac{tempoInatividade}{totalFaltas} \tag{3.4}$$

É o total de tempo que o sistema ficou indisponível por causa de faltas.

Disponibilidade: Disponibilidade (Weibull, 2017) é a probabilidade de um sistema funcionar de forma contínua sem interrupções. Para calcular esta métrica, dividi-se o tempo de atividade pelo tempo de atividade mais o tempo de inatividade do sistema.

$$disponibilidade = \frac{MTBF}{MTBF + MTTR} \tag{3.5}$$

3.4 Formato dos SLAs

Para construir o SLA, seguiu-se o formato JSON que o *Amazon CloudWatch* utiliza (Amazon, 2010). A justificação para a utilização desse formato consistiu em:

- Facilidade de leitura;
- As linguagens especificadas em 2.3.4 são muito complexas para serem utilizadas num ambiente de simulação, sendo que se consegue expressar tais métricas de forma simplificada em JSON;
- Os parâmetros encontrados no formato do *CloudWatch* foram adequados para este trabalho.

O Código 3.1 apresenta um trecho do código de um contrato definido em JSON.

Código 3.1: Exemplo de um SLA em JSON.

```
{
    "metrics": [
    {
        "name": "TaskTimeCompletion",
        "dimensions": [{

            "name": "minValue",
            "value": 0,
            "unit": "Milliseconds"
        },
        {
            "name": "maxValue",
            "value": 30,
            "unit": "Milliseconds"
        }]},

        {
        "name": "Availability",
        "dimensions": [{

            "name": "minValue",
            "value": 99.90,
            "unit": "%"
```

```
        },
        {
            "name": "maxValue",
            "value": 100.00,
            "unit": "%"
        }]}
    ]
}
```

Neste exemplo, temos o nome da métrica e as dimensões de cada uma. Uma dimensão é um par nome/valor que identifica exclusivamente uma métrica. Cada métrica contém características específicas. Para cada dimensão de uma métrica pode haver um valor máximo, mínimo ou ambos. Tais valores fazem parte das obrigações do fornecedor, correspondendo a valores limites aceitáveis para cada métrica estabelecida. A violação do SLA ocorre quando tais limites são ultrapassados.

3.5 Implementações no CloudSim Plus

Nesta seção são apresentadas as implementações realizadas no CloudSim Plus.

3.5.1 Leitura de SLAs

Foi implementado no simulador um módulo dedicado à leitura de SLAs. A partir desta implementação, podem ser criados e lidos contratos em formato JSON com o objetivo de especificar metas de serviços entre clientes e fornecedores.

O diagrama de classes da Figura 3.2 representa a modelagem do contrato SLA, definido em formato JSON no Código 3.1.

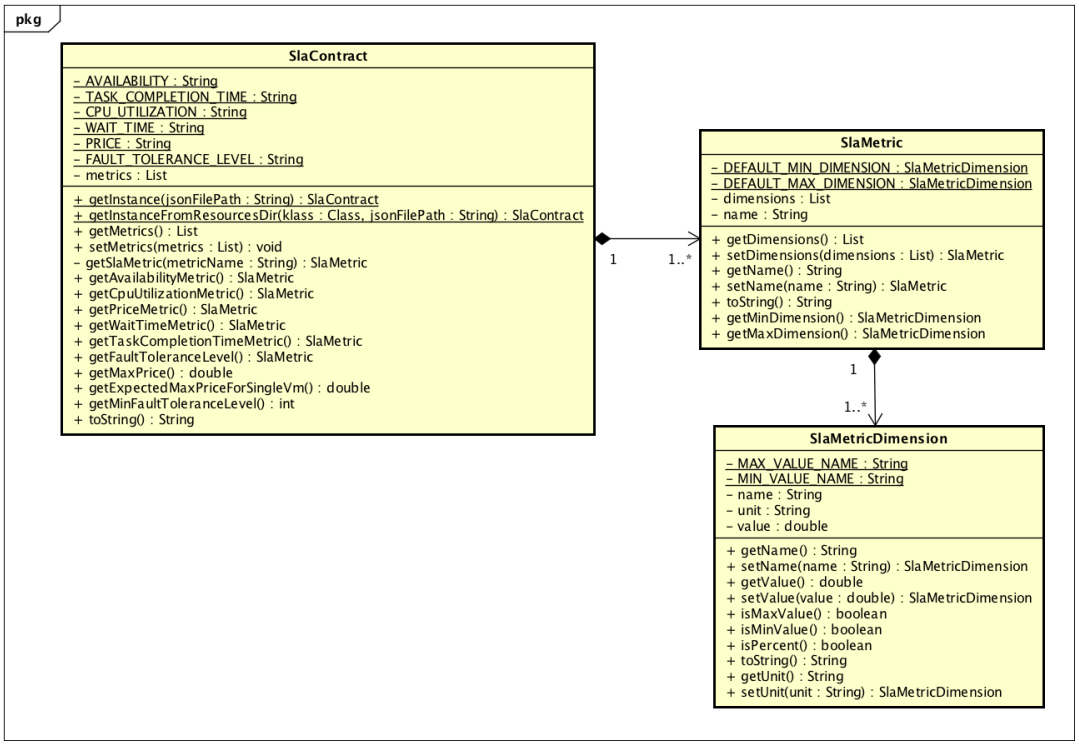

Figura 3.2: Diagrama de Classes - Módulo de Leitura de SLAs.

3.5.2 Módulo de Injeção e Recuperação de Faltas

Foi implementado no simulador, um injetor de faltas a nível de cores de CPUs dos *hosts* de um *data center*, utilizando um Pseudo-Random Number Generator (PRNG). O módulo possui um conjunto de classes que permitem definir o tipo de gerador a ser utilizado, seguindo distribuições estatísticas como a de Poisson. A classe `HostFaultInjection` é responsável por gerar os tempos da falta, selecionar um *host* para falhar e definir o total de cores naquele *host* que irão falhar.

Foram definidas três possíveis circunstâncias para a ocorrência de uma falta, como descrito a seguir:

1. Falta de todas os cores da CPU do *host*: se todos os cores do *host* falharem, todas as VMs que estão alocadas neste *host* irão falhar, sendo imediatamente destruídas.

2. Total de cores da CPU do *host* é maior que o requerido pelas VMs: mesmo após possíveis faltas, se o *host* ainda assim possuir mais cores funcionando do que requerido pelas VMs, a falta não afetará tais VMs.

3. Cores da CPU do *host* em funcionamento é menor que o requerido pelas VMs: o *host* teve uma falta, resultando num total de cores em funcionamento menor do que o requerido pelas VMs. Neste caso, a diferença entre o total de cores requerido por tais VMs e o total de cores em funcionamento representa o número de cores a serem removidas das VMs.

Por exemplo, considere um *host* inicialmente com 8 cores. Se uma falta fizer o *host* perder 4 cores, restarão 4 em funcionamento. Se houver apenas uma VM nesse *host* e a VM requisitar 6 cores, a VM irá continuar seu funcionamento porém 2 cores serão desalocados. Se havia 2 aplicações correndo na VM, cada uma usando 3 cores, o desempenho das aplicações será afetado uma vez que cada uma terá um core a menos para usar.

Algoritmo: Como descrito acima, há três circunstâncias possíveis para a ocorrência da falta, as quais estão representadas nas condições no Algoritmo 1.

```
failedCores = random.sample()
for i ← 1 to failedCores do
    host.setCoreStatus(i, FAILED)
end
if host.getWorkingCores() == 0 then
    for vm in host.getVmList() do
        vm.destroy()
    end
end
else if host.getHostWorkingCores() >= host.getVmsRequiredCores() then
    printNoVmAffectedByFault()
end
else
    i ← 0
    coresToRemove = failedCores
    while host.getVmList().size() > 0 and coresToRemove > 0 do
        i ← i%host.getVmList().size()
        Vm vm ← host.getVmList().get(i)
        host.deallocateOneCoreFromVm(vm)
        coresToRemove − −
        i + +
    end
end
```

Algoritmo 1: Gerador de faltas para o host.

A quantidade de cores falhos do *host* será definida e a partir desse valor são feitas as verificações. Se a quantidade de cores disponíveis (ou seja, os cores que não foram afetados pela falta) for igual a zero, implica a destruição das VMs que foram alocadas para determinado *host*. Se a quantidade de cores disponíveis for maior ou igual à quantidade que as VMs requerem para serem executadas, as VMs não são afetadas. Na última condição, se a quantidade de cores disponíveis for menor que a quantidade que as VMs requerem para a sua execução, serão desalocados cores das VMs. A lista de VMs será percorrida de forma cíclica até que todos os cores necessários sejam removidos. Se todos os cores de determinada VM forem desalocados, ela será destruída, caso contrário continuará a sua execução consequentemente com menos cores.

