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

Modelo Multi-estratégico de Tolerância a Falhas para Ambiente de Nuvem Federada

Jefferson Chaves Gomes

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

Orientadora
Profa. Dra. Aletéia Patrícia Favacho de Araújo

Brasília
2018

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

Modelo Multi-estratégico de Tolerância a Falhas para Ambiente de Nuvem Federada

Jefferson Chaves Gomes

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

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

Profa. Dra. Célia Ghedini Ralha
Universidade de Brasília

Prof. Dr. José Viterbo Filho
Universidade Federal Fluminense

Prof. Dr. Bruno L. Macchiavello Espinoza
Coordenador do Programa de Pós-graduação em Informática

Brasília, 15 de Agosto de 2018

Dedicatória

Dedico este trabalho com todas as minhas forças, para minhas Filhas, Ana Laura e Ana Lis. E, dedico-o especialmente para minha Esposa Rosilma, que sempre esteve a me esperar durante as várias madrugadas que cheguei em casa após os estudos. Ela, que mesmo grávida, enjoada e indisposta, esteve sempre disposta a se levantar e esquentar minha comida para que eu pudesse dormir o quanto antes, visto que eu tinha que trabalhar logo cedo. Ela, que com certeza foi a pessoa que mais sentiu minha falta, mas sempre compreendeu o motivo de minha ausência e sempre me incentivou a continuar, sempre me deu forças e sempre me ajudou a seguir em frente.

Agradecimentos

Agradeço primeiramente a Deus, por me dar forças, por me tirar o sono e por aliviar o meu cansaço em muitos os momentos de sobrecarga que passei para o completo desenvolvimento deste trabalho. Agradeço a Deus por confortar e cuidar da minha Esposa e Filhas quando não pude estar presente, devido as várias madrugadas que passei me dedicando a este trabalho. Agradeço a Deus pela minha família, que soube compreender e aceitar minha ausência em momentos em que estamos acostumados a passar juntos.

Agradeço a minha Mãe Dalete, que mesmo se preocupando o com tempo que eu estive fora nas madrugadas, evitava me ligar com o objetivo de não atrapalhar a minha concentração. Ela, que chegou a sentir até mesmo o meu cansaço e a chorar por ele. Ela, que esteve sempre me apoiando e dando forças para que eu seguisse em frente dizendo que eu conseguiria.

Agradeço a minha orientadora Aletéia, que me aceitou como orientando e com muita paciência me conduziu até este ponto. Agradeço aos meus amigos, que me deram incentivo e me ajudaram a aliviar a pressão com bons momentos de descontração. Não posso deixar de agradecer meu amigo Breno Moura, que muito me ajudou com o entendimento da plataforma BioNimbuZ e é claro, ao meu amigo Felipe Souza, que esteve ao meu lado durante todo o curso de mestrado e durante toda a trajetória deste trabalho, me ajudando e me incentivando.

Não penso que eu mesmo já o tenha alcançado, mas uma coisa faço: esquecendo-me das coisas que ficaram para trás e avançando para as que estão adiante, prossigo para o alvo... (Filipenses 3:13-14)

Resumo

A computação em nuvem é tipicamente caracterizada por hospedar sistemas de larga escala, que apresentam comportamento dinâmico e escalável devido à sua grande oscilação de carga de trabalho. Porém, a influência dessas características afetam diretamente a dependabilidade e a segurança de sistemas implantados em ambientes de nuvem. Atualmente, com o objetivo de obter maior poder computacional ou maior capacidade de armazenamento, muitos usuários estão migrando do modelo de nuvem única para o modelo de nuvem federada, pois neste modelo há a combinação de mais de um provedor de nuvem, o que possibilita a utilização diversificada para a infraestrutura, para a arquitetura ou para a localização geográfica. Em geral, o modelo de federação de nuvens é utilizado para estender ou potencializar a capacidade de sistemas em nuvem computacional. Todavia, o fornecimento de serviço confiável dentro de uma federação de nuvens, que depende da orquestração de várias nuvens para fornecer os serviços, continua a ser um problema não resolvido. Diante deste contexto, a proposta deste trabalho é desenvolver um modelo de tolerância a falhas para plataformas de nuvens federadas, que possibilite a detecção e a recuperação proativa e reativa de falhas de forma parametrizável. A parametrização proposta visa flexibilizar o nível desejado de tolerância a falhas para os usuários finais, possibilitando que os usuários otimizem os custos necessários para tal, de acordo com as necessidades de seus modelos de negócio. Como análise experimental, foi utilizada a plataforma de federação de nuvem BioNimbuZ, e os resultados mostram que com o modelo de tolerância a falhas proposto, foi possível garantir para a plataforma BioNimbuZ, o mínimo de 95,23% de disponibilidade e 95,64% de confiabilidade, mesmo com a injeção periódica de falhas a cada 5 minutos.

Palavras-chave: Modelo de Tolerância a falhas, nuvens federadas, *workflows*, dependabilidade, BioNimbuZ, multi-estratégia, repetição, reenvio de tarefas, replicação, rejuvenescimento de software

Abstract

Cloud computing is typically characterized by hosting large-scale systems which exhibit dynamic and scalable behavior due to large variation in their workload. These characteristics directly affect the dependability and security of systems deployed in cloud environments. Currently, in order to obtain greater computing power or storage capacity, many users are migrating from the single cloud model to the federated cloud model because, in the latter, there is a combination of more than one cloud provider which increases robustness allowing diversified use for infrastructure, architecture, or geographic location. In general, the cloud federation model is used to extend or enhance the capacity of cloud computing systems. However, providing a dependable service within a cloud federation, which depends on orchestrating multiple clouds to provide services, remains an unresolved issue. In this context, the purpose of this research is to implement a fault-tolerant model for federated cloud platforms which enables proactive and reactive fault detection and recovery in a parameterizable manner. The proposed parameterization aims to make the especification of desired level of fault tolerance easier for end users, allowing them to optimize the costs required to tolerate faults in their business model. As a experimental analysis, the BioNimbuZ cloud federation platform was used, and the results show that with the proposed fault tolerance model, was possible to guarantee for the BioNimbuZ platform a minimum of 95.23% to availability and 95.64% to reliability, even with a periodic injection of faults at every 5 minutes.

Keywords: Fault tolerance, federated clouds, workflows, dependability, BioNimbuZ, multi-strategy, retry, task resubmission, replication, software rejuvenation

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API Applica-tion Programming Interface.

BFTCloud Byzantine Fault Tolerant Cloud.

DC Data Center.

DPM Defects Per Million.

FT-FC Fault-Tolerance in Federated Cloud.

FTM Fault Tolerance Manager.

HTTP Hypertext Transfer Protocol.

IaaS Infrastructure as a Service.

IP Internet Protocol.

JSON JavaScript Object Notation.

LLFT Low Latency Fault Tolerance.

MTBF Mean Time Between Failures.

MTTF Mean Time To Failure.

MTTR Mean Time To Repair.

NIST (National Institute of Standards and Technology.

P2P Peer-to-Peer.

PaaS Platform as a Service.

PID Process Identifier.

SaaS Software as a Service.

SLA Service-Level Agreement.

TC Task Coordinator.

TE Task Executor.

VFT Virtualized Fault Tolerance.

Capítulo 1

Introdução

A computação em nuvem é um modelo extensamente difundido, capaz de prover sob demanda vários tipos de recursos computacionais como, por exemplo, recursos virtuais, rede, armazenamento de dados e aplicações. Esses recursos são disponibilizados como serviço e podem ser rapidamente provisionados e liberados, com um esforço mínimo de gerenciamento ou interação com o provedor do serviço [10]. Em outras palavras, é um modelo computacional em que os recursos e os serviços são fornecidos por meio da Internet, utilizando diferentes modelos e camadas de abstrações [11].

Os provedores de nuvem computacional passam a ideia, aos usuários finais, que os recursos computacionais são virtualmente ilimitados, mas na prática, para atender a grande demanda de utilização é necessário aplicar limitações de utilização a cada usuário [12, 13, 14]. Para superar estes limites, usuários podem construir aplicações que utilizem recursos combinados de diversos provedores de nuvem, formando assim, uma federação de nuvem computacional [15, 16].

Plataformas de nuvens federadas buscam aumentar o conjunto de recursos, a flexibilidade, a disponibilidade e a confiabilidade, além de reduzir o custo necessário à execução de tarefas, isto de forma dinâmica e transparente [15]. Plataformas de nuvens federada são conectadas a cada provedor participante da federação com o objetivo de provisionar recursos sob demanda, atendendo as requisições de processamento de dados. Desta forma, o ambiente de execução, mesmo que em diferentes provedores, passa a ser transparente aos solicitantes [17].

A popularização do uso de aplicações distribuídas em ambientes de nuvem vem aumentando consideravelmente, isto faz com que a preocupação com a alta disponibilidade dos serviços seja cada vez mais prioritária, pois em ambientes distribuídos as falhas são comuns [18]. Além disso, em ambiente de nuvem computacional é certo afirmar que falhas ocorrerão em algum momento durante a execução da aplicação [8]. Aplicações de alto risco, as quais devem garantir disponibilidade máxima aos seus usuários, são em

sua maioria produzidas de forma distribuída. Em aplicações distribuídas que executam *workflows* científicos, o tempo de execução pode levar dias ou até mesmo semanas para concluir suas tarefas, e aplicações deste tipo requerem estratégias de monitoramento e tolerância a falhas para reduzir as chances de perder o trabalho executado (*i.e.*, a execução da aplicação), antes do mesmo ser concluído [19].

Assim, em sistemas distribuídos as possíveis falhas estão frequentemente em debate no que se refere à infraestrutura e aos ambientes em nuvens. Nesses debates procura-se definir se a responsabilidade por prover um mecanismo de tolerância a falhas é do provedor do serviço de nuvem, ou se é dos proprietários de aplicações distribuídas. Contudo, provedores precisam manter uma taxa de falhas baixa, a fim de conquistar a confiança e aumentar o número de clientes.

Todavia, um investimento substancial é necessário para manter um eficiente mecanismo de tolerância a falhas [20]. Em particular, a disponibilidade de uma determinada aplicação possui maior relevância para quem a produz, logo, os proprietários são os que tem maior interesse em implementar o mecanismo de tolerância a falhas em suas aplicações. Porém, existem falhas cuja recuperação não depende do mecanismo de tolerância a falhas em nível de aplicação, mas sim em nível de infraestrutura. Logo, tolerar falhas é dever de ambos os envolvidos, provedores e consumidores.

Dados os fatos, surge a preocupação em produzir um modelo tolerante a falhas que possibilite ou favoreça o nível de dependabilidade para plataformas que utilizem ambientes de nuvem computacional, tanto do ponto de vista da aplicação, quanto do recurso computacional que a executa, pois se um recurso falha, outro precisa estar disponível. Além disso, se a aplicação falha de maneira a parar sua execução, outra deve ser iniciada. Entretanto, nem sempre a falha está no recurso ou na aplicação, pois em uma execução distribuída, o recurso/aplicação dependem de comunicação com outros servidores para concluírem suas tarefas, em todo caso, a causa da falha deve ser detectada e corrigida. Para isto, há a necessidade de um modelo de tolerância a falhas capaz de detectar e recuperar sistemas distribuídos destes tipos de falhas. E por se tratar de ambientes nos quais paga-se pelo que for utilizado, é importante que consumidores possam configurar o nível de tolerância a falhas desejado, visando a otimização de custos de acordo com o seu modelo de negócio.

Diante do contexto apresentado, este trabalho propõe um modelo de tolerância a falhas multi-estratégico, que permita aos usuários adotar uma estratégia de tolerância a falhas conforme a sua necessidade, e que opere de maneira integrável a sistemas, sejam eles distribuídos ou não. Ou seja, um modelo implementado como biblioteca que deve ser integrado facilmente ao sistema do usuário, isto de maneira independente no modelo de nuvem utilizado, a fim de favorecer a dependabilidade. Como avaliação experimental,

será utilizado o sistema BioNimbuZ [21] que é uma plataforma de federação de nuvens computacionais híbridas para a execução de *workflows* de Bioinformática.

O modelo de tolerância a falhas proposto foi construído com base em pesquisas exploratórias, na qual foi efetuado um levantamento bibliográfico respaldado, principalmente, por livros e artigos científicos.