Para recuperação das faltas foi utilizada a simulação de *Snapshots* de VMs. Antes do

host falhar, é feita uma cópia da VM. Se o *host* vier a falhar completamente, ou seja, se todos os seus cores falharem, todas as VMs que estavam em execução neste *host* irão falhar. Neste caso, é submetido um *backup* da VM com as mesmas aplicações para outro host que esteja em funcionamento.

Diagrama de classe: O módulo de injeção de faltas implementado possui a classe `HostFaultInjection` onde estão todos os métodos necessários para gerar faltas aleatórias de *host*. A classe `HostFaultInjection` é uma entidade que monitora eventos gerados durante a simulação e define aleatoriamente quando faltas devem ser injetadas, para qual *host* e quantos cores serão afetados. A classe usa uma implementação da interface `VmCloner`, como a `VmClonerSimple`, para criar clones de VMs vinculadas ao cliente quando todas as VMs associadas a este cliente forem destruídas.

A classe `VmClonerSimple` fornece uma implementação básica que permite a clonagem de uma VM destruída devido a uma falta de *host*. Esta classe fornece todos os recursos para clonar uma VM, simulando a criação de outra VM a partir de um *snapshot* da VM falha, permitindo também reinicializar aplicações que estavam sendo executadas dentro de tal VM.

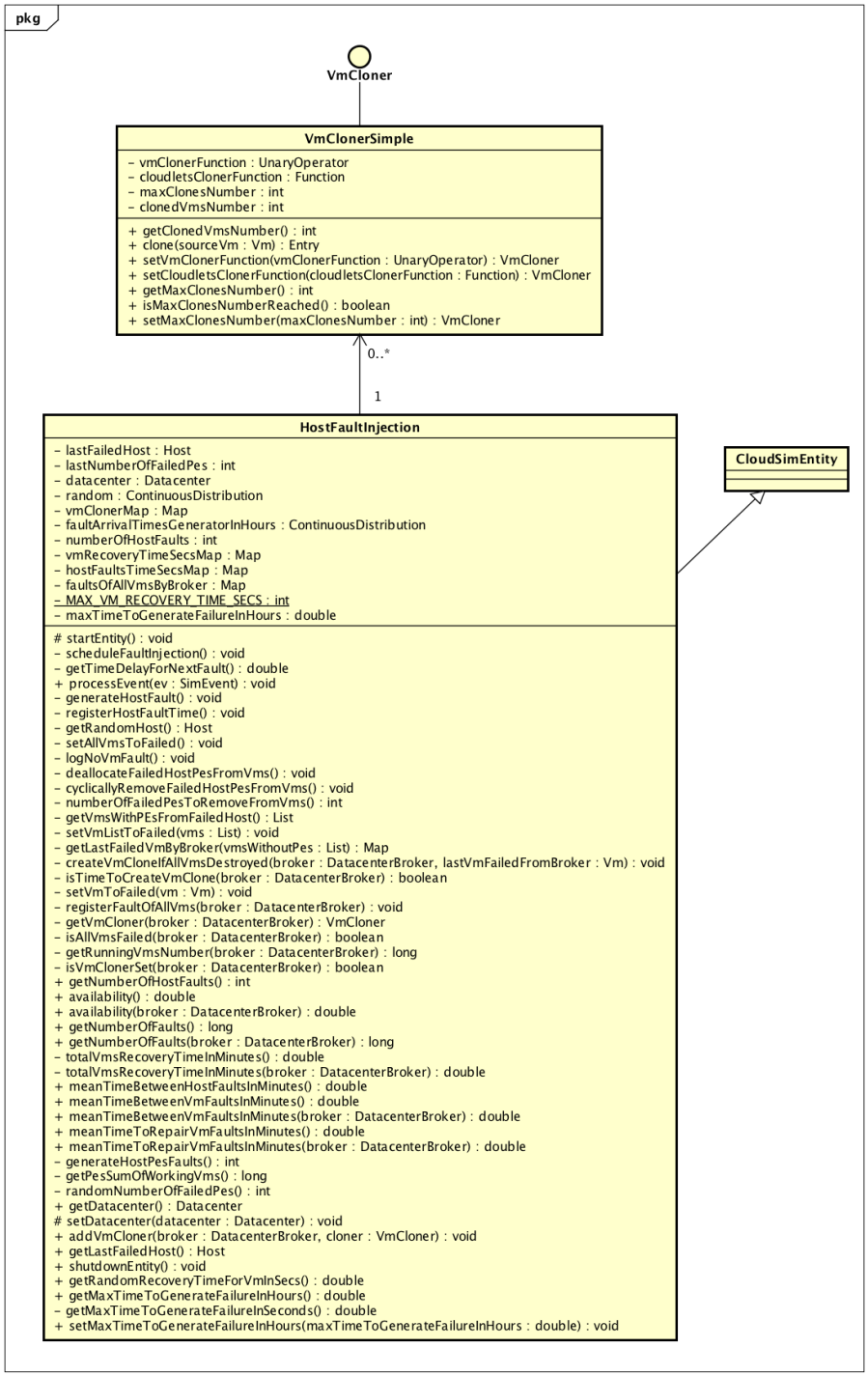

Figura 3.3: Diagrama de Classes - Módulo de Injeção e Recuperação de Faltas.

3.5.3 Algoritimo Worst Fit Decreasing (WFD) para Mapeamento de Aplicações para VMs

Foi implementado neste trabalho o algoritmo *Worst Fit Decreasing (WFD)* (Wang et al., 2004), com o intuito de minimizar o tempo de execução da aplicação. Com este algoritmo, as aplicações que tiverem o maior tempo de conclusão esperado serão mapeadas para VMs com maior capacidade. Para a implementação foram considerados alguns

fatores importantes, tais como:

- Poder de processamento da VM, incluindo quantidade e capacidade dos cores;
- Estimativa de tempo de execução da aplicação.

O CloudSim Plus possuia apenas o algoritmo RR para mapeamento de aplicações, que não considera nenhum fator para selecionar as aplicações. Tal algoritmo faz com que cada aplicação seja mapeada para uma VM diferente, de forma cíclica, controlando todos os processos sem qualquer prioridade.

Algoritmo: A seguir, no Algoritmo 2, é apresentado.

Input: cloudlet, vmList
Output: The vm
$vmList.sortDecreasing($**(**$vm.expectedCloudletCompletionTime(cloudlet)$**)**
$.sortDecreasing($**(**$vm.getExpectedNumberOfFreePes()$**)**
return $vmList.get(0)$

Algoritmo 2: Mapeando aplicações para VMs.

Utilizando o WFD, as aplicações e as VMs são ordenadas de forma decrescente: as VMs pela quantidade de cores livres e as aplicações pela sua estimativa de finalização. O tempo esperado de conclusão da aplicação é encontrado dividindo o tamanho da aplicação pela capacidade em MIPS da VM. Dessa forma, a aplicação que levar mais tempo a executar será encaminhada para a VM com maior capacidade (que é a primeira da lista), tornando assim a execução das aplicações mais eficiente.

3.5.4 Leitura de *Templates* com Configurações de Instâncias do Amazon EC2

Além da leitura de SLAs, foi introduzido um módulo para a leitura de configurações de instâncias de VMs no formato JSON. Tais configurações foram definidas seguindo os padrões do *Amazom AWS EC2* (Amazon, 2016). Com essa implementação, é possível escolher o tipo de VM adequada para os requisitos do cliente (como custo, por exemplo), a partir de um conjunto de *templates* de VMs, seguindo configurações de VMs reais fornecidas pelo serviço do Amazon.

O Código 3.2 apresenta alguns desses *templates* introduzidos no CloudSim Plus.

Código 3.2: Exemplo de um ficheiro de configurações de instância Amazon EC2 em JSON.

```
{
  "instanceName": "t2.nano",
  "vCPU":1,
  "memoryInMB":512,
  "pricePerHour":0.0082
},

{
  "instanceName": "t2.micro",
  "vCPU":1,
  "memoryInMB":1024,
  "pricePerHour":0.017
},

{
  "instanceName": "t2.medium",
  "vCPU":2,
  "memoryInMB":4096,
  "pricePerHour":0.065
},

{
  "instanceName": "p2.xlarge",
  "vCPU":4,
  "memoryInMB":62464,
  "pricePerHour":1.084
}
```

Cada ficheiro JSON representa as configurações de uma determinada instância de VM do *Amazon* EC2. Cada tipo de VM pode ser utilizada para diversos casos de uso, de acordo com o custo e recursos computacionais exigidos pelo cliente.

Os *templates* utilizados foram: *t2.nano*, *t2.medium*, *t2.micro*, *t2.large*, *p2.xlarge*, *m4.large*, *m4.2xlarge* com as configurações disponíveis em (Amazon, 2016).

Diagrama de Classes: A Figura 3.4 apresenta a classe criada para a realização da leitura de instâncias EC2, correspondendo ao modelo definido em JSON na Figura 3.2.

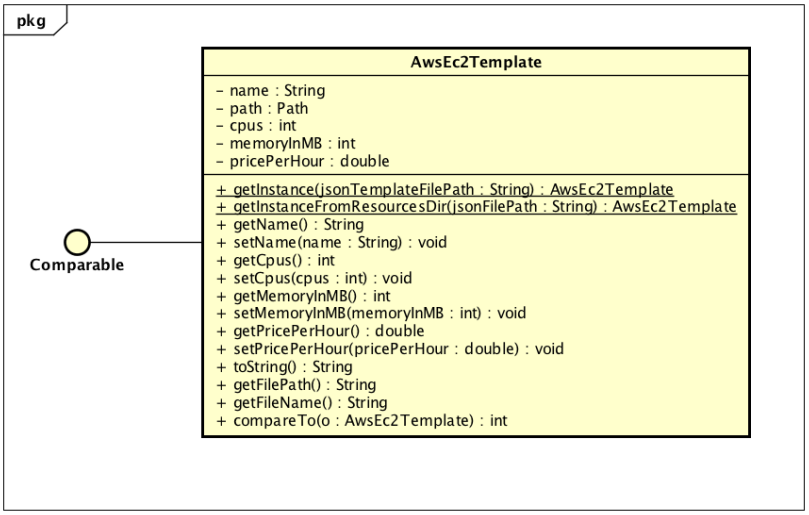

Figura 3.4: Diagrama de Classes - Módulo de Leitura de Configurações de Instâncias do *Amazon* EC2.

3.5.5 Automação da Criação de VMs Baseada em Requisitos de SLA

No contrato SLA pode ser incluída uma métrica de nível de tolerância a falta que o cliente deseja. Por exemplo, se o cliente requer um nível de tolerância a falta igual a 3, 3 VMs serão criadas em simultâneo. O algoritmo vai tentar criar as VMs com maior desempenho, respeitando o nível de tolerância a faltas acordado e o custo máximo por hora estipulado pelo cliente. Caso o custo não suporte a alocação das VMs com essas configurações, é obtida a VM mais barata de toda a lista.

Se o cliente determinar um nível de tolerância em que o custo imposto por ele não for suportado, o nível de tolerância apresentado no contrato será reduzido. Se não houver nenhum tipo de VM com custo menor ou igual ao estipulado pelo cliente, uma única instância da VM mais barata será criada e o custo do cliente será violado.

Serviços do tipo Amazon EC2, por exemplo, não automatizam a criação de máquinas virtuais para tolerar faltas, ficando sob a responsabilidade do cliente tais configurações. Existem serviços como o Amazon Lambda (Lambda, 2017) e Google Cloud Functions (Google, 2017), em que o utilizador não precisa de especificar VMs nem requisitos de infraestrutura. Toda a infraestrutura é automaticamente alocada utilizando Linux Containers. O cliente em vez de hospedar aplicações completas na nuvem, instala funções individuais desenvolvidas em linguagens como Java, Python e Java Script. Tais funções podem ser executadas, por exemplo, a partir de aplicações para dispositivos móveis e jogos. Tais serviços garantem uma extrema simplicidade de uso, porém, o cliente perde toda a flexibilidade de definir o sistema operativo, configurações, bibliotecas e aplicações a serem instaladas numa máquinas virtual. Desta forma, os serviços citados tem objetivos diferentes do nossos, pelo que estão fora do âmbito desta dissertação.

Algoritmo: O Algoritmo 3 representa o algoritmo utilizado para encontrar a quantidade e o tipo de instância adequado para o cliente, considerando o valor que o cliente

está disposto a pagar para utilizar os serviços na nuvem e também o nível de tolerância a faltas firmado.

Input: customer, templates
Output: The template selected
$Contract\ contract \leftarrow customer.getContract()$
$AwsEc2Template\ selected \leftarrow templates.getCheaperVmTemplate()$
for $template$ ***in*** $templates$ **do**
  if $template.getPricePerHour() \times contract.getMaxFaultToleranceLevel()$ **<=** $contract.getMaxPrice()$ ***and*** $template.getPricePerHour()$ **>** $selected.getPricePerHour()$ **then**
    $selected \leftarrow template$
end
return $selected$

Algoritmo 3: Automação da criação de VMs baseada em requisitos de SLA.

Em primeiro lugar, o algoritmo vai selecionar o *template* com o menor preço por hora de entre os *templates* disponíveis. Em seguida, vai verificar se o preço do *template* multiplicado pelo nível de tolerância firmado é menor ou igual ao valor imposto pelo cliente no contrato. Adicionalmente, se o preço desse *template* for maior que o preço do último *template* selecionado, retorna o *template* mais caro que possui um maior desempenho. Caso o valor do *template* multiplicado pelo nível de tolerância ultrapasse o firmado pelo cliente, a instância selecionada será a mais barata.

3.6 Conclusões

Neste capítulo, mostrou-se que podem ser lidos SLAs e, a partir da leitura, os serviços podem ser garantidos através de alocação de VMs e definição de custos.

Além da leitura de SLAs, foi também mostrado que é possível fazer a leitura de configurações de instâncias EC2 e a escolha da mesma a partir dos requisitos do cliente. O cliente define o nível de tolerância que deseja e, a partir do nível e do custo que ele estiver disposto a pagar, são alocadas as VMS. Desta forma, o cliente não precisa de criar manualmente e configurar as máquinas virtuais proporcionando assim uma automação da escolha, abstendo isso do cliente.

Foi também implementado um módulo de injeção de faltas com o objetivo maior de fornecer uma ferramenta útil para fazer e validar testes de faltas. Os eventos são gerados aleatoriamente seguindo a distribuição de Poisson, porém qualquer outra distribuição poderá ser utilizada. A justificativa para utilizar o Poisson foi pelo fato de ser bastante utilizado para modelar número de ocorrência de um evento, no nosso caso a ocorrência da falta. O Capítulo 4 apresenta os resultados obtidos a partir destas implementações.

Capítulo 4

Experiências e Resultados

4.1 Introdução

Neste capítulo são descritas as experiências realizadas e apresentados os resultados obtidos. A secção 4.2 aborda o *workload* real de um data center. Na secção 4.3 foram realizados testes para mapear aplicações para VMs utilizando o *workload* sintético e o *workload* real. Os algoritmos RR e WFD foram avaliados nestes dois cenários considerando a métrica de tempo de conclusão da aplicação. Na secção 4.4 foram realizados testes utilizando o injetor de faltas implementado, considerando a tolerância a faltas e disponibilidade. Na secção 4.5 são apresentados os testes referentes à automação da definição de máquinas virtuais baseada no custo para o cliente e no nível de tolerância a faltas que ele deseja e por fim é apresentada a conclusão do capítulo.

Cada uma das experiências realizadas neste capítulo foi executada 1200 vezes para obtenção de médias e intervalos de confiança (IC), para um nível de confiança de 95%. Foram também aplicadas técnicas de variantes antitéticas (Gentle, 2003) com o intuito de reduzir a variância dos resultados da simulação.

4.2 Data Center Workload

O *workload* real utilizado para a realização das simulações foi obtido a partir da página da Universidade de Jerusalém, disponível em (Feitelson, 2016; Feitelson et al., 2014). Cada *workload* representa um conjunto de *logs* de sistemas paralelos de grande escala a nível mundial.

O *Metacentrum* (University of Jerusalem, 2009) foi escolhido por conter vários meses de registros da rede nacional da República Checa, designado por *Metacentrum*. Ele representa dados de um grid composto por 14 *clusters* e diversas máquinas, conforme apresentado na Tabela 4.1. O registo de *workloads Metacentrum* foi fornecido pela *Czech National Grid Infrastructure Metacentrum*.

Tabela 4.1: Configurações do Ficheiro de Workloads Metacentrum.