1.1 Motivação

Atualmente, o paradigma de federação de nuvem computacional vem ganhando considerável força, como sendo uma estratégia para a execução de aplicações distribuídas devido a sua maior capacidade de provisionamento e de escalabilidade de recursos sob demanda. Para atender as necessidades deste tipo de aplicação, a utilização de federações de nuvens vem sendo explorada como meio de estender a abstração do uso de recursos virtualmente ilimitados [16, 22, 23].

Embora o conceito de nuvens federadas aludam ainda mais a possibilidade de provisionamento de recursos virtualmente infinitos, até mesmo os mais robustos provedores de nuvem computacional tem problemas para atender a demanda, que vem crescendo com o passar do tempo, pois os provedores podem ter problemas de interrupção no fornecimento de serviços hospedados em seus ambientes [24, 25, 26, 27, 28, 29, 30].

Alguns dos fatores comuns às interrupções de serviços oferecidos por provedores de nuvens são problemas de localidade, que podem afetar o tempo de resposta; problemas de rede (*i.e.*, comunicação); e problemas técnicos que envolvam falhas de software e de hardware, dentre outros [19]. O provedor *Amazon Web Services*[1], por exemplo, declarou a interrupção de serviços implantados em sua infraestrutura devido a problemas de comunicação, queda de energia e de atualização de configurações de rede, que foi executada de forma incorreta, causando assim, impacto à disponibilidade de seus serviços [24, 25, 26]. Já o provedor *Google Cloud Platform*[2], teve problemas com comunicação de rede, perda de pacotes e atualizações aplicadas à configurações de seu balanceador de carga, que causaram problemas para a conectividade de instâncias de máquinas virtuais [27, 28, 29]. No caso do provedor *Microsoft Azure*[3], os impactos à disponibilidade ocorreram devido a falhas em seu sistema rotativo de fornecimento de energia ininterrupto, que falhou de maneira a tornar seus *datacenters* indisponíveis, além disto, registrou-se que seus serviços de máquinas virtuais apresentaram grande latência ou baixo desempenho devido a problemas em serviços de *back end* utilizados em sua infraestrutura [30].

[1] Amazon Web Services, `https://aws.amazon.com`
[2] Google Cloud Platform, `https://cloud.google.com`
[3] Microsoft Azure, `https://azure.microsoft.com`

1.2 Problema

Diante do exposto, é correto afirmar que falhas são inevitáveis, pois projetos de software e de hardware possuem alta complexidade e sofrem diante da fragilidade humana em lidar com grande volume de detalhes, bem como a deficiência de sua especificação [4]. Assim, conforme análise da literatura, existem ainda muitos desafios a serem enfrentados para aumentar a dependabilidade e a segurança de sistemas em ambientes de nuvens. Alguns dos principais desafios são elencados a seguir [1, 8, 31, 32]:

- A heterogeneidade da nuvem é o maior obstáculo para a localização de falhas, logo, é necessário implementar técnicas eficientes para localizar e reparar falhas neste tipo de ambiente;
- O processamento de tarefas em recursos remotos favorece a ocorrência de erros;
- As falhas que ocorrem nos *datacenters* não estão sob o escopo da organização do usuário, necessitando da implementação de uma técnica automática de tolerância a falhas para computação de aplicativos em ambiente de nuvens;
- É difícil interpretar a alteração do estado do sistema, devido ao ambiente de nuvem ser dinamicamente escalável;
- Devido aos diferentes níveis de permissão de acesso, dadas aos usuários finais à infraestrutura dos diversos provedores de nuvem, o projeto de uma solução ideal de tolerância a falhas se torna ainda mais complexo.

Para fazer frente aos desafios citados e amenizar o impacto causado pelos mesmos à dependabilidade, há a necessidade de um modelo de tolerância a falhas capaz de permitir a correta operação de sistemas distribuídos mesmo sob a presença de falhas. Por se tratar de ambientes que possuem custo associado à utilização de recursos computacionais, é importante que consumidores possam configurar ou parametrizar o nível de tolerância a falhas desejado com o objetivo de definir estratégias que visem a otimização de custos de acordo com o seu modelo de negócio.

1.3 Objetivos

O objetivo principal deste trabalho é propor um modelo multi-estratégico de tolerância a falhas para sistemas distribuídos, que possibilite a detecção e a recuperação de falhas de forma dinâmica, automática e parametrizável. A parametrização proposta visa flexibilizar para os usuários finais o nível de tolerância a falhas desejado, possibilitando que o mesmo

otimize, de acordo com a sua necessidade, os custos necessários para tolerar falhas em seu modelo de negócio.

Para cumprir o objetivo principal, este trabalho apresenta os seguintes objetivos específicos:

- Construir um modelo distribuído, em ambiente de federação de nuvens, que trabalhe com as políticas de tolerância a falhas proativa e reativa;
- Integrar a um único modelo as técnicas de tolerância a falhas: (1) rejuvenescimento de software (*Software Rejuvenation*), (2) repetição (*Retry*), (3) reenvio de tarefas(*Task Resubmission*) e (4) replicação (*Replication*);
- Pesquisar e implementar rotinas que permitam a configuração, por usuários finais, do nível de tolerância a falhas desejado;
- Implementar um protocolo de *heartbeat* adaptativo para utilização na fase de detecção de falhas;
- Integrar o mecanismo de tolerância a falhas proposto na plataforma de nuvem federada BioNimbuZ;
- Realizar uma análise experimental com injeção de falhas para verificar, de acordo com as métricas de tolerância a falhas, a eficácia, o desempenho, o tempo de resposta, o *overhead* e os custos associados ao modelo proposto.

1.4 Organização do Documento

Este documento contém, além deste capítulo introdutório, mais seis capítulos que contextualizam os ambientes e as tecnologias que fazem parte do escopo do trabalho aqui apresentada.

No Capítulo 2 são apresentados vários aspectos relacionados ao paradigma de computação em nuvem, tais como definições, características essenciais, arquitetura, modelos de serviço e os modelos de implantação e os diferentes tipo de nuvens. Além disto, este capítulo também apresenta uma visão geral da evolução do modelo de computação em nuvem, chamada federação de nuvens. No decorrer deste capítulo são abordados aspectos como definições, benefícios e desafios identificados no novo paradigma de federação de nuvens computacionais.

O Capítulo 3 apresenta aspectos de tolerância a falhas, como conceitos básicos, falha, erro, defeito, classificações de falhas, requisitos da dependabilidade e segurança, medidas de avaliação da disponibilidade e confiabilidade, estratégias para alcançar a dependabilidade e a segurança, bem como os tipos de falhas e suas fases de gerenciamento. Por fim,

são descritas as principais técnicas de tolerância a falhas para ambientes de nuvem e para ambientes de federação de nuvens.

O Capítulo 4 discorre sobre o sistema BioNimbuZ, que é uma plataforma de federação de nuvens computacionais híbridas para a execução de *workflows* de Bioinformática. O BioNimbuZ será utilizado como prova de conceito para a implementação do modelo de tolerância a falhas aqui proposto.

O Capítulo 5 apresenta os detalhes da proposta deste trabalho, um modelo de tolerância a falhas multi-estratégico que funciona de maneira integrável, ou seja, um mecanismo implementado como biblioteca que é utilizado pelo sistema do usuário final. Também são apresentados os principais trabalhos relacionados encontrados na literatura. Finalmente, o capítulo descreve como o modelo proposto foi integrado à plataforma BioNimbuZ.

O Capítulo 6 apresenta a análise experimental para o modelo proposto, sendo constituído por detalhes do ambiente de testes, da injeção de falhas e da inspeção de falhas. Também são apresentados os resultados alcançados para o tempo médio de reparo e os percentuais obtidos para a disponibilidade e para a confiabilidade. Por fim, este capítulo apresenta o *overhead* associado à utilização do modelo de tolerância a falhas proposto.

Para finalizar, o Capítulo 7 apresenta uma visão geral do que foi apresentado neste trabalho, bem como as considerações finais obtidas com a realização desta pesquisa e algumas sugestões para trabalhos futuros.

Capítulo 2

Computação em Nuvem

Este capítulo contextualiza o paradigma da computação em nuvem, tratando de vários aspectos, tais como as definições mais citadas na literatura, características essenciais, arquitetura, modelos de serviço e, por fim, os modelos de implantação e os diferentes tipos de nuvens. Além disto, também discorre-se sobre a visão geral da chamada evolução do modelo de computação em nuvem, a federação de nuvens, abordando aspectos como definições, benefícios e desafios identificados no novo paradigma de federação de nuvens computacionais.

2.1 Visão Geral

A computação em nuvem vem cada vez mais ganhando popularidade em relação aos sistemas tradicionais de tecnologia da informação. Provedores deste tipo têm aplicado amplo investimento para a construção de grandes centros de dados (DC, do inglês *Data Centers*), distribuindo-os em várias regiões geográficas com o objetivo de atender, com eficácia, a crescente demanda por serviços hospedados em seus ambientes de nuvem computacional. Geralmente, estes DCs são construídos com a utilização de centenas de milhares de servidores de *commodities*, e a tecnologia de virtualização é, tipicamente, utilizada para provisionar os recursos computacionais solicitados através da Internet, segundo o modelo de negócio *pay-per-use* [33].

Assim, a computação em nuvem é um paradigma de computação distribuída que surgiu como tendência para o provisionamento de recursos e serviços de maneira ágil, barata, escalável e flexível [34]. Diversos tipos de recursos podem ser disponibilizados, tais como memória, capacidade de processamento, de armazenamento, dentre outros. Atualmente, o modelo de nuvem computacional é extensamente difundido. Muitas definições de computação em nuvem foram propostas nos últimos anos.

Segundo Foster *et al.* [34], a nuvem computacional é definida como um paradigma de computação distribuída em larga escala, impulsionado por uma economia de escala, na qual um *pool* abstraído de recursos computacionais virtualizados, dinamicamente escalável e gerenciado, bem como um *pool* de armazenamento, plataformas e serviços, são entregues sob demanda para consumidores externos por meio da Internet.

De acordo com o Instituto Nacional de Padrões e Tecnologia (NIST, do inglês, *National Institute of Standards and Technology*) [10], a nuvem computacional é definida como um modelo que possibilita de forma ubíqua, conveniente e sob-demanda, o acesso a um *pool* compartilhado de recursos configuráveis, por meio de acesso à rede, podendo ser rapidamente provisionados ou liberados com um mínimo esforço de gerenciamento ou de interação com os provedores do serviço.

Além disso, Armbrust *et al.* [35] a definem como a união de aplicações fornecidas como serviços, por meio da Internet, com o hardware e o software utilizados em DCs que fornecem esses serviços.

Finalmente, Buyya *et al.* [36] definem a computação em nuvem como sendo um tipo de sistema paralelo e distribuído, constituído de uma coleção de computadores interconectados e virtualizados, apresentados como um ou mais recursos de computação unificada que podem ser provisionados dinamicamente, baseados em Acordos de Nível de Serviço (SLA, do inglês *Service-Level Agreement*) estabelecidos através da negociação entre o provedor de serviços e seus consumidores.

Assim, embora existam diferenças consideráveis entre as definições de computação em nuvem, a maioria delas afirma que um sistema de computação em nuvem deve fornecer: (1) serviços sob o moledo *pay-per-use*; (2) elasticidade e ilusão de recursos infinitos; (3) serviços sob demanda; e (4) recursos virtualizados [19]. A Seção 2.2 descreve as principais características de uma plataforma de nuvem.

2.2 Características

Mell *et al.* [10] elencam cinco caraterísticas como sendo essenciais para compor o modelo de computação em nuvem, são elas:

1. **Serviço sob demanda**: clientes podem provisionar recursos computacionais sem a necessidade de negociação ou intervenção humana com os provedores;
2. **Acesso amplo à rede**: serviços em nuvem estão disponíveis por meio da Internet e podem ser acessados por meio de protocolos de rede padrão;
3. **Agrupamento de recursos**: os recursos fornecidos pelos provedores aos seus consumidores se apresentam como um grande *pool* de recursos, nos quais os consu-

midores os consomem de acordo com a necessidade e de forma dinâmica, sem a necessidade de conhecer a infraestrutura utilizada, apenas algumas informações de alto nível, como a região geográfica, são disponibilizadas aos consumidores;

4. **Elasticidade**: recursos podem ser dinamicamente provisionados ou liberados sob demanda, podendo ocorrer até mesmo de maneira automática;

5. **Serviço medido**: o provedor monitora e controla os recursos utilizados pelos usuários da nuvem, a fim de fornece níveis adequados de relatório de acordo com o tipo de serviço, objetivando possibilitar a cobrança aos consumidores com base no modelo *pay-per-use*.