Cluster	Processador	Nós	Total Cores
0	Itanium2 1.5GHz	8	8
1	Opteron 2.2GHz	16	16
2	Xeon 3.2GHz	10	10
3	Opteron 2.6GHz	5	80
4	AthlonMP 1.6GHz	16	32
5	Xeon 2.4GHz	32	64
6	Xeon 2.7GHz	36	148
7	Xeon 3.1GHz	35	70
8	Opteron 1.6GHz	10	20
9	Opteron 2.4GHz	3	6
10	Opteron 2.0GHz	23	92
11	Xeon 3.0GHz	19	152
12	Xeon 2.7GHz	8	64
13	Xeon 2.3GHz	11	44

4.3 Mapeamento de Aplicações para VMs

4.3.1 Algoritmo Implementado para Mapeamento de Aplicações para VMs

Conforme indicado na secção 3.5.3, o único algoritmo que existia no CloudSim Plus para mapeamento de aplicação para VM era o RR, que não é muito eficiente. Este algoritmo executa as aplicações nas VMs em uma fila circular, como exemplificado na Figura 4.1.

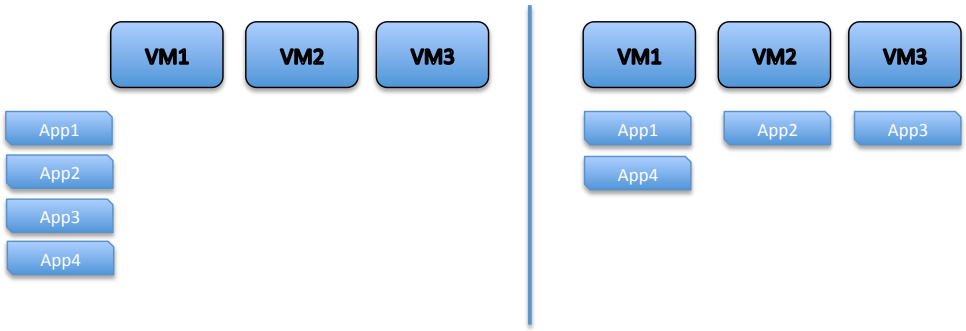

Figura 4.1: Mapeamento de Aplicações para VMs Utilizando o Algoritmo Round-Robin.

A Figura 4.1 apresenta 3 VMs e 4 aplicações com as seguintes configurações:

1. Quantidade de cores e MIPS por VM:
    - VM 1 = 4 cores, 1000 MIPS;
    - VM 2 = 8 cores, 2000 MIPS;
    - VM 3 = 4 cores, 1000 MIPS;
2. Tamanho das aplicações (em milhões de instruções):
    - App1 = 14000 MI;

- App2 = 14000 MI;
- App3 = 10000 MI;
- App4 = 20000 MI.

3. Cada aplicação App1, App2, App3 e App4 requer 2 cores da VM.

O RR mapeia a primeira aplicação para a primeira VM, a próxima para a segunda VM e assim sucessivamente. Como a terceira VM é o fim do círculo, o algoritmo volta para o início da lista de VMs, mapeando a quarta aplicação para a primeira VM. O processo ocorre até que todas as aplicações tenham sido mapeadas, sem qualquer parâmetro de escolha. Em alternativa, esta dissertação implementa o WFD, conforme descrito na Secção 3.5.3, que se preocupa com os requisitos da aplicação e a capacidade da VM. Com este algoritmo, o mapeamento das aplicações é feito conforme apresentado na Figura 4.2.

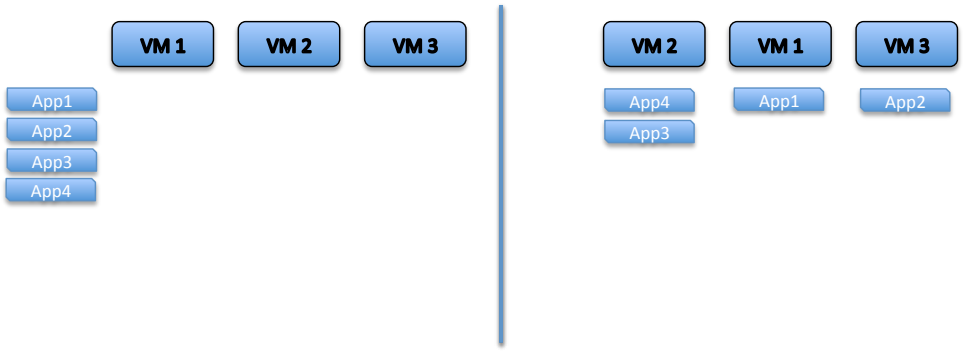

Figura 4.2: Mapeamento de Aplicações para VMs Utilizando o Algoritmo Worst Fit Decreasing.

Este algoritmo mapeia a aplicação com o maior tempo de conclusão esperado para a VM com maior capacidade de processamento, tornando o processo de mapeamento mais eficiente.

O algoritmo WFD ordena a lista de:

- VMs pela quantidade de cores disponíveis e capacidade de processamento: VM2, VM1, VM3. Ao enviar uma aplicação, ela será encaminhada para a VM com maior capacidade;
- Aplicações pelo tempo de conclusão esperado: App4, App1, App2, App3. O tempo de conclusão esperado para cada aplicação é dado pela divisão do seu tamanho pela capacidade de processamento em MIPs da VM.

Quando falamos em aplicações nos referimos a aplicações que não demandam resposta imediata ao cliente e que são executadas em segundo plano, como por exemplo processamento de imagens, computação científica, aplicações utilizando o framework mapreduce e etc.

4.3.2 Resultados de Simulação Utilizando o *Workload* Sintético

Foram realizadas experiências para testar a real eficiência do algoritmo implementado, em comparação com o RR. A Tabela 4.2 apresenta as configurações do cenário de simulação.

Tabela 4.2: Parâmetros e Respetivos Valores do Cenário Utilizando o *Workload* Sintético: Mapeamento de Aplicações para VMs.

Parâmetro	Valor
Número de Simulações	1200
Semente Base	1475098589732L
Número de VMs	30
Número de Aplicações	70
Possível Capacidade de Processamento de cada VM	1000 ou 2500 milhões de instruções/segundo (MIPS)
Possível Número de Cores para cada VM	2, 4
Número de Cores para cada Aplicação	2
Possível Tamanho para cada Aplicação (em MI)	10000, 14000, 20000, 40000
Aplicação da Técnica de Variantes Antitéticas	Sim

Para o cenário apresentado na Tabela 4.2, foram realizadas 7 experiências. O total de VMs é fixo, tendo as aplicações sido incrementadas de 10 em 10 para cada experiência. Na primeira experiência a quantidade de aplicações é igual a 10 e na última será igual a 70. A capacidade de processamento, o número de cores de cada VM e o tamanho das aplicações são atribuídos aleatoriamente, seguindo uma distribuição uniforme.

No gráfico da Figura 4.3 são apresentados os resultados das 7 experiências. A métrica de SLA levada em consideração nestas experiências foi o tempo de conclusão da aplicação (*task completion time*). O valor estabelecido para a métrica nestes testes foi de 30s. Utilizando o algoritmo WFD, as aplicações são melhor organizadas, fazendo com que o desempenho melhore, logo a percentagem de aplicações atendidas será maior.

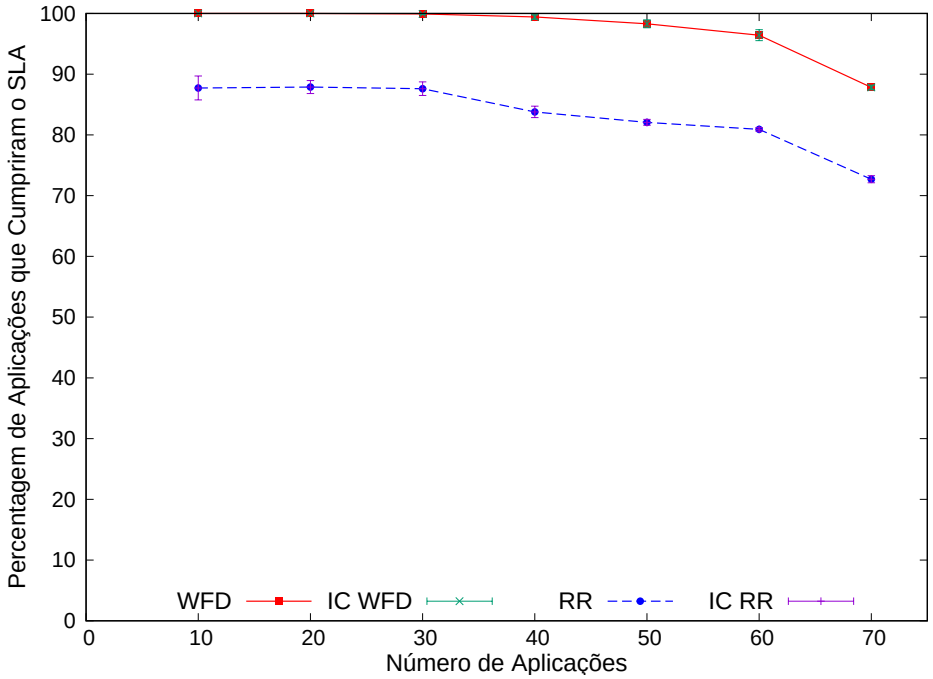

Figura 4.3: Percentagem de aplicações que cumpriram o SLA e respetivo intervalo de confiança (IC), em função do número de aplicações utilizando o *workload* sintético.

Observando o gráfico acima, é notória a eficácia do algoritmo WFD comparado com o RR. Em todas as experiências foram obtidas melhores respostas. Pode-se perceber que ao aumentar a quantidade de aplicações em cada *VM*, o desempenho dos algoritmos diminui. Considere a razão média entre o total de cores existente por VM ($vCores$) e a quantidade de cores requerida por aplicação ($aCores$), que chamamos de taxa $vCores/aCores$. Na primeira experiência, utilizando 10 aplicações, a taxa $vCores/aCores$ é maior que 1. Isto indica que, em média, há mais cores nas VMs do que o requerido pelas aplicações. Isso explica o bom desempenho dos dois algoritmos neste cenário sem sobrecarga. No último cenário, a taxa $vCores/aCores$ é igual a 0,4, indicando que há 0,4 $vCores$ da VM para um $aCores$ da aplicação, caracterizando um cenário de sobrecarga. Esta situação mostra o superior desempenho do algoritmo WFD em relação ao RR.

4.3.3 Resultados de Simulação Utilizando o Workload Real

A Tabela 4.3 apresenta as configurações do segundo cenário de simulação para mapeamento de aplicações para VMs utilizando o workload real. Os parâmetros utilizados nesta experiência foram maiores, assim como o tempo para finalização da aplicação, pois as aplicações e a quantidade de cores são maiores, logo toda a estrutura tem que ser ampliada para suportar a execução das mesmas.

Tabela 4.3: Parâmetros e Respetivos Valores do Cenário Utilizando o *Workload* Real: Mapeamento de Aplicações para VMs.

Parâmetro	Valor
Número de Simulações	1200
Semente Base	1475098589732L
Número de VMs	80
Número de Aplicações	440
Possível Capacidade de Processamento de cada VM	1000, 15000, 20000 ou 28000 MIPS
Possível Número de cores para cada VM	1, 2, 4, 8, 10
Número de Cores para cada Aplicação	Variável
Possível Tamanho para cada Aplicação (em MI)	Variável
Aplicação da Técnica de Variantes Antitéticas	Sim

Neste cenário, foram utilizadas 80 VMs e as aplicações foram incrementadas de 40 a 40, até chegar a 440 na última experiência. A capacidade de processamento e os cores de cada VM são atribuídos aleatoriamente, seguindo também uma distribuição uniforme. O número de cores para cada aplicação e o seu tamanho são variáveis. Esta informação foi retirada do *workload* METACENTRUM.

No gráfico da Figura 4.4 são apresentadas as percentagens de aplicações que foram atendidas de acordo com o *task completion time* estabelecido. Neste caso, o tempo máximo para finalização de cada aplicação foi fixado em 100s.

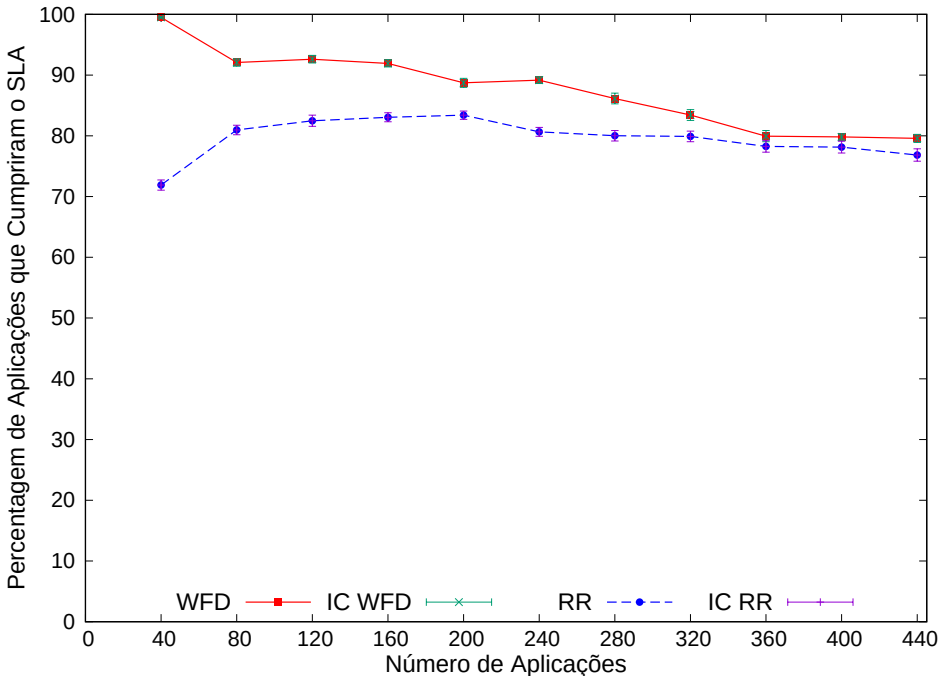

Figura 4.4: Percentagem de aplicações que cumpriram o SLA e respetivo intervalo de confiança (IC), em função do número de aplicações utilizando traces de data center real (METACENTRUM).

Começamos a experiência com um cenário sem sobrecarga, onde os algoritmos estavam livres para mapear as aplicações para VMs. O algoritmo WFD novamente teve uma

melhor resposta comparado com o RR, como pode ser observado no gráfico da Figura 4.4. O WFD escolhe de forma eficiente e prioriza a aplicação com maior estimativa de tempo de conclusão para a VM com maior capacidade, tornando assim a execução mais rápida. Já o algoritmo RR não é eficiente, explicando o seu desempenho inferior num cenário sem sobrecarga.

4.4 Injeção de Faltas

Como detalhado na Seção 3.5.2, as faltas injetadas são a nível do *host*, podendo falhar as **VMs** que estão sendo executados dentro do *host* ou não. As faltas são difíceis de evitar, apesar de as consequências poderem ser minimizadas. A tolerância a faltas é a capacidade do sistema continuar em funcionamento mesmo após a ocorrência de faltas (Soares e Pinho, 2016). Uma solução para conseguir um sistema tolerante a faltas é através da redundância. Contudo, ambientes altamente disponíveis trazem consigo grandes custos.

Para esta experiência, utilizou-se a redundância ativa $k+1$, onde k representa o número de componentes que podem faltar, com a adição de um backup independente. Nos testes, o total de *VMs* a serem criadas para cada cliente é definido de forma autónoma, de acordo com o nível de tolerância a faltas k definido no SLA. As *VMs* são executadas em simultâneo, entretanto o backup não participa ativamente no sistema até que as *VMs* falhem. Enquanto todas as k VMs não forem afetadas por faltas do *host*, a aplicação vai continuar a ser executada. O interrompimento do serviço só vai ocorrer quando todas as VMs falharem. Nesse momento, há um *downtime*, até que um clone da última VM destruída seja criado, tornando o serviço disponível novamente. A existência de k componentes implica que o sistema pode tolerar até k faltas.

A Tabela 4.4 abaixo apresenta as configurações do cenário de simulação:

Tabela 4.4: Parâmetros do Cenário: Injeção de Faltas.

Parâmetro	Valor
Número de Simulações	300
Semente Base	1475098589732L
Número de Clientes	6
Número de Hosts	50
Número de VMs	3
Número de Aplicações	6
Capacidade de Processamento de cada VM	1000 milhões de instruções/segundo (MIPS)
Número de cores para cada VM	2
Possível Tamanho para cada Aplicação (em MI)	1000000000, 1800000000, 2800000000
Aplicação da Técnica de Variantes Antitéticas	Sim

O número de *hosts* na primeira experiência é igual a 10 e em cada nova experiência

este valor é incrementado de 5 em 5 chegando até 50 *hosts*, como pode ser observado na Figura 4.5. A simulação é realizada para 6 clientes, cada um possuindo 3 *VMs* ($k = 3$) e 6 aplicações.