Murugesan *et al.* [8], ainda adiciona mais uma característica como sendo essencial ao modelo de computação em nuvem:

6. **Múltiplos inquilinos ou *Multitenancy***: consumidores utilizam recursos compartilhados por meio de agrupamento de recursos como um único recurso. Isso implica na utilização de recursos por vários consumidores, chamados de inquilinos, simultaneamente.

Essas características são o que diferenciam a plataforma de computação em nuvem de outras plataformas de computação distribuída.

2.3 Arquitetura e Modelos de Serviço

Várias arquiteturas de nuvens foram propostas na literatura. Geralmente, a arquitetura de um sistema de computação em nuvem pode ser dividida em três camadas, que correspondem respectivamente às camadas de Aplicação, Plataforma e Infraestrutura, conforme exemplificado na Figura 2.1.

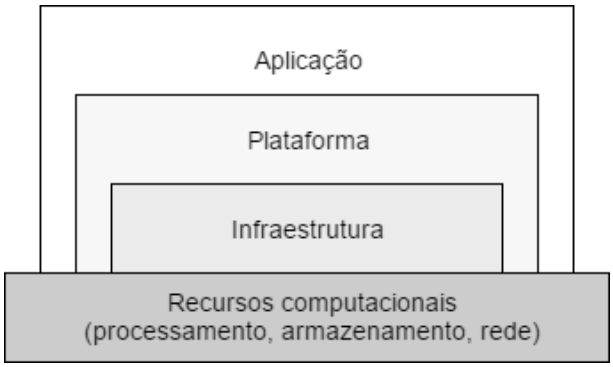

Figura 2.1: Arquitetura de Nuvem Computacional, adaptado de Toosi *et al.* [1].

Além disso, recursos computacionais, recursos de rede, aplicativos ou qualquer outro tipo de serviço de tecnologia da informação oferecido a um usuário por um provedor de nuvem, é chamado de serviço em nuvem. Os serviços em nuvem vão desde aplicativos simples (*e.g.*, e-mail, calendário, processamento de texto e compartilhamento de fotos) até aplicativos complexos, que geralmente necessitam de recursos de computação escalável. Os modelos de serviços em nuvem computacionais podem ser classificados em três categorias principais, de acordo com o nível de abstração fornecido aos consumidores [1]: (1) Software como Serviço (SaaS, do inglês *Software as a Service*), plataforma como serviço (PaaS, do inglês *Platform as a Service*) e infraestrutura como serviço (IaaS, do inglês *Infrastructure as a Service*). Apesar de ser possível encontrar na literatura mais de três categorias [37], é mais comum dividir os serviços nas três categorias aqui mencionadas. Cada categoria de serviço pode ser utilizada de forma independente ou de forma combinada.

Em um nível mais alto, está o modelo de Software como Serviço (SaaS), em um nível mais intermediário o modelo de Plataformas como Serviço (PaaS) e no nível mais baixo, mais próximo dos componentes de hardware, está o modelo de Infraestrutura como Serviço (IaaS). Os modelos de serviços podem ser descritos da seguinte forma [8, 10, 37, 38, 39]:

- **SaaS**: modelo de fornecimento de softwares que tem como característica principal a *multitenancy* (multi inquilino), que corresponde ao compartilhamento de recursos computacionais entre vários consumidores que utilizam o mesmo serviço de software. Os usuários podem acessar estes softwares em qualquer lugar, sem a limitação de ter o software instalado na sua máquina;
- **PaaS**: consiste em estruturas de aplicativos e uma coleção de ferramentas especializadas no topo da camada de infraestrutura, para fornecer uma plataforma de desenvolvimento/implantação, visando minimizar o ônus da implantação de aplicativos diretamente em recursos virtuais;
- **IaaS**: contém recursos abstraídos, tipicamente através da utilização de técnicas de virtualização, criando um conjunto de recursos computacionais expostos como recursos integrados às camadas superiores e aos usuários finais. Esta camada é um componente importante da computação em nuvem, uma vez que muitos recursos, como a atribuição de recursos elásticos, são disponibilizados nesta camada [39].

A Tabela 2.1 resume como é realizada a usabilidade e o controle para os consumidores dos modelos de serviços em nuvem computacional.

Tabela 2.1: Usabilidade e Controle dos Modelos de Serviços [8].

Modelo	**Utilidade ao consumidor**	**Limite de utilização ao consumidor**
SaaS	Utilizar aplicativos que são executados na nuvem.	A configuração de aplicativos é limitada; e não há controle sobre a infraestrutura, rede, servidores, sistemas operacionais, armazenamento ou recursos de aplicativos individuais.
PaaS	Implantar aplicativos na infraestrutura da nuvem (linguagens de programação, bibliotecas, serviço e ferramentas).	O usuário tem o controle de aplicativos implantados e suas configurações de ambiente, mas nenhum controle de infraestrutura, rede, servidores, sistemas operacionais ou armazenamento.
IaaS	Provisionamento de recursos de processamento, armazenamento, redes, entre outros; implantação e execução de sistemas operacionais e aplicativos.	O usuário tem controle de sistemas operacionais, armazenamento e aplicativos implantados, atribuídos ao usuário, mas sem controle sobre a infraestrutura.

2.4 Modelos de Implantação

As nuvens são classificadas em quatro categorias, com base no local onde a nuvem é implantada, por quem a implanta, por quem a gerencia e por quem são seus principais usuários. As principais categorias de implantação são [10, 37]:

- **Nuvem Pública**: qualquer organização ou consumidor (público geral) é capaz de acessar e provisionar recursos da nuvem em questão;
- **Nuvem Privada**: a infraestrutura é provisionada apenas para uma determinada organização, somente ela e seus membros possuem acesso aos recursos;
- **Nuvem Comunitária**: a infraestrutura é compartilhada por organizações que mantém algum tipo de interesse em comum (jurisdição, segurança, economia), e pode ser administrada, gerenciada e operada por qualquer uma das organizações participantes da comunidade;
- **Nuvem Híbrida**: é a combinação de dois ou mais modelos anteriormente citados, obtida por tecnologias ou padrões que permitem a portabilidade de dados e serviços.

2.5 Federação de Nuvens

A fim de obter maior poder computacional ou maior capacidade de armazenamento, muitos usuários estão migrando do modelo de nuvem única para o modelo de nuvem federada, pois neste modelo há a combinação de mais de uma nuvem, o que possibilita o compartilhamento da infraestrutura de outros provedores [15, 16].

Em geral, o modelo de federação de nuvens é utilizado para estender ou potencializar a capacidade de sistemas em nuvem computacional, com o objetivo de atender requisitos de qualidade do serviço [40]. Por exemplo, um único provedor de nuvem pode ser incapaz de provisionar recursos computacionais em um dado momento (por exemplo, em um horário de pico) [41], em outro exemplo, pode não permitir que mais recursos sejam provisionados a um determinado usuário (por exemplo, por questões de limites de utilização) [12, 13, 14].

Assim, problemas como esses podem ser solucionados por meio da distribuição dos serviços em diferentes provedores de nuvem ou em diferentes locais de nuvem [42]. Esse processo é conhecido como federação de nuvem, e provavelmente representa uma nova fase do paradigma de computação em nuvem [36]. A federação de nuvens visa aumentar a disponibilidade de recursos de forma semelhante à que existe em ambientes de nuvem de modelo único. Todavia, a federação em nuvem permite aos usuários diversificar sua carteira de infraestrutura em termos de fornecedores e de localização geográfica. E ainda, permite aos provedores de nuvem aumentarem a capacidade de sua infraestrutura *on-the-fly*, alugando recursos de outros fornecedores, a fim de atender a cargas de trabalho imprevisíveis, sem a necessidade de manter recursos extras em um estado reservado. Além disso, também os ajuda a cumprir SLA [43].

2.5.1 Estágios da Nuvem Federada

Uma federação de nuvem pode ser definida como um modelo de nuvem que tem a finalidade de garantir qualidade de serviço, como desempenho e disponibilidade, permitindo a reatribuição sob demanda de recursos e a migração de carga de trabalho. Isto ocorre por meio de uma rede de diferentes provedores de nuvem para oferecer serviços não triviais aos seus consumidores, isto com base em interfaces normalizadas, e sem uma coordenação centralizada [19].

Dessa forma, a evolução do mercado de computação em nuvem ocorreu em três etapas subsequentes, são elas [2, 3]:

1. **Monolítico**: caracterizada por grandes ilhas proprietárias, com serviços fornecidos,

por empresas de grande porte, como Google[1], Amazon[2] e Microsoft[3], isto de maneira exclusiva e independente umas das outras (Figura 2.2);

Figura 2.2: Estágio Monolítico [2].

2. **Cadeia Vertical de Suprimentos**: ainda com foco em ambientes proprietários, mas aqui as empresas começam a utilizar alguns serviços de outras nuvens, logo, esta etapa dá origem ao eco sistema de integração de nuvens (Figura 2.3);

Figura 2.3: Estágio Vertical [2].

3. **Federação Horizontal**: provedores de nuvem de qualquer tamanho fazem acordos entre si para obter maiores ganhos de economia de escala, uso eficiente de seus ativos e a ampliação de suas capacidades (Figura 2.4).

Figura 2.4: Estágio Horizontal [2].

No estágio de federação horizontal, as nuvens tornam-se uma verdadeira nuvem-de-nuvens (*cloud-of-clouds*), uma grande nuvem composta de nuvens interconectadas, cada uma com suas próprias características, servindo à diferentes necessidades [1, 44].

Em um ambiente federado, cada nuvem pode ter sua própria política de alocação de recursos, o que exige capacidade de negociação dos provedores de nuvem, a fim de negociar os recursos mais adequados, visando alcançar seus objetivos. Isso requer coordenação e orquestração de recursos que pertencem a mais de uma nuvem. Além disto, alguns requisitos são necessários para que se estabeleça uma federação de nuvens, são eles [3]:

[1]Google Cloud Platform, `https://cloud.google.com/`
[2]Amazon Web Services, `https://aws.amazon.com/`
[3]Microsoft Azure, `https://azure.microsoft.com/`

- **Automatismo e Escalabilidade**: uma nuvem, que utilize mecanismos automáticos de descoberta deve ser capaz de eleger, dentre as nuvens participantes da federação, as que melhor satisfaçam às suas necessidades, bem como ser capaz de reagir a mudanças;
- **Segurança Interoperável**: é necessária a integração de diferentes tecnologias de segurança, permitindo que uma nuvem residencial, por exemplo, possa juntar-se à federação sem a necessidade de alterar suas políticas de segurança.

Goiri *et al.* [45], observaram que os provedores apresentam dois comportamentos característicos para criação de federação de nuvens:

- ***Outsourcing***: o provedor, ao perceber que seus recursos estão se esgotando, busca recursos extras em outros provedores. Essa estratégia é importante para que não haja violação de qualidade de serviço por parte do provedor que teve seus recursos esgotados, evitando assim a perda de credibilidade de seus consumidores.
- ***Insourcing***: por outro lado, o provedor que possui recursos subutilizados, para evitar um grande consumo de sua infraestrutura (o consumo de energia elétrica por exemplo, chega a custar 30% do gasto total com infraestrutura [46]), fornece o acesso aos seus recursos para provedores externos que desejam aluga-los.

A Figura 2.5 exemplifica como os recursos podem ser alocados em uma federação, na qual uma Nuvem Local aluga recursos de uma Nuvem Estrangeira A e B (*outsorcing*), e estas por sua vez fornecem recursos disponíveis em sua infraestrutura, para nuvens externas (*insorcing*).