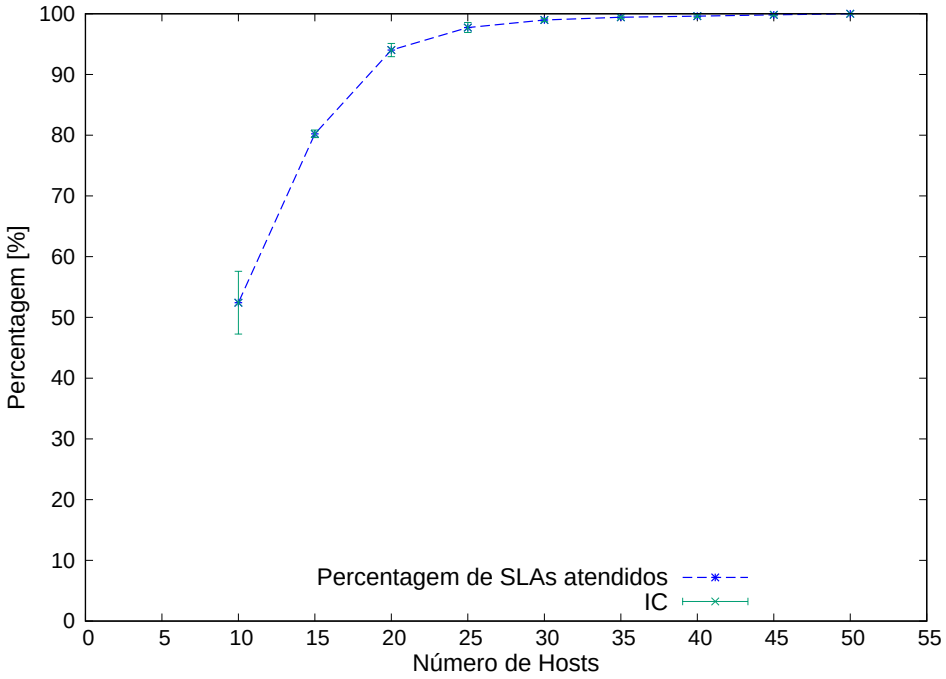

Figura 4.5: Percentagem de aplicações que cumpriram o SLA e respetivo intervalo de confiança (IC), em função do número de *hosts* utilizando workload sintético.

A Figura 4.5 apresenta a percentagem para os quais o SLA foi atendido, de acordo com a disponibilidade esperada, para diferentes quantidades de *hosts* disponíveis. No SLA, a disponibilidade mínima acordada foi de 99,9%, sendo que valores abaixo desse limite violam o acordo. Quanto mais *hosts* disponíveis, melhor será a capacidade de recuperação de faltas, logo a taxa de cumprimento irá aumentar. O pior cenário foi o primeiro com apenas 10 *hosts* disponíveis, onde um pouco mais da metade dos clientes tiveram recuperação de faltas e garantiram a disponibilidade contratada. Se todas as *VMs* do cliente faltarem, o backup é enviado para algum *host* disponível. A recuperação da falta vai depender da quantidade de *hosts* disponíveis e com cores em funcionamento suficientes para que o backup seja criado com sucesso.

Com o aumento do número de *hosts*, o cumprimento do SLA aumenta significativamente, chegando à taxa de cumprimento de 100%, como pode ser observado na Figura 4.5. A Figura 4.6 apresenta as médias da disponibilidade de todos os clientes da simulação.

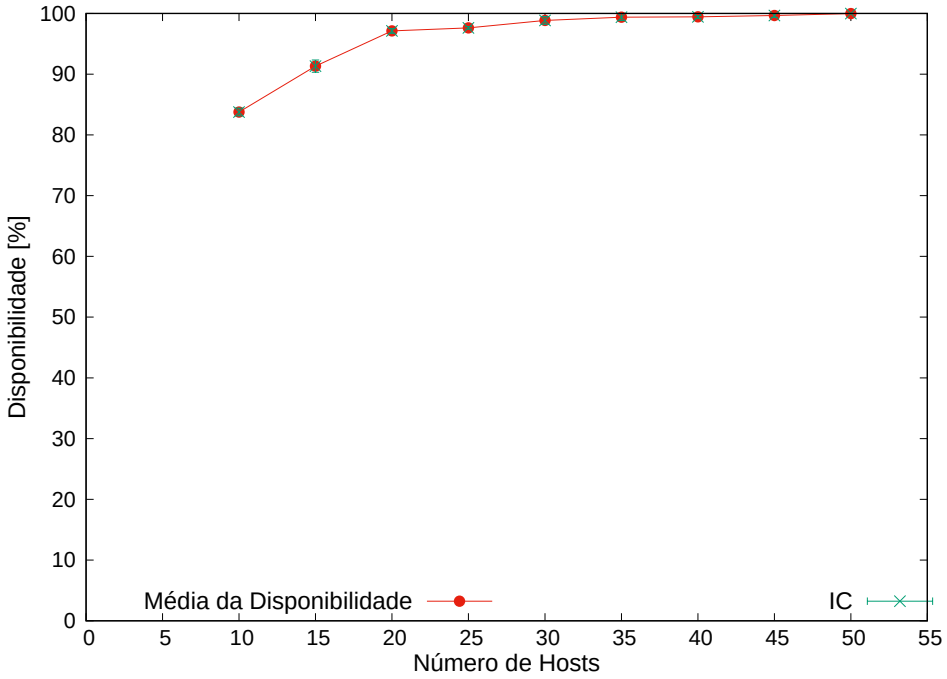

Figura 4.6: Média da disponibilidade da simulação e respetivo intervalo de confiança (IC) utilizando o workload sintético.

Durante a simulação, mediu-se também a taxa média de total de VMs por *host* $(VMs/host)$, onde se pode constatar que quanto menos *VMs* por *host*, melhor será o desempenho, como se pode observar na Tabela 4.5. Por exemplo, para atingir a disponibilidade de 99,9%, foi necessária uma taxa média de 0,4 *VMs*/*host*.

Tabela 4.5: Relação da Taxa de VMs/*host* e Disponibilidade.

Número de Hosts	Taxa de VMs/*host*	Disponibilidade
10	2,0	83,77
15	1,3	99,32
20	1,0	97,13
25	0,8	97,63
30	0,6	98,86
35	0,57	99,39
40	0,5	99,44
45	0,44	99,67
50	0,4	99,99

4.5 Custo para o Cliente

Cada cliente define os valores para as métricas de SLA que espera. Desta forma, cada cliente possui requisitos diferentes e específicos. Para as experiências de custo, considerou-se o custo que cada cliente está disposto a pagar por hora, para todas as VMs que possam ser instanciadas. A quantidade de máquinas virtuais para cada cliente é definida a partir do custo e do nível de tolerância a faltas definido no SLA. O cenário

definido para as experiências é apresentado na Tabela 4.6.

Tabela 4.6: Parâmetros do Cenário: Custo do Cliente.

Parâmetro	Valor
Número de Simulações	300
Semente Base	199975098589732L
Número de Clientes	8
Número de Hosts	30
Número de VMs	Variável
Número de Aplicações	2 por VM
Capacidade de Processamento de cada VM	1000 milhões de instruções/segundo (MIPS)
Número de cores para cada VM	Variável
Tamanho da Aplicação	1 bilião de MIs
Aplicação da Técnica de Variantes Antitéticas	Sim

Nesta experiência, a quantidade de clientes foi fixada em 8, no entanto qualquer outra quantidade poderia ter sido escolhida para esta simulação. Por outro lado, o número de *VMs* e cores é definido a partir do *template* AWS EC2 que for mais adequado para o custo por hora que o cliente atribuir ao SLA. A Tabela 4.7 apresenta os resultados obtidos.

Tabela 4.7: Custo e disponibilidade do serviço para o cliente, em que os valores assinalados a vermelho violaram a disponibilidade ou custo do contrato.

Clientes	Disponibilidade Esperada	Disponibilidade Atual	Custo/ hora Esperado $	Custo/ hora Atual $	k-tolerance level Esperado	k-tolerance level Atual	Template Ade-quado
C1	99,990 %	99,999 %	0,050	0,034	2	2	T2.micro
C2	99,900 %	99,990 %	0,020	0,0164	3	2	T2.nano
C3	99,000 %	98,230 %	0,0080	0,0082	2	1	T2.nano
C4	99,999 %	99,999 %	0,055	0,051	3	3	T2.micro
C5	99,990 %	99,990 %	0,045	0,0246	3	3	T2.nano
C6	99,990 %	99,220 %	0,005	0,0082	2	1	T2.nano
C7	99,990 %	99,999 %	0,035	0,0246	3	3	T2.nano
C8	99,900 %	99,750 %	0,0082	0,0082	2	1	T2.nano

Como mencionado, cada cliente possui as suas próprias especificações. Na tabela acima são apresentados resultados para os 8 clientes definidos (representados por C1 a C8), cada um deles possuindo custos e níveis de disponibilidade distintos. Foi escolhido para esta simulação 8 clientes, porém qualquer outra quantidade poderia ter sido escolhida.

Com esta experiência, o custo esperado pelos clientes foi atendido em 75% dos casos. Os *templates* foram definidos a partir do valor que cada cliente está disposto a pagar por hora. O custo é violado quando, para um custo esperado, não existir nenhum *template* dentro do valor estipulado. Desta forma, mesmo criando-se uma única instância para o template mais barato, poderá haver uma violação do custo definido no contrato. Se a percentagem de clientes para estes casos for elevada, isto pode servir como indicativo ao fornecedor de serviços que é necessário reajustar o preço das VMs ou fornecer VMs com configurações mais adequadas de modo que os clientes possam ter mais opções.

Em relação ao nível de tolerância, se o cliente definir um valor que não é possível ser atendido de acordo com o custo máximo estipulado, o nível será reduzido evitando a violação do custo.

A Tabela 4.7, mostra que apenas 62.5% dos clientes tiveram a disponibilidade atendida, especificamente os clientes C1, C2, C4, C5 e C7. O C8 não teve o custo violado, mas com o decréscimo do nível de tolerância, a disponibilidade não alcançou o valor esperado. A disponibilidade poderia ser alcançada se o cliente estivesse disposto a pagar por mais uma VM, aumentando assim o nível de redundância dos dados. Nestes casos, o fornecedor deve notificar o cliente informando que a disponibilidade foi reduzida porque o custo máximo estipulado não poderá ser atendido e que o cliente deverá ajustar o contrato em conformidade, caso o pretenda ajustar.

4.6 Conclusões

Com as experiências, foi comprovada a efetividade do algoritmo WFD em relação ao RR, garantindo melhor desempenho em todos os testes, tanto no primeiro cenário com *workload* sintético, quanto utilizando *workloads* reais. No entanto, pode-se observar que a disponibilidade depende tanto do fornecedor quanto do cliente. O fornecedor precisa de ter *hosts* suficientes para que a taxa de VMs/host não seja elevada, evitando prejudicar o nível de tolerância a faltas. Por outro lado, o cliente não pode limitar excessivamente o custo por hora que está disposto a pagar e ainda assim esperar ter alta disponibilidade, sendo necessário haver um equilíbrio entre estes dois fatores.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Principais Conclusões

A partir deste trabalho, os investigadores podem ter acesso a um conjunto de métricas disponíveis no simulador de computação em nuvem CloudSim Plus, tais como, a disponibilidade do serviço, o custo, o tempo de conclusão da aplicação, o MTTR e o MTBF. Além destas métricas implementadas, podem ser realizadas leituras de SLAs. O contrato foi implementado seguindo o mesmo formato definido pelos serviços do *Amazon CloudWacth*.

O módulo de injeção de faltas também é uma contribuição importante que foi introduzida no CloudSim Plus e é *open source*, assim como todas as implementações realizadas no simulador. A partir deste módulo, podem ser aplicadas e validadas faltas a nível de host. Podem também ser recuperadas faltas de VMs ocasionadas por faltas do *host*, através da execução paralela de VMs redundantes em diferentes *hosts*. No caso de múltiplas VMs de um mesmo cliente falharem, *snapshots* de tais VMs podem ser criados, recuperando o sistema da falta. Porém isso só é possível se o fornecedor tiver *hosts* disponíveis para alocar as VMs que possam vir a falhar e assim manter o serviço ativo.

Foi também observado por simulação que o algoritmo WFD implementado para mapeamento de aplicações para VMs apresenta um desempenho superior ao do algoritmo RR, aumentando o desempenho da métrica *task completion time*. Por fim, este trabalho apresenta uma proposta para automatizar a criação de máquinas virtuais para os clientes, de acordo com o custo e nível de tolerância estipulados em contrato. Desta forma, o fornecedor pode libertar o cliente de realizar a criação de VMs redundantes para balancear carga e tolerar faltas. Todo este processo pode ser automatizado pelo fornecedor, a partir das informações constantes no SLA. Atualmente, fornecedores como o *Amazon* deixam a cargo do cliente a criação de máquinas virtuais, o que pode levar ao desperdício de dinheiro, ociosidade de recursos computacionais e um nível de tolerância a faltas e balanceamento de carga diferente do que o cliente espera.

5.2 Direções para Trabalhos Futuros

Como trabalhos futuros, sugere-se:

- Monitorizar métricas de SLA em tempo de execução com penalidades se ocorrer violações;
- Melhorar o injetor de faltas adicionando outros tipos de faltas;

- Realizar experiências em infraestruturas reais de computação em nuvem para validação experimental dos resultados obtidos;
- Realizar experiências de cumprimento de métricas com migração de máquinas virtuais.

Referências

Amazon (2010). Amazon CloudWatch Command Line Tool. Disponivel em: http://aws.amazon.com/developertools/2534. Data do último acesso: 02/03/2017. 27

Amazon (2016). Amazon EC2 Pricing. Disponível em: https://aws.amazon.com/ec2/pricing/. Data do último acesso: 15/03/2017. 25, 33, 34

Ardagna, D., Casale, G., Ciavotta, M., Pérez, J. F., e Wang, W. (2014). Quality-of-Service in Cloud Computing: Modeling Techniques and their Applications. *Journal of Internet Services and Applications*, 5(1):11. 3

Aslanpour, M. S. e Dashti, S. E. (2016). SLA-Aware Resource Allocation for Application Service Providers in the Cloud. *In Proceedings of 2016 2nd International Conference on Web Research (ICWR 2016)*, pages 31–42. 21

Bardsiri, A. K. e Hashemi, S. M. (2014). QoS Metrics for Cloud Computing Services Evaluation. *International Journal of Intelligent Systems and Applications*, 6(12):27–33. 14

Barroso, L. A. e Hölzle, U. (2007). The Case for Energy-proportional Computing. *Computer*, 40(12):33–37. 8

Begum, S. e Prashanth, C. (2013). Review of Load Balancing in Cloud Computing. *IJCSI International Journal of Computer Science Issues*, 10(1):343–352. 14

Bianco, P., Lewis, G. a., e Merson, P. (2008). Service Level Agreements in Service-Oriented Architecture Environments. Technical Report September, Carnegie Mellon University. xv, 10, 11, 12, 17

Calheiros, R. N., Ranjan, R., De Rose, C. A. F., e Buyya, R. (2009). CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *arXiv preprint arXiv:0903.2525*, page 9. 21

Calheiros, R. N., Ranjany, R., e Buyya, R. (2011). Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments. *In Proceedings of the International Conference on Parallel Processing*, pages 295–304. xv, 8, 20, 21

Catela, M. F., Pedron, C. D., e Macedo, B. A. (2014). Service Level Agreement em Cloud Computing: Um Estudo de Caso Em Uma Empresa Portuguesa. *Universitas: Gestão e TI*, 4(1). 14

Chen, C.-C., Yuan, L., Greenberg, A., Chuah, C.-N., e Mohapatra, P. (2014). Routing-as-a-Service (RaaS): A Framework for Tenant-Directed Route Control in Data Center. *IEEE/ACM Transactions on Networking*, 22(5):1401–1414. 2

Cisco System (2016). Tráfego Cloud vai Multiplicar-se por Quatro Entre 2014 e 2019. Disponível em: http://www.cisco.com/c/pt_pt/about/press/news-archive-2015/20151028.html. Data do último acesso: 30/05/2016. 1

Cisco Systems (2006). Understanding Jitter in Packet Voice Networks (Cisco IOS Platforms) Jitter in Packet Voice Networks. Disponível em: http://www.cisco.com/c/en/us/support/docs/voice/voice-quality/18902-jitter-packet-voice.pdf. Data do último acesso: 04/03/2017. 14

Di, S. e Cappello, F. (2015). GloudSim: Google Trace Based Cloud Simulator with Virtual Machines. *Software - Practice and Experience*, 45(11):1571–1590. 14, 19

Feitelson, D. G. (2016). Parallel Workloads Archive. Disponível em: http://www.cs.huji.ac.il/labs/parallel/workload/. Data do último acesso: 05/07/2016. 37

Feitelson, D. G., Tsafrir, D., e Krakov, D. (2014). Experience with using the Parallel Workloads Archive. *Journal of Parallel and Distributed Computing*. 37

Fernandes, D. A. B., Soares, L. F. B., Gomes, J. V., Freire, M. M., e Inácio, P. R. M. (2014). Security Issues in Cloud Environments: A Survey. *International Journal of Information Security*, 13(2):113–170. 2

Gentle, J. E. (2003). *Random Number Generation and Monte Carlo Methods*. Springer-Verlag New York. 37

Google (2017). Cloud Functions. Disponível em: https://cloud.google.com/functions/. Data do último acesso: 02/06/2017. 35