2.5.2 Benefícios

Além dos benefícios já citados, outros benefícios importantes justificam uma utilização mais ampla da federação de nuvens pelos provedores e consumidores, tais como [1, 43, 45]:

- **Melhor aproveitamento de recursos**: (1) recursos subutilizados podem ser alugados, gerando renda e reduzindo custos de manutenção de infraestrutura; (2) empresas que adotam federação de nuvens pode ter sua infraestrutura reduzida, pois neste caso as empresas só contratam recursos extras em momentos de pico, reduzindo assim custos operacionais, energia e espaço;
- **Questões legais**: dados e sistemas que dependem de questões legais específicas, se beneficiam ao serem implantados em regiões geográficas diferentes;

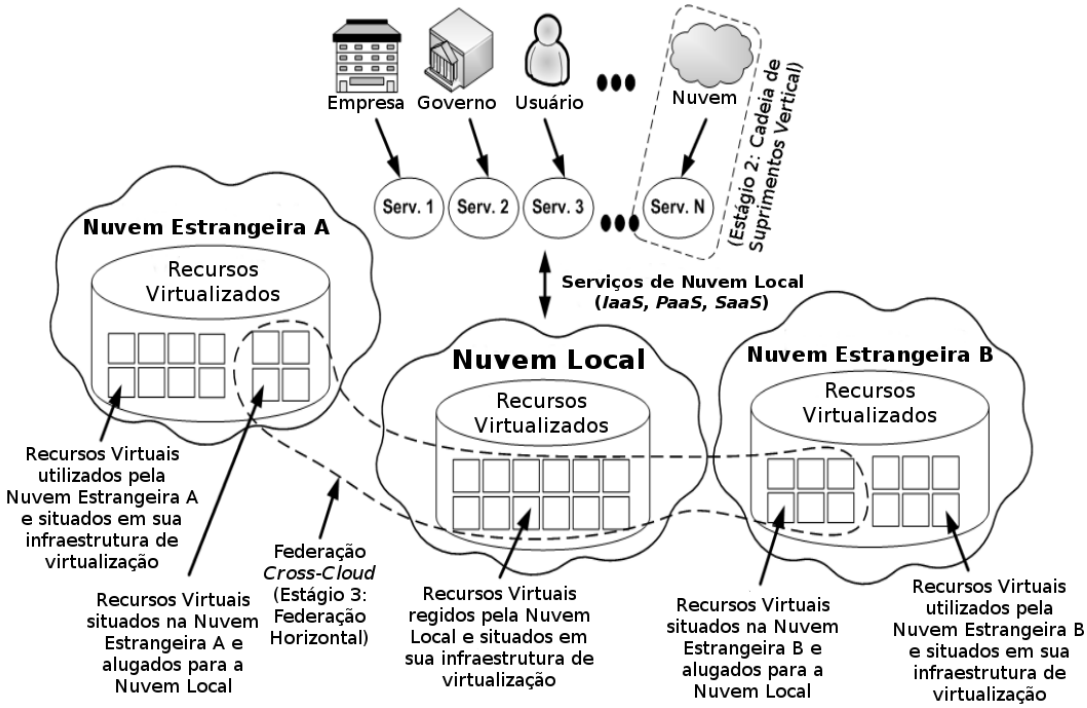

Figura 2.5: Arquitetura para Federação de Nuvens, proposta por Celesti *et al.* [3].

- **Compartilhamento**: a verdadeira noção de compartilhamento entre as partes, tem diferentes políticas organizacionais e capacidades técnicas diferentes;
- **Tolerância a falhas**: a replicação de serviços em diferentes provedores evita a indisponibilidade dos mesmos, no caso de um determinado provedor enfrentar algum tipo de interrupção, o que é bastante comum;
- **Melhor nível de qualidade de serviço**: apresenta maiores opções para minimizar a latência e os atrasos relacionados ao tempo de resposta, isto por meio da utilização de um provedor mais próximo geograficamente, ou por meio da utilização de um provedor mais capacitado;
- **Otimização de custos**: é possível combinar provedores com o objetivo de provisionar recursos mais baratos;
- **Redução na violação de SLA**: no caso de escalonamento de recursos, um determinado provedor de serviços em nuvem pode evitar uma violação de SLA por sua parte, alugando/alocando recursos de outros membros da federação;
- **Independência de provedor**: consumidores não necessitam ficar dependentes de um único fornecedor;

- **Vencimento de contratos**: se o contrato com um determinado provedor está prestes a vencer, reduzem-se a preocupações com o bloqueio de serviços, pois um ambiente federado é composto por mais de um provedor de serviços.

2.5.3 Desafios

Existem diversos desafios dentro do tema da federação de nuvens, pois uma completa federação depende de muitos fatores. Atualmente tem-se vivido uma transição para um estágio em que as nuvens tenham completa comunicação entre si, os maiores provedores de nuvens ainda ditam os padrões do mercado. Os problemas mais comuns observados são [1, 31, 32]:

- **Carência de padrões abertos**: cada provedor adota seu próprio padrão (*e.g.*, mecanismos de segurança e autenticação, comunicação, máquinas virtuais, armazenamento e etc.), dificultando o desenvolvimento de sistemas que fornecem soluções de federação de nuvens. Dessa forma, surge o problema chamado *vendor lock-in*, no qual as empresas ou os desenvolvedores ficam presos à determinadas tecnologias, o que dificulta e até mesmo impossibilita a migração de negócios para outras tecnologias, devido ao alto custo associado (recapacitação ou reciclagem de funcionários também entra no custo);
- **Manutenção de SLA**: os SLA diferem-se entre nuvens, e garantir que estejam de acordo entre todas as nuvens participantes é uma tarefa muito difícil. É mais fácil que grandes provedoras cubram os custos relacionados a violação de acordos para não perder seus clientes, o que se torna um problema para pequenas empresas, pois custear tais violações inviabilizaria ou prejudicaria seus negócios;
- **Dificuldade de monitoramento**: dificuldade em consolidar dados de monitoramento de recursos de uma federação com uma grande quantidade de nuvens em regiões diferentes. As nuvens em questão podem estar situadas em locais geográficos muito distantes, aumentando assim a latência da rede de comunicação;
- **Confiança**: empresas se mostram relutantes em manter suas informações fora de seu domínio, tanto por questões de segurança quanto pelo controle adequado dessas informações.

2.6 Considerações Finais

Este capítulo tratou de aspectos do modelo de nuvem computacional, os quais evidenciaram que os principais benefícios da adoção do modelo de nuvem incluem a redução de

custos operacionais, a flexibilidade aprimorada, a escalabilidade sob demanda, a implantação de aplicativos de maneira mais fácil e rápida, a facilidade de uso e a disponibilidade de um vasto campo de recursos computacionais. Muitos tipos de serviços ou aplicações, incluindo e-mail, ferramentas utilitárias para escritórios e armazenamento de dados, continuam migrando para o modelo de nuvem com o objetivo de colher os benefícios desse novo paradigma em tecnologia de informação e aumentar a capacidade de atender suas demandas.

Contudo, Murugesan [8] *et al.* alertam que usuários devem considerar as limitações existentes, antes de migrar para o modelo de nuvem computacional. As principais limitações presentes no modelo de nuvem são:

- Necessidade de acesso a rede confiável, sempre disponível e de alta velocidade para se conectar e utilizar os serviços providos através de nuvens;
- Possibilidade de queda do tempo de resposta, devido ao aumento do tráfego em horários de pico ou incertezas na infraestrutura da rede;
- Vulnerabilidades adicionais à segurança de dados e aos processos em nuvem;
- Risco de acesso não autorizado a dados organizacionais e até mesmo a dados de consumidores.

Além disto, apesar dos benefícios verificados para o modelo de computação em nuvem, atualmente, ele já não é mais suficiente, em alguns casos, para suprir a demanda cada vez mais crescente de recursos computacionais.

O modelo de federação de nuvem descrito neste capítulo mostrou-se um conceito que tem grande potencial e pode ter uma enorme influência na forma como os recursos e as aplicações de computação serão manipulados, desenvolvidos e utilizados. É uma etapa adicional proporcionar recursos de computação de uma forma semelhante a outros serviços básicos, como por exemplo eletricidade ou água. No entanto, a evolução do modelo de nuvens é extremamente dinâmica, assim é difícil predizer seu rumo a longo prazo.

Apesar da crescente utilização do modelo de nuvens e das muitas definições e padrões já existentes, o modelo de federação de nuvens ainda não possui padrões universais de interfaces e arquiteturas. Em outras palavras, a construção de uma federação necessita de diferentes implementações de *plugins* que possibilitem a integração de infraestruturas heterogêneas. Barril *et al.* [31] afirmam que é necessária a criação e definição de padrões universais para impulsionar ainda mais o modelo de nuvens federadas. A padronização possibilitaria a formação de nuvens federadas com maior facilidade.

Como consequência dos desafios descritos neste capítulo, nota-se que o monitoramento e o tratamento de falhas em sistemas federados são problemas não triviais. Além do mais,

o modelo de nuvens federadas ainda herda todos os desafios existentes no modelo de nuvem única. Mesmo DCs cuidadosamente projetados estão sujeitos a um grande número de falhas, principalmente, quando distribuídos ou espalhados em várias regiões geográficas [47]. Dados esses fatos, é evidente que a tolerância a falhas é uma importante questão tanto para provedores quanto para consumidores de nuvens. O Capítulo 3 discorre sobre conceitos de tolerância a falhas.

Capítulo 3

Tolerância a Falhas

Este capítulo apresenta aspectos da tolerância a falhas, tais como conceitos básicos, falha, erro, defeito, classificações de falhas, requisitos da dependabilidade e segurança, medidas de avaliação da disponibilidade e confiabilidade, estratégias para alcançar a dependabilidade e segurança, bem como os tipos de falhas e suas fases de gerenciamento. Além disso, são descritas as principais técnicas de tolerância a falhas para ambientes de nuvem e federação de nuvens computacional. Por fim, também é apresentado a relação de trabalhos relacionados encontrados na literatura.

3.1 Falha, Erro e Defeito

Assume-se que um sistema falha quando o mesmo não atende às suas especificações [48]. Falhas em certos tipos de sistemas podem ser catastróficas para a segurança de usuários ou para a imagem e reputação das empresas. À medida que a popularização do uso de computadores e sistemas distribuídos aumenta, a necessidade de evitar falhas torna-se, consequentemente maior.

Antes de prosseguir, é fundamental deixar claro o que é falha, erro e defeito, ou seja, definir conceitos que nos permita diferenciar cada um destes termos. Aqui serão apresentados conceitos utilizados por grande parte da comunidade científica, que foram derivados de Avizienis *et al.* [5], Laprie [49] e Lee *et al.* [50].

Assim, para esses autores, falha é definida como a causa física ou algorítmica do erro [4]. Em outras palavras, as falhas estão associadas ao universo físico. O erro é um estado que ocorre quando o processamento posterior a uma falha levar ao defeito, ou seja, o erro está relacionado ao universo de informação. E por último, o defeito, que não pode ser tolerado, mas deve-se evitar que o sistema apresente defeito. Defeitos são associados ao universo do usuário.

Uma falha não necessariamente ocasiona um erro, pois uma falha de hardware, por exemplo, pode não ser percebida ou ocorrer em uma área que nunca será utilizada pelo sistema. Da mesma forma, um erro não necessariamente leva o sistema a um defeito, pois tal erro pode não refletir ou não ser percebido no universo do usuário. A Figura 3.1, exemplifica o conceito apresentado para as definições de falha, erro e defeito.

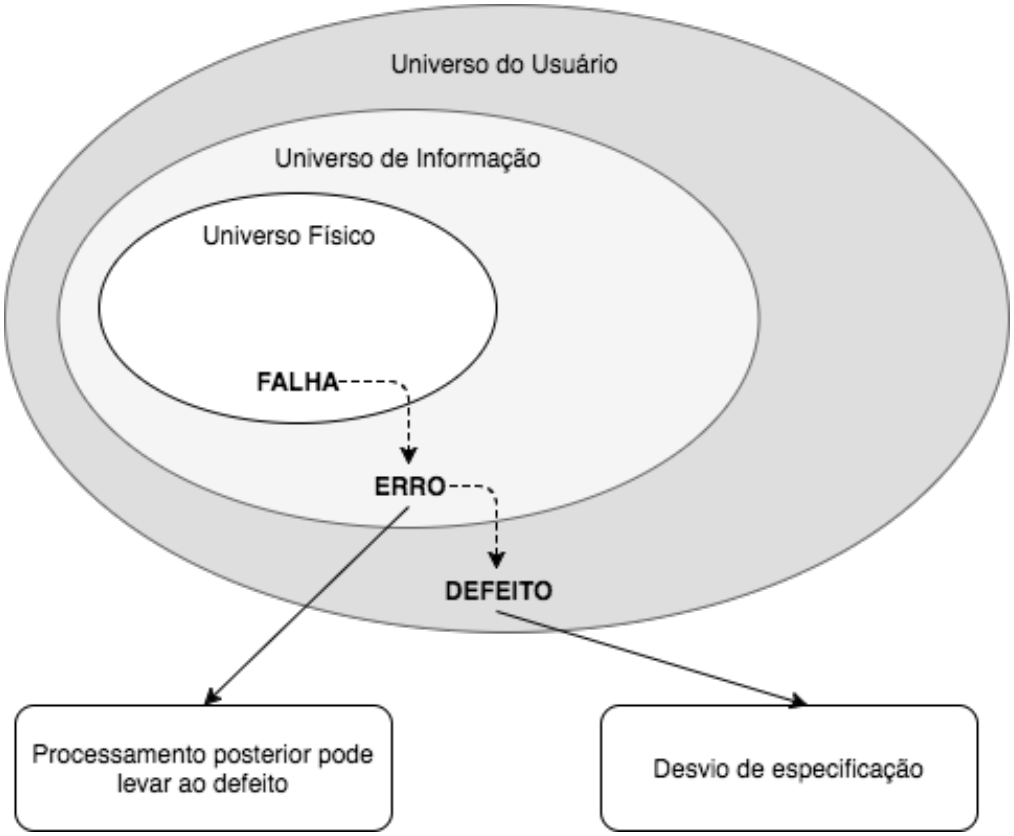

Figura 3.1: Universo de Falha, Erro e Defeito, adaptado de Weber [4].

Conforme ilustrado na Figura 3.1, para que exista um defeito no sistema é necessário que antes ocorra uma falha seguida de um erro, que exponha o defeito ou o torne perceptível ao usuário. Neste contexto, define-se que a latência de falha é dada pelo tempo o qual a falha ocorre, até o dado momento em que o erro se manifesta. Já a latência do erro é dada pelo tempo passado desde a ocorrência do erro até a manifestação do defeito [4].

3.2 Classificação de Falhas

As falhas são, geralmente, classificadas como transientes, intermitentes ou permanentes. Estas classificações podem ser definidas conforme segue [9, 51, 52]:

- **Falha transiente**: é aquela que ocorre num dado momento e desaparece em execuções futuras. Em outras palavras, um sistema pode apresentar a falha em uma de suas execuções, porém esta mesma falha não se manifesta novamente em execuções futuras e idênticas. Como exemplo, a falha pode ocorrer devido uma interferência ou mal funcionamento na comunicação de rede, que pode causar perdas de pacotes num certo momento, porém ao se repetir a operação, esta perda de pacotes pode não ocorrer, o que resultará na execução conforme especificado, ou seja, na execução normal do sistema;
- **Falha intermitente**: é aquela que aparece esporadicamente sem razão aparente. Tipicamente está relacionada a defeitos no hardware ou software, assim um componente defeituoso gera, de forma irregular, uma série de comportamentos fora do especificado. Falhas intermitentes são difíceis de lidar, pois se manifestam de maneira aparentemente aleatória e em intervalo de tempo irregular, mesmo observando uma execução idêntica do sistema repetidas vezes. Logo, o comportamento imprevisível de falhas intermitentes dificulta o seu diagnóstico;
- **Falhas permanentes**: é aquela que continua a existir a menos que o componente defeituoso, tanto no hardware quanto no software, seja substituído ou corrigido.

É difícil determinar se uma falha é transiente ou intermitente simplesmente pela observação do sistema, pois falhas destas classificações podem possuir as mesmas características. Distinguir estas falhas é importante, pois a falha transiente não implica necessariamente que o sistema deve ser declarado defeituoso, pois um ambiente instável pode justificar um defeito temporário [7].

No geral, as principais causas de falhas são problemas de especificação, implementação, operação e envelhecimento de componentes de hardware. Além disto, existem as interferências do ambiente externo, como temperatura, umidade e pressão.

Em sistemas distribuídos, existe ainda a noção de falhas parciais, sendo esta uma característica que difere os sistemas centralizados de sistemas distribuídos. Falhas parciais ocorrem quando um componente do sistema distribuído falha. Este tipo de falha pode acarretar ou não no mal funcionamento de outros componentes. Todavia, falhas em sistemas distribuídos muitas vezes afetam a todos os componentes do sistema, podendo facilmente causar a indisponibilidade total [9].

Em ambientes de nuvem computacional é comum que a causa de falhas seja dada, ainda, por problemas de comunicação, de latência e de localidade de recursos computacionais [1].

Sistemas distribuídos tem como um importante desafio continuar sua operação mesmo sob a existência de falhas, ou seja, eles devem ser capazes de detectar e recuperar-se

automaticamente de falhas parciais, sem que seu desempenho seja comprometido, isto é, o sistema deve tolerar falhas, permitindo que elas ocorram, porém, sem lhe causar considerável impacto.

3.3 Dependabilidade e Segurança

Segundo Avizenis *et al.* [5], a dependabilidade é definida pela capacidade de fornecer um serviço que possa ser justificadamente confiável, e esta definição ressalta a necessidade de se justificar a confiança. Em uma definição alternativa, a dependabilidade de um sistema é a capacidade de evitar falhas de serviço que ocorrem com mais frequência e gravidade do que pode-se considerar aceitável.

Na taxonomia de sistemas confiáveis e seguros, proposta por Avizienis *et al.* [5], um sistema é uma entidade que interage com outras entidades, ou seja, outros sistemas, incluindo hardware, software, *stakeholders* e fenômenos naturais. Esta interação afeta a computação e a comunicação entre os sistemas, que é caracterizada por importantes aspectos relacionados à tolerância a falhas, que impactam na disponibilidade, isto é, na confiabilidade, na segurança e no desempenho.

Atualmente, o meio científico tem tido a tolerância a falhas como um importante objeto de pesquisa. Primeiramente, é necessário entender o que realmente significa tolerar falhas para um sistema distribuído. Desta forma, prover tolerância a falhas está diretamente relacionado a dependabilidade de sistemas. Segundo Kopetz *et al.* [53], existe uma série de requisitos para sistemas distribuídos que estão contidos no termo dependabilidade, tais como a disponibilidade, a confiabilidade, a proteção e a manutenibilidade.

Assim, referente à dependabilidade, Tanenbaum *et al.* [9] e Avizienis *et al.* [5] definem os principais requisitos ou atributos para sistemas distribuídos, conforme abaixo:

- **Disponibilidade**: propriedade na qual um sistema deve manter-se disponível para utilização imediata. Logo, refere-se a probabilidade de um sistema estar operando conforme o esperado, e em qualquer dado momento estar disponível para atender suas requisições;
- **Confiabilidade**: propriedade na qual um sistema mantém sua execução continuamente sem falhas. Diferentemente da disponibilidade, que é definida por um dado momento, a confiabilidade é definida por um intervalo de tempo relativamente longo e é relativo à corretude do serviço provido. Dado este fato, não se deve confundir disponibilidade com confiabilidade;
- **Proteção** (segurança operacional ou de funcionamento): propriedade na qual mesmo perante um mal funcionamento do sistema, nada catastrófico acontece. A proteção

é muito importante em sistemas que podem por em risco a integridade física de seus usuários, por exemplo, sistemas que operam em controle de tráfego aéreo, usinas nucleares, cirurgias médicas, lançamentos espaciais, dentre outros. Existem muitos exemplos que mostram o quão complexo é construir sistemas seguros;

- **Integridade**: é a propriedade de sistemas resistirem a ataques externos, não permitindo que ocorram alterações impróprias no sistema, sejam elas causadas de forma acidental, intencional ou devido a um mal funcionamento do sistema;
- **Manutenibilidade**: é a facilidade com que um sistema, após apresentar falhas, pode ser reparado. Sistemas com alta manutenibilidade favorecem a alta disponibilidade, principalmente, se este for capaz de detectar e recuperar-se das falhas automaticamente.

Já em relação à segurança, destacam-se a combinação dos requisitos confidencialidade, integridade e disponibilidade, sendo que, os dois últimos também se aplicam à dependabilidade. Avizienis *et al.* [5] e Jonsson [54] os definem como:

- **Confidencialidade**: é a propriedade de não expor ou divulgar informações não autorizadas, ou seja, informações confidenciais;
- **Integridade**: aqui, a definição dada anteriormente se aplica da mesma maneira, ou seja, este requisito é comum tanto à dependabilidade quanto à segurança;
- **Disponibilidade**: já definida anteriormente, porém neste contexto, adiciona-se o fato de que o sistema deve manter-se disponível apenas para requisições autorizadas.

A Figura 3.2 indica a principal relação de interesse e de atividade entre a dependabilidade e a segurança. A dependabilidade e a segurança especificadas para um sistema devem considerar a gravidade e a frequência aceitável de falhas em um determinado ambiente de aplicação. Para determinados sistemas, um ou mais dos requisitos apresentados nesta seção podem não fazer parte de sua especificação.

3.3.1 Medidas de Avaliação de Dependabilidade

Transparecer o quão um sistema pode ser disponível e confiável, isto é, a medição de tolerância a falhas de um sistema, é um aspecto crucial para sistemas distribuídos e também para o paradigma de computação em nuvem. Estas medidas são utilizadas para quantificar a dependabilidade de sistemas. As duas principais medidas para tolerância a falhas de um sistemas são a disponibilidade e a confiabilidade. A disponibilidade é dada pela razão entre o tempo disponível, e a soma entre o tempo disponível e o tempo indisponível de um sistema. A disponibilidade pode ser quantificada conforme a Equação 3.1 [55].

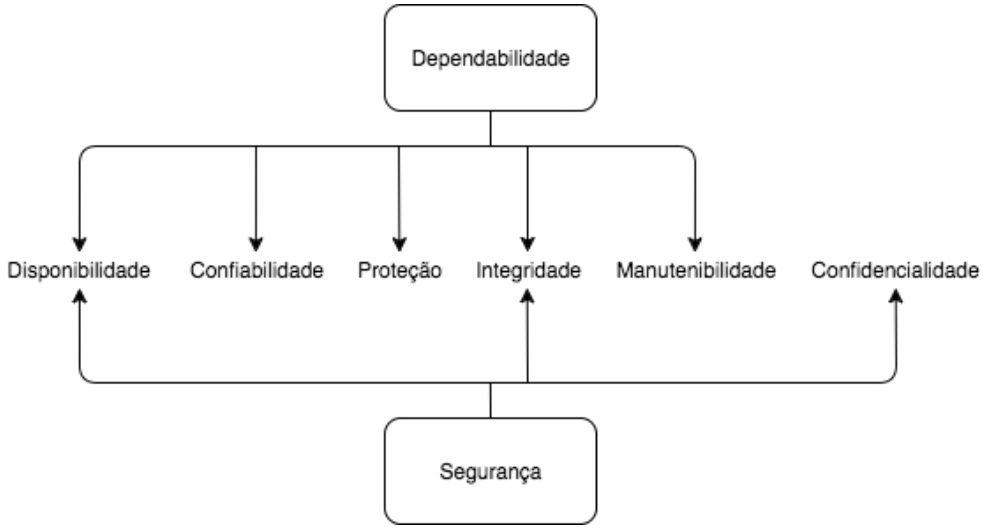

Figura 3.2: Relação de Requisitos de Dependabilidade e de Segurança [5].

$$Disponibilidade = \frac{Tempo\ disponível}{Tempo\ disponível + Tempo\ indisponível} \tag{3.1}$$

Onde o tempo disponível (ou tempo de atividade), bem como o tempo indisponível (ou tempo de inatividade), podem ser preditos através do uso de modelos de disponibilidade como o de Markov [56], ou também podem ser calculados a partir do monitoramento do sistema [6].