Gupta, A., Kalé, L. V., Milojicic, D., Faraboschi, P., e Balle, S. M. (2013). HPC-aware VM Placement in Infrastructure Clouds. *In Proceedings of the IEEE International Conference on Cloud Engineering (IC2E 2013)*, pages 11–20. 3

Gutmann, P. (2007). The Commercial Malware Industry. Disponível em: https://www.defcon.org/images/defcon-15/dc15-presentations/dc-15-gutmann.pdf. Data do último acesso: 01/06/2017. In *Defcon 15*. 2

IBM (2017). IBM Research. Disponível em: http://research.ibm.com/. Data do último acesso: 22/12/2016. 15

ITI v3 Foundation (2009). ITIL® v3 Foundation Study Guide. Disponível em: http://www.inf.unideb.hu/~fazekasg/oktatas/ITIL_V3_Study_Guide.pdf. Data do último acesso: 14/10/2016. 3

José Mauricio Santos Pinheiro (2008). Métricas de Qualidade de Serviço em Redes de Computadores. Disponível em: http://www.projetoderedes.com.br/artigos/artigo_metricas_qos_em_redes.php. Data do último acesso: 13/01/2017. 14

Keller, A. e Ludwig, H. (2003). The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81. 15

Khajeh-Hosseini, A., Greenwood, D., Smith, J., e Sommerville, I. (2012). The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in The Enterprise. *Software - Practice and Experience*, 43(4):447–465. 23

Kim, K. H., Beloglazov, A., e Buyya, R. (2011). Power-aware Provisioning of Virtual Machines for Real-time Cloud Services. *Concurrency Computation Practice and Experience*, 23(13):1491–1505. 8

Lamanna, D. D., Skene, J., e Emmerich, W. (2003). SLAng: A Language for Defining Service Level Agreements. *In Proceedings of the Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, 2003 (FTDCS 2003)*, 34069:100–106. 16

Lambda (2017). AWS Lambda. Disponível em: https://aws.amazon.com/lambda/. Data do último acesso: 02/06/2017. 35

Lo, N. W., Fan, P. C., e Wu, T. C. (2014). An Efficient Virtual Machine Provisioning Mechanism for Cloud Data Center. *In Proceedings of 2014 IEEE Workshop on Electronics, Computer and Applications (IWECA 2014)*, pages 703–706. 8

Ludwig, H., Keller, A., Dan, A., King, R. P., e Franck, R. (2002). Web Service Level Agreement (WSLA) Language Specification. *IBM Corporation*, pages 1–110. 15

Malik, A., Sharma, A., e Saroha, V. (2013). Greedy Algorithm. *In Proceedings of International Journal of Scientific and Research Publications*, 3(8). 9

Mell, P. M. e Grance, T. (2011). The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology. Disponivél em: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Data do último acesso: 02/02/2017. 1, 2

Morse, E. L. (2017). Metrics and Measures. Disponível em: http://zing.ncsl.nist.gov/nist-icv/documents/section5.htm. Data do último acesso: 22/02/2017. 26

Nita, M. C., Pop, F., Mocanu, M., e Cristea, V. (2014). FIM-SIM: Fault Injection Module for CloudSim Based on Statistical Distributions. *Journal of Telecommunications and Information Technology*, 2014(4):14–23. 22

Nitika, Shaveta, G. R. (2012). Comparative Analysis of Load Balancing Algorithms in Cloud Computing. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1(3):120–124. 9

OpenNebula (2008). OpenNebula Documentation - Monitoring Overview. Disponível em: http://docs.opennebula.org/4.12/administration/monitoring/mon.html. Data do último acesso: 24/01/2017. 17

OpServices (2015). Entenda o Que é SLA e Qual a sua Importância. Disponível em: http://www.opservices.com.br/o-que-e-sla-e-qual-a-sua-importancia/. Data do último acesso: 21/12/2016. 11

Opservices (2015). MTTR e MTBF, o que são e quais suas diferenças? Disponível em: https://www.opservices.com.br/mttr-e-mtbf/. Ultima data de acesso: 15/06/2017. 26

OpServices (2015). Quais as Vantagens de Um Service Level Agreement? Disponível em: http://www.opservices.com.br/quais-as-vantagens-de-um-service-level-agreement/. Data do último acesso: 21/12/2016. 11

ORACLE (2010). Sun Java System Application Server Enterprise Edition 8.2 Deployment Planning Guide. Disponível em: https://docs.oracle.com/cd/E19900-01/819-4741/abfch/index.html. Data do último acesso: 22/07/2017. 14

Rak, M., Cuomo, A., e Villano, U. (2013). A Proposal of a Simulation-Based Approach for Service Level Agreement in Cloud. *In Proceedings of Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*, pages 1235–1240. xv, 18, 19

Riggs, C. (2016). Windows as a Service. Disponível em: https://sec.ch9.ms/slides/winHEC/3_01_WaaSServicing.pdf. Data do último acesso: 10/06/2017. 2

Righi, R., Pellissari, F., e Westphall, C. (2004). *Sec-SLA: Especificação e Validação de Métricas para Acordos de Níveis de Serviço Orientados à Segurança*. PhD thesis, Universidade Federal de Santa Catarina, Florianópolis, SC. Disponível em: `http://en.scientificcommons.org/6843995`. 14

Science, C. e Studies, M. (2014). A Comparative Study of Static and Dynamic Load Balancing Algorithms. *International Journal of Advance Research in Computer Science and Management Studies*, pages 386–392. xv, 8

Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C., Inácio, P. R. M., e Freire, M. M. (2016). CloudSim Plus: A Modern, Full-featured, Highly Extensible and Easier-to-use Java 8 Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. Disponível em: http://cloudsimplus.org. Data do último acesso: 10/12/2016. 5, 23

Silva Filho, M. C., Oliveira, R. L., Monteiro, C. C., Inácio, P. R. M., e Freire, M. M. (2017). CloudSim Plus: a Cloud Computing Simulation Framework Pursuing Software Engineering Principles for Improved Modularity, Extensibility and Correctness. In *Proceedings of IFIP/IEEE International Symposium on Integrated Network Management*, pages 400–406, Lisbon, Portugal, 8-12 May. 23

Singh, H. e Gangwar, R. C. (2014). Comparative Study of Load Balancing Algorithms in Cloud Environment. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(10). 9

Soares, G. T. e Pinho, P. H. V. (2016). *Sistema de Arquivos Distribuído Tolerante a Falhas Maliciosas: Uma Prova de Conceito*. Monografia do curso de computação - licenciatura, Universidade de Brasília. 43

Sturm, R., Morris, W., e Jander, M. (2000). *Foundations of Service Level Management*. Sams Publishing. 17, 18

TechNet - Microsoft (2009). Interpreting CPU Utilization for Performance Analysis. Disponível em: https://blogs.technet.microsoft.com/winserverperformance/2009/08/06/interpreting-cpu-utilization-for-performance-analysis/. Data do último acesso: 21/12/2016. 14

University of Jerusalem (2009). The MetaCentrum Log. disponível em: http://www.cs.huji.ac.il/labs/parallel/workload/l_metacentrum/index.html. Data do último acesso: 05/07/2016. 37

Uriarte, R. B., Tiezzi, F., e De Nicola, R. (2014). SLAC: A Formal Service-Level-Agreement Language for Cloud Computing. *In Proceedings of 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC 2014)*, pages 419–426. 14, 17

Vu, Q. H., Pham, T.-V., Truong, H.-L., Dustdar, S., e Asal, R. (2012). DEMODS: A Description Model for Data-as-a-Service. In *Proceedings of 2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 605–612. 2

Wang, F., Lim, A., e Chen, H. (2004). Flexible Demand Assignment Problem. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*. IOS Press. 32

Wang, Y., Chen, S., e Pedram, M. (2013). Service Level Agreement-based Joint Application Environment Assignment and Resource Allocation in Cloud Computing Systems. *In Proceedings of IEEE Green Technologies Conference*, pages 167–174. 3

Weibull (2017). Availability and the Different Ways to Calculate It. Disponível em: http://www.weibull.com/hotwire/issue79/relbasics79.htm. Data último acesso: 22/02/2017. 14, 26

Zaouch, A. (2015). Load Balancing for Improved Quality of Service in the Cloud. *International Journal of Advanced Computer Science and Applications*, 6(7):184–189. 9

Zeginis, C. e Plexousakis, D. (2010). Monitoring the QoS of Web Services using SLAs Technical Report. *Technical Report 404*, pages 1–17. 9, 10, 13

Zhang, Z. e Zhang, X. (2010). A Load Balancing Mechanism Based on Ant Colony and Complex Network Theory in Open Cloud Computing Federation. *In Proceedings of 2010 2nd International Conference on Industrial Mechatronics and Automation (ICIMA 2010)*, 2:240–243. 7