O percentual do tempo de serviço disponível e o tempo de inatividade do serviço também expressa a disponibilidade do sistema. O tempo de serviço disponível é o tempo operacional esperado para um sistema (por exemplo, tempo operacional esperado por mês). Esta medida é calculada conforme apresentado na Equação 3.2 [57].

$$Disponibilidade = \frac{Tempo\ disponível - Tempo\ indisponível}{Tempo\ disponível} * 100 \tag{3.2}$$

Métricas como o tempo médio para o defeito (MTTF, do inglês *Mean Time To Failure*), tempo médio entre os defeitos (MTBF, do inglês *Mean Time Between Failures*) e o tempo médio de reparo (MTTR, do inglês *Mean Time To Repair*) também podem ser utilizadas para quantificar a disponibilidade de um sistema. O MTTF é o tempo médio que o sistema opera corretamente, conforme o especificado, até que ocorra um defeito. O MTBF é o tempo médio entre dois defeitos apresentados consecutivamente pelo sistema, ou seja, é o tempo esperado para que o sistema apresente novamente o defeito. Finalmente, o MTTR é o tempo necessário para que o sistema passe do estado de defeito para um estado livre de defeitos, voltando assim, a operar corretamente. Segundo estas métricas, a disponibilidade pode ser calculada como apresentado na Equação 3.3 [57].

$$Disponibilidade = \frac{MTBF}{MTBF + MTTR} \tag{3.3}$$

A Equação 3.3 e seus termos podem ser facilmente compreendidos quando considerada a ilustração da Figura 3.3. Também é possível notar que a Equação 3.3 é equivalente a fórmula da disponibilidade apresentada na Equação 3.1, na qual o MTBF é igual ao tempo disponível, e o MTTR é igual ao tempo indisponível.

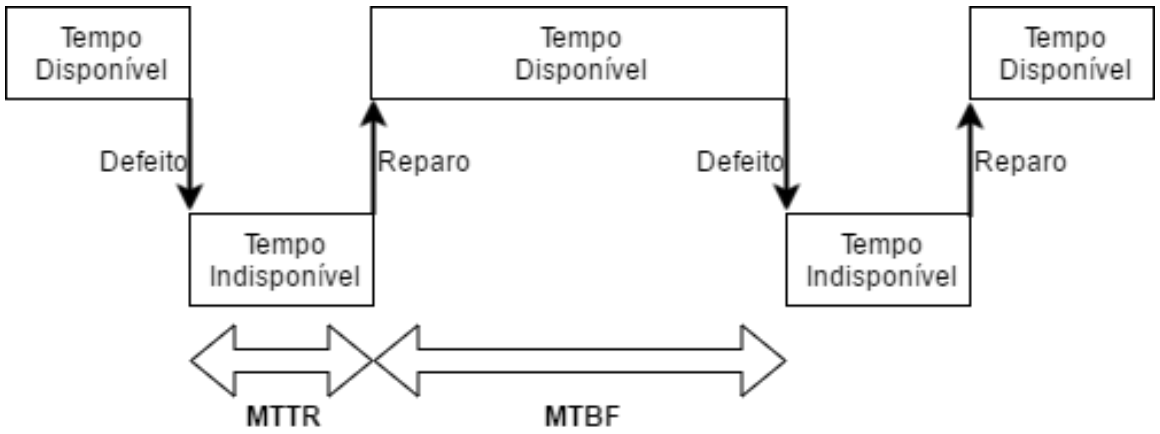

Figura 3.3: MTBF e MTTR, adaptado de Bauer [6].

Por outro lado, a confiabilidade é denotada como $R(t)$, que é a probabilidade de um sistema funcionar corretamente em uma função de tempo t [55]. Conforme definido anteriormente, a confiabilidade é a propriedade de um sistema manter sua execução continuamente sem falhas. Já Bauer *et al.* [6], relatam que o manual de medição TL9000 define a confiabilidade como a capacidade de um sistema executar corretamente uma função requisitada por um determinado período de tempo. A confiabilidade de um serviço pode ser quantificada de acordo com a Equação 3.4 [6].

$$Confiabilidade = \frac{Respostas\ bem\ sucedidas}{Total\ de\ requisições} * 100 \tag{3.4}$$

Também foi dito anteriormente na Seção 3.3 que a confiabilidade é definida por um intervalo de tempo relativamente longo. Por este motivo, a confiabilidade é quantificada, na prática, pela quantidade de requisições sem êxito sobre o total de milhões de requisições efetuadas (DPM, do inglês *Defects Per Million*). O DPM é formulado pela Equação 3.5 [8].

$$Confiabilidade = \frac{Total\ de\ requisições - Respostas\ bem\ sucedidas}{Total\ de\ requisições} * 1.000.000 \tag{3.5}$$

Assim, a disponibilidade e a confiabilidade de um sistema têm papeis fundamentais, principalmente, no paradigma de nuvem, visto que, uma pequena fração de tempo de

inatividade pode resultar em graves impactos financeiros. Desta forma, foi estimado que interrupções em serviços de tecnologia da informação ocasionam uma perda de receita de aproximadamente U$ 26,5 bilhões por ano [58]. Dito isto, é fácil perceber que a disponibilidade e a confiabilidade de serviços em nuvem computacional são questões fundamentais para um ambiente de nuvem.

3.3.2 Meios para Obter Dependabilidade e Segurança

Durante a evolução científica, vários meios foram propostos e desenvolvidos a fim de alcançar os requisitos de segurança e de dependabilidade. Assim, para que os sistemas possam atender aos requisitos de dependabilidade e de segurança, estes devem munir-se de técnicas e métodos de tolerância a falhas. Estes meios podem ser agrupados em quatro principais categorias, conforme a seguir [5]:

- **Prevenção de falhas**: meios para evitar o surgimento de falhas. Em outras palavras, impede a ocorrência ou a introdução de falhas no sistema;
- **Tolerância a falhas**: meio para evitar falhas e fornecer o serviço corretamente, mesmo sobre a presença de falhas;
- **Remoção de falhas**: meio para verificar a presença e reduzir o número e a gravidade das falhas;
- **Predição de falhas**: meio para estimar o número atual, a incidência futura e as consequências prováveis de falhas.

A capacidade de fornecer um serviço confiável é objetivo da prevenção de falhas e da tolerância a falhas. Por outro lado, a remoção de falhas e a predição de falhas visam alcançar a confiança nesta capacidade.

Assim, sistemas confiáveis estão diretamente relacionados ao tratamento de falhas. É possível afirmar que existe uma distinção entre a prevenção, a remoção e a predição de falhas [5]. A taxonomia referente à dependabilidade e segurança abordada neste capítulo é ilustrada na Figura 3.4.

3.4 Tipos de Falhas

Conforme mencionado na Seção 3.1, um sistema é dado por uma entidade que interage com outras entidades, isto é, outros sistemas. Em sistemas distribuídos, na maioria das vezes, não é fácil determinar o tipo da falha, pois o mal funcionamento de uma entidade pode não estar relacionado a uma determinada entidade em si, pois, se tal entidade depende de

Figura 3.4: Árvore de Dependabilidade e Segurança, traduzido de Avizienis [5].

outras entidades para operar corretamente, a causa da falha pode ser o mal funcionamento de uma outra entidade a qual esta se relaciona. Esta relação de dependência é bastante comum em sistemas distribuídos [5].

Dessa forma, a fim de obter melhor compreensão sobre a gravidade de uma falha, vários grupos de tipos de falhas foram desenvolvidos, nos quais afirma-se que um subconjunto de classes mais fortes é um subconjunto de classes mais fracas. Os grupos, do mais forte para o fraco, são se-falhar/para (*fail-stop failure*), falha por queda (*crash failure*), falha por omissão (*omission failure*), falha por tempo (*timing failure*), falha de resposta (*response failure*), e falhas arbitrárias (*byzantine failure*). A Tabela 3.1 apresenta uma breve descrição dos tipos de falhas citados.

Assim, seguem mais detalhes dos principais tipos de falhas e suas definições conforme Lamport *et al.* [59], Schlichting *et al.* [60], Laranjeira *et al.* [61], Barborak *et al.* [7] e Tanenbaum *et al.* [9]:

- **Se-falhar/para**: ocorre quando um sistema para e alerta outros sistemas sobre sua falha. Em outras palavras, sempre que ocorre uma falha o sistema para. Este é o tipo mais simples de falha, pois não há a produção de resultados errôneos, ao invés disto, o sistema simplesmente para;
- **Falha por queda**: ocorre quando o sistema perde seu estado interno, parando ou travando antes de concluir sua execução, porém mantinha o processamento correto até a sua parada. Uma característica importante deste tipo de falha é que o sistema se torna incapaz de executar qualquer tipo de operação, ou seja, é como se estivesse

Tabela 3.1: Tipos de Falhas, adaptado de [9].

Tipo de Falhas	**Descrição**
Se-falhar/para	O sistema encerra sua operação sempre que ocorre uma falha.
Falha por queda	O sistema para ou trava, porém operava corretamente antes da falha.
Falha por omissão: -Recebimento -Envio	O sistema falha em responder solicitações: -Falha em receber solicitações; -Falha ao tentar responder solicitações.
Falha por tempo	O sistema responde fora do intervalo de tempo especificado.
Falha por resposta: -Valor -Transição de estado	O sistema responde de forma incorreta: -O valor da resposta enviada é incorreto; -O sistema se desvia do fluxo de operações correto.
Falha arbitrária	O sistema produz respostas arbitrárias aleatoriamente.

congelado. Sistemas operacionais comumente apresentam este tipo de falha, no qual a única solução é a reinicialização do mesmo;

- **Falha por omissão**: ocorre quando o sistema não é capaz de receber uma solicitação (*i.e.*, se comporta como se não a tivesse recebido), ou quando o sistema não é capaz de enviar uma resposta (*i.e.*, a solicitação é conhecida e processada corretamente, porém o sistema não consegue enviar uma resposta). Este tipo de falha pode resultar em muitos problemas, pois um sistema 'A' pode aguardar eternamente uma resposta do sistema 'B', que no primeiro caso, nem ao menos tem conhecimento da solicitação do sistema 'A'; e, o segundo caso, é incapaz de enviar a resposta à solicitação. O primeiro caso, omissão por recepção, não afeta o estado atual do sistema, pois o mesmo não tem conhecimento de qualquer solicitação feita a ele;

- **Falha por tempo**: está relacionado ao tempo de resposta e ocorre quando o sistema completa a execução de uma solicitação antes ou depois do tempo de sua especificação. A resposta precipitada também pode causar problemas ao solicitante, caso este não esteja preparado para recebê-la. Todavia, é mais comum que os sistemas respondam de forma tardia, e neste caso, ocorrem falhas de desempenho;

- **Falha de resposta**: ocorre quando o sistema falha em produzir o resultado correto em resposta a uma solicitação com entradas corretas. Em outras palavras, ocorre quando simplesmente o sistema envia uma resposta incorreta para uma solicitação correta. Este é um tipo de falha grave, visto que, é melhor uma resposta não dada ao invés de uma resposta dada incorretamente. Há dois tipos de falhas de resposta: falha de valor e falha de transição de estado. No primeiro caso, o sistema envia

uma resposta de valor incorreto. Já no segundo caso, o sistema reage de maneira inesperada a uma solicitação, ou seja, o sistema pode tomar ações que nunca deveria ter iniciado;

- **Falhas arbitrárias**: este tipo de falhas também é conhecido como falhas bizantinas, é o tipo de falha que pode ser considerado como um conjunto de falhas universal, e o tipo mais grave de falhas. Falhas arbitrárias podem levar o sistema a produzir saídas incorretas que podem não ser detectadas como incorretas. Neste contexto, sistemas podem operar maliciosamente com outros sistemas objetivando produzir respostas intencionalmente incorretas. Este caso salienta o fato da segurança ser considerada um requisito para sistemas confiáveis. O termo "bizantino"refere-se ao Império Bizantino, tempo e local onde conspirações, intrigas e mentiras eram frequentemente utilizadas por seus governantes [59]. Falhas bizantinas estão intimamente relacionadas a falhas por queda (*crash failure*) e a falhas se-falhar/para (*fail-stop failure*).

A Figura 3.5 ilustra uma representação gráfica dos grupos de tipos de falhas mencionados anteriormente. Os grupos mais internos representam os grupos mais fortes (representados pelos círculos menores na figura), ou seja, sua ocorrência é independente da ocorrência de um tipo de falha mais fraco (representados pelos círculos maiores na figura).

Figura 3.5: Ordem de Força dos Tipos de Falhas, adaptado de [7].

3.5 Fases de Gerenciamento de Falhas

A tolerância a falhas é constituída de quatro fases que, juntas, fornecem os meios gerais para a implementação de tolerância a falhas, visando assim, evitar que falhas gerem defeitos no sistema. As três primeiras fases estão relacionadas com a descoberta e com a maneira de lidar com os erros, já a quarta fase, está relacionada com as falhas que deram origem aos erros. Logo, as quatro fases, segundo Lee *et al.* [62], são:

- **Detecção de erros**: o ponto de partida para técnicas de tolerância a falhas é a detecção de um estado errôneo. De acordo com a definição apresentada para uma falha, na Seção 3.1, a manifestação da falha poderá gerar erros em algum lugar do sistema, ou seja, para tolerar falhas em um sistema, seus efeitos devem primeiramente ser detectados;
- **Avaliação de danos**: após a detecção de um erro, deve-se avaliar os possíveis danos em todo o estado do sistema, pois informações incorretas podem ter se espalhado pelo sistema, isto devido ao intervalo de tempo entre a manifestação do erro e a detecção de suas consequências. Logo, antes de tentar lidar com o erro detectado, faz-se necessário avaliar até que ponto o estado do sistema foi danificado;
- **Recuperação de erros**: é um dos aspectos mais importantes da tolerância a falhas, e na literatura é uma das áreas mais exploradas. Após a detecção e a avaliação de danos, devem ser utilizadas técnicas que visam transformar o estado errôneo em um estado correto, para que o sistema possa retornar e continuar com sua operação normal;
- **Tratamento de falhas**: falhas cujos efeitos tenham sido tratados na fase de recuperação, devem ainda ser tratadas para que não voltem a se repetir. A detecção de erros não identifica a falha e, frequentemente, a causa de um erro é dada por várias falhas. Assim, o principal objetivo do tratamento de falhas é localizar precisamente a origem do erro, ou seja, a falha, para então reparar ou reconfigurar o sistema para evitar a recorrência da falha. Porém, nenhuma ação é necessária se a falha é identificada como transiente.

A base para a concepção e a implementação de sistemas tolerantes a falhas é constituída pelas quatro fases citadas, além disto, elas também são a base para todas as técnicas de tolerância a falhas [50].

Nem sempre todas as quatro fases são realmente necessárias em um sistema de computação tolerante a falhas. Isto pode variar de sistema para sistema, por exemplo, frequentemente decisões tomadas durante a especificação de um projeto podem evitar a

necessidade de medidas como a avaliação de danos e a localização de falhas. Isto é dito pois estas são técnicas exploratórias ou heurísticas, que são difíceis de implementar de forma eficiente e confiável [62]. Embora essas fases não necessitem estar presentes em um determinado sistema, isto não significa que elas não são importantes no que se refere a tolerância a falhas. Isto simplesmente reflete o fato de que elas podem ser consideradas durante a especificação do projeto de um sistema.

3.6 Tolerância a Falhas em Ambientes de Nuvem

Muitos dos novos desafios enfrentados pela comunidade de computação em nuvem estão relacionados aos conceitos de dependabilidade e de segurança, pois as falhas neste tipo de ambiente podem acarretar em graves prejuízos econômicos, além disto, falhas são cada vez mais comuns neste tipo de ambiente [58, 63].

Um dos maiores desafios à utilização de tolerância a falhas em uma arquitetura de nuvem computacional é elencar a melhor maneira de implementar as duas principais fases de gerenciamento de falhas, a detecção e o reparo. Isso porque neste tipo de arquitetura existem diferentes níveis de acesso para os participantes (*i.e.*, provedores e consumidores). Assim, uma importante questão a ser definida é em relação ao melhor ou ao mais indicado participante da nuvem para implementar as duas fases de gerenciamento de falhas, isto de acordo com suas permissões de acesso. Isso porque (1) falhas de hardware apenas podem ser detectadas e reparadas pelos provedores de nuvem, (2) falhas em instâncias de máquinas virtuais podem ser detectadas por ambos os participantes, mas só podem ser reparadas pelos provedores, e (3) falhas na aplicação só podem ser detectadas pelos usuários ou consumidores, mas também podem ser reparadas por ambos os participantes [64].

A manutenção, a atualização, os reparos, a carga intensa de solicitações aos servidores, dentre outras, são as principais razões que podem induzir falhas em ambientes complexos e dinâmicos de nuvem computacional [24, 27, 30, 57]. Logo, os riscos para usuários que executam suas aplicações neste tipo de ambiente são bastante significativos, visto que as falhas que ocorrem nos DCs dos provedores podem estar fora do escopo da organização do usuário.

Um outro aspecto do modelo de federação de nuvens é o maior potencial de uso de técnicas de tolerância a falhas para aumentar a dependabilidade de aplicações espalhadas pelos vários provedores de nuvem participantes da federação. Em outras palavras, a probabilidade de uma determinada falha ocorrer, simultaneamente, em vários provedores é menor do que a probabilidade de a mesma ocorrer em um único provedor. Desta forma,

sistemas federados podem manter sua disponibilidade, mesmo quando toda a infraestrutura de um determinado provedor se apresente indisponível.

Todavia, o monitoramento e o tratamento de falhas em sistemas federados se torna mais complexo, visto que neste tipo de ambiente há a heterogeneidade de infraestrutura, arquitetura e localização geográfica dos recursos. Além do mais, o modelo de nuvens federadas ainda herda todos os desafios referentes a tolerância a falhas, presentes no modelo de nuvem única.

3.6.1 Políticas e Técnicas de Tolerância a Falhas

Em um ambiente típico de nuvem computacional ou federação de nuvem, a tolerância a falhas trata de reparar rapidamente e substituir componentes defeituosos, objetivando manter a correta operação dos sistemas, ou seja, trata de suportar mudanças abruptas que ocorrem devido a falhas de hardware, falhas de software, falhas de rede, entre outros. As falhas mais comuns em ambientes de nuvem são as falhas por queda e as falhas arbitrárias, que normalmente ocasionam o comportamento inadequado dos sistemas [40, 65].

Conforme mencionado anteriormente no Capítulo 2, a computação em nuvem é dividade em três camadas, SaaS, PaaS e IaaS. Em relação à tolerância a falhas, uma falha que ocorre em uma das camadas pode acabar afetando os serviços oferecidos pelas outras camadas, por exemplo, uma falha que ocorre na camada PaaS pode gerar erros nos serviços oferecidos na camada de SaaS, e uma falha que ocorre na camada de IaaS pode afetar negativamente ambas as camadas, PaaS e IaaS [57]. Isto implica que o impacto de falhas na infraestrutura é significativamente alto, daí percebe-se a importância de desenvolver estratégias tolerantes a falhas também em nível de infraestrutura.

Uma grande quantidade de pesquisas e de produções literárias a respeito de tolerância a falhas em ambientes de nuvem computacional vem sendo conduzida com o objetivo de aprimorar e de criar técnicas ou estratégias de tolerância a falhas cada vez mais eficientes, isto para aumentar a dependabilidade e a segurança de sistemas hospedados neste tipo de ambiente. Com base nas principais políticas de tolerância a falhas, é possível classificá-las em duas categorias: proativa e reativa [65, 66].

Dessa forma, estratégias de tolerância a falhas incluem basicamente duas fases para o gerenciamento de falhas, a detecção: mecanismo utilizado para detectar falhas no sistema; e o reparo: transforma o estado do sistema, de um estado errôneo para um estado livre de erros [64]. Atualmente, exitem várias técnicas de tolerância que compõe as políticas citadas. A seguir, são descritas algumas das principais técnicas de tolerância a falhas existentes atualmente para sistemas que executam em ambientes de nuvens [18, 33, 57, 67, 68, 69, 70, 71]:

1. **Política Proativa**: seu princípio é evitar a recorrência de falhas, erros e defeitos, tomando medidas preventivas de maneira proativa. As medidas são tomadas com base no estudo de indicadores de falhas e previsão de falhas subjacentes. O segundo passo é aplicar proativamente medidas corretivas em tempo de implementação, alterando o código ou substituindo os componentes propensos à falha. Para a concepção desta política tolerante a falhas, algumas técnicas são necessárias, dentre elas, destacam-se:

    - **Rejuvenescimento de Software** (*Software Rejuvenation*): consiste na reinicialização periódica do sistema, a fim de trazê-lo novamente a um estado de operação limpo. Para esta técnica, suposições são feitas a respeito de quando o sistema falharia se nenhuma medida fosse tomada, e o tempo de reinicialização é dado de acordo com esta suposição. É importante ressaltar que intervalos curtos implicam em maior custo, e intervalos longos implicam em maior chance de que falhas ocorram;
    - **Auto-Reparação** (*Self-Healing*): consiste no tratamento automático de falhas de aplicações que executam em múltiplas instâncias de máquinas virtuais. Em outas palavras, a falha de uma instância do sistema é controlada e tratada automaticamente;
    - **Migração Preventiva** (*Preemptive Migration*): consiste na observação e na análise constante do sistema (do inglês, *loop-feedback*). Com base neste monitoramento, após o armazenamento de seu estado atual, o sistema é migrado antecipadamente para outro recurso;
    - **Balanceamento de Carga** (*Load Balancing*): consiste em monitorar a utilização de recursos computacionais, tais como memória, processador, disco, tráfego de rede e latência, dentre outros, e tomar ações para que a carga entre os diversos recursos computacionais seja balanceada. Por exemplo, caso a utilização de memória de um determinado recurso computacional alcance o pico de 80% do total disponível, então novas requisições para este recurso devem ser transferidas ou redirecionadas para um outro recurso, cuja utilização seja de menor intensidade.

2. **Política Reativa**: seu princípio é reduzir o impacto de falhas que efetivamente já ocorreram, fazendo com que o sistema passe de um estado instável para um estado estável, para que então volte a operar e atender suas requisições normalmente. As principais técnicas usadas são:

- **Repetição ou Tentar Novamente** (*Retry*): consiste em executar novamente a tarefa que falhou, no mesmo recurso computacional. Esta é a técnica mais simples para tolerância a falhas;
- **Reenvio de Tarefas** (*Task Resubmission*): consiste em reenviar a tarefa que apresentou falha para o mesmo ou para um outro recurso computacional. Isto ocorre em tempo de execução, sem interromper o fluxo de trabalho do sistema;
- **Ponto de Checagem/Reinício** (*Checkpointing/Restart*): consiste no armazenamento, em intervalos de tempo regulares ou irregulares, do estado global da execução de uma tarefa, para que não seja necessário executar novamente a tarefa do ponto de partida, mas sim do ponto o qual foi realizado um dos pontos de checagem, no ideal, o último. Em outras palavras, consiste em reverter o sistema para seu último estado válido, ou seja, um estado imediatamente anterior a falha. Desta forma, apenas o trabalho efetuado entre o ponto de checagem e a ocorrência da falha é perdido, caso contrário, todo o trabalho já efetuado seria perdido. Esta técnica é, geralmente, utilizada para tarefas de longa duração e a técnica pode ser utilizada por meio de dois mecanismos, os quais são [68]:
  - **Completo**: sempre grava o estado completo do sistema. Este mecanismo está associado a uma sobrecarga, pois necessita de tempo e de espaço para armazenar todas as informações necessárias.
  - **Incremental**: grava apenas as áreas que sofreram alteração. Este mecanismo ajuda a reduzir a sobrecarga do ponto de checagem completo, que grava todo o estado do sistema.
- **Replicação** (*Replication*): consiste na criação ou na manutenção de cópias do sistema em recursos diferentes. Dois tipos de replicação foram observados na literatura: (1) a replicação ativa, na qual ambos os recursos principal e secundário executam a operação independentemente uma da outra; e (2) a replicação passiva, na qual somente o recurso principal executa a operação e espera-se que o mesmo produza o resultado, enquanto os recursos secundários realizam a operação apenas em casos de falha do recurso principal [72]. Com esta técnica espera-se que ao menos uma das réplicas finalize sua tarefa com sucesso. Esta técnica adiciona redundância ao sistema, e é uma das técnicas mais utilizadas para tolerar falhas;
- **Migração de Tarefas** (*Job migration*): como o próprio nome diz, consiste em migrar tarefas que não conseguiram terminar seu trabalho para um outro

recurso computacional, a fim de possibilitar que tal tarefa sua execução, livre de falhas;

- **Checagem por Tempo** (*Timing check*): conhecida como *watchdog* é uma técnica que supervisiona periodicamente uma tarefa, isto com um tempo de função crítica;
- **Tratamento de Exceção Definido pelo Usuário** (*User defined exception handling*): está relacionada a modelar um tratamento específico para falhas. Ou seja, o próprio usuário especifica qual ação o sistema deve tomar caso uma falha seja detectada.

As políticas de tolerância a falhas, proativa e reativa, possuem vantagens e desvantagens que devem ser analisadas. A melhor escolha pode estar relacionada ao tipo de um determinado sistema. Chalermarrewong *et al.* [73] citam um experimento que evidencia a eficácia da migração de tarefas (*i.e.*, *Job migration*) sobre o ponto de checagem/reinício *(i.e.*, *Checkpointing/Restart*). Muito embora as técnicas reativas apresentem pior desempenho, ainda são utilizadas com mais frequência, pois são menos afetadas por previsões incorretas e também relativamente mais simples de se implementar. No entanto, para *clusters* de alta disponibilidade, por exemplo, políticas de tolerância a falhas reativas podem não ser apropriadas, já que falhas que efetivamente já ocorreram podem diminuir bruscamente sua disponibilidade. A Figura 3.6 ilustra as políticas e as principais técnicas de tolerância a falhas evidenciadas na literatura.

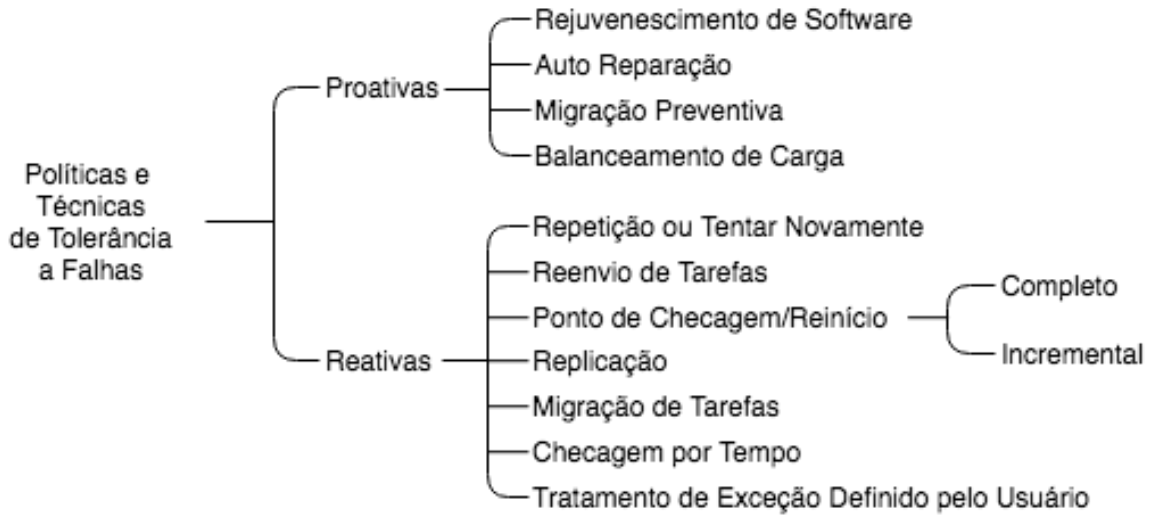

Figura 3.6: Políticas e Técnicas de Tolerância a Falhas.

Finalmente, os trabalhos relacionados encontrados na literatura são descritos na Seção 3.7.

3.7 Trabalhos Relacionados

No decorrer das duas últimas décadas uma série de artigos foram publicados com foco em enfrentar os desafios da tolerância a falhas em ambientes de nuvem computacional. Esta seção aborda alguns dos trabalhos já propostos. A Tabela 3.2 ilustra as principais características dos trabalhos relacionados.

Tabela 3.2: Trabalhos Relacionados.

Proposta	**Políticas**	**Técnicas**	**Voltado**	**Parametrizável**	**Modelo de Nuvem**
Zhao *et al.* [74]	Reativa	Replicação	Provedores	Não	Única
Zhang *et al.* [75]	Reativa	Replicação	Provedores	Não	Única
Veronese *et al.* [76]	Reativa	Replicação	Provedores	Não	Única
Garraghan *et al.* [77]	Reativa	Replicação	Consumidores	Não	Única e Federada
Jhawar *et al.* [78]	Reativa	Repetição e Migração	Provedores	Não	Única
Das *et al.* [79]	Proativa Reativa	Balanceamento de Carga, Replicação e Ponto de Checagem/Reinício	Provedores	Não	Única
Joshi *et al.* [80]	Reativa	Balanceamento de Carga e Replicação	Provedores	Não	Única
Bala *et al.* [81]	Proativa	Aprendizado de máquina	Provedores	Não	Única
Chen *et al.* [82]	Reativa	Repetição	Provedores	Não	Única

Zhao *et al.* [74] propuseram um *middleware* de tolerância a falhas de baixa latência (do inglês, Low Latency Fault Tolerance - LLFT) para tratar falhas por queda e falhas por tempo. O LLFT fornece tolerância a falhas para aplicativos distribuídos hospedados em ambientes de computação em nuvem, isto como um serviço oferecido pelo provedor. Para proteger aplicações de vários tipos de erros, o *middleware* proposto utiliza a replicação segundo uma abordagem de réplica líder e réplica seguidora. O *middleware* utiliza o protocolo de mensagens LLFT para garantir a comunicação confiável entre os processos replicados e para garantir que todo processo replicado tenha visão consistente de todo o sistema distribuído.

Zhang *et al.* [75] identificaram o problema bizantino de tolerância a falhas em nuvem de recursos voluntários, e propuseram um *framework* bizantino de tolerância a falhas, denominado BFTCloud (Byzantine Fault Tolerant Cloud). Para garantir a robustez de aplicações em nuvem, o BFTCloud utiliza técnicas de replicação para tolerar vários tipos de falhas, incluindo falhas arbitrárias. O BFTCloud é considerado, pelos autores, o primeiro *framework* tolerante a falhas arbitrárias na literatura da computação em nuvem. De acordo com os autores, após a realização de experimentos em vários tipos de ambientes de nuvem, concluiu-se que o BFTCloud, ao compor um grupo de uma réplica primária e $3f$ réplicas secundárias (*i.e.*, $3f+1$), pode garantir a robustez dos sistemas, quando até f

nós apresentam falha em tempo de execução, isto conforme Lamport *et al.* [59] já haviam dito.

No entanto, Veronese *et al.* [76] afirmam que é possível reduzir o número de réplicas para $2f + 1$, preservando as mesmas propriedades do algoritmo. Isto foi alcançado com utilização de um serviço confiável com troca de mensagem autenticadas para reduzir o número de réplicas e o número de passos necessários à comunicação entre as réplicas, o que por sua vez, reduz o custo da infraestrutura necessária para tolerar falhas bizantinas.

Garraghan *et al.* [77] desenvolveram um *framework* chamado FT-FC (Fault-Tolerance in Federated Cloud), para tolerar falhas arbitrárias em ambientes de federação de nuvens. O FT-FC também utiliza a técnica de replicação, e é constituído por uma ferramenta de agendamento automático de tarefas, um sistema de comunicação com nuvens heterogêneas e por um sistema de adjudicação, que pode enviar e receber comunicações de vários provedores de nuvens, a fim de obter consenso sobre os resultados retornados pelos provedores que potencialmente poderiam falhar.

Jhawar *et al.* [83, 84, 78] apontam um modelo para explorar a camada de virtualização, com o objetivo de oferecer, para aplicações hospedadas em uma infraestrutura de computação em nuvem, propriedades de tolerância a falhas como serviço sob demanda. Para isto, foi desenvolvido um *framework* conceitual chamado gerente de tolerância a falhas (do inglês, Fault Tolerance Manager - FTM), que é capaz de replicar e migrar instâncias de máquinas virtuais de forma transparente. O FTM funciona como um *middleware* que, por meio de monitoramento em tempo de execução de aplicações do usuário, seleciona ou combina os mecanismos de tolerância a falhas que mais adequadamente correspondem aos requisitos do usuário. O FTM se integra com infraestruturas de nuvens, e facilita para terceiros oferecer tolerância a falhas como um serviço.

Das *et al.* [79] propõem um modelo de tolerância a falhas integrado à virtualização, chamado de VFT, que utiliza ambas as políticas de tolerância falhas, proativa e reativa, isto por meio do uso das técnicas de balanceamento de carga, replicação, e ponto de verificação. As decisões tomadas nestes modelo se baseiam na taxa de sucesso de resposta e na taxa de processamento, obtidas de cada nó disponível na infraestrutura. A abordagem apresentada para a virtualização se dá devido à ajuda do *hypervisor*, no qual o balanceador de carga assume uma alta responsabilidade pela distribuição de cargas apenas para os nós virtuais, cujos servidores físicos correspondentes possuam um bom histórico de desempenho.

Joshi *et al.* [80] propõem um modelo de tolerância a falhas para lidar com falhas diretamente em servidores físicos de provedores de nuvem, migrando todas as máquinas virtuais hospedadas em um servidor que apresentou falhas, para um novo local. Objetivando reduzir o impacto da falha, a migração ocorre de forma uniforme, utilizando uma

heurística de balanceamento de carga, também proposta neste trabalho, onde as máquinas virtuais que estavam em execução no servidor que falhou são distribuídas entre os servidores físicos disponíveis na infraestrutura em questão.

Bala *et al.* [81] propuseram modelos para detecção de falhas de forma inteligente, a fim de facilitar a tolerância proativa de falhas, ao prever falhas durante a execução de aplicações de *workflows* científicos. O trabalho é distribuído em dois módulos, sendo que no primeiro módulo, falhas de tarefas são previstas com abordagens de aprendizado de máquina. Já no segundo módulo, falhas reais são localizadas após a execução do *workflow*. Abordagens de aprendizagem de máquina como *Naive Bayes* [85], redes neurais artificiais, regressão logística e floresta aleatória são implementadas para prever as falhas de forma inteligente, isto a partir de um conjunto de dados ou parâmetros referentes a utilização de recursos, como utilização de processador, memória, disco e utilização de largura de banda.

Chen *et al.* [82] propõem três métodos de clusterização dinâmica, baseados na técnica de tolerância a falhas *Retry*, para melhorar a tolerância a falhas no *cluster* de tarefas de *workflows* científicos. O primeiro método repete tarefas que falharam em uma execução. O segundo método ajusta dinamicamente o tamanho da granularidade do *cluster* (número de tarefas em um *cluster*) de acordo com o tempo estimado entre a ocorrência de falhas. O terceiro método repete a execução após dividir os agrupamentos de tarefas, reduzindo a granularidade do *cluster*.

3.8 Considerações Finais

Neste capítulo foram abordados diversos aspectos referentes a tolerância a falhas, dentre eles, foram apresentadas as principais técnicas existentes para garantir o correto funcionamento de sistemas mesmo sobre a presença de falhas. A computação em ambientes de nuvens federadas, além de herdar desafios de sistemas distribuídos e de sistemas do modelo de nuvem única, também apresenta muitos desafios próprios, como a heterogeneidade de infraestrutura, arquitetura, localização geográfica, entre outros.

Para enfrentar esses desafios e favorecer a dependabilidade e a segurança de sistemas federados, é crucial que técnicas de tolerância a falhas sejam utilizadas, pois um sistema deve ser capaz de tolerar falhas e permanecer com sua correta operação mesmo em ambientes de nuvem federadas. Assim sendo, a tolerância a falhas foi o item escolhido para ser tratado neste trabalho.

Como análise experimental, o modelo proposto será integrado ao sistema BioNimbuZ, uma plataforma de federação de nuvens computacionais híbridas para a execução de

workflows de Bioinformática, com o objetivo de fornecer tolerância a falhas a todas as suas camadas. Mais detalhes sobre a plataforma BioNimbuZ são apresentados no Capítulo 4.

Capítulo 4

Plataforma de Federação BioNimbuZ

O BioNimbuZ foi originalmente proposto por Saldanha [17], tendo como principal objetivo permitir a execução de *workflows* científicos de Bioinformática em ambientes de nuvem. Com o passar dos anos, várias evoluções foram feitas, os quais estão fundamentadas nos trabalhos [21, 86, 87, 88, 89, 90, 91, 92]. Em consequência do surgimento de novas tecnologias que ajudam no melhor aproveitamento dos ambientes de nuvens federadas, as evoluções executadas no BioNimbuZ modificaram sua arquitetura com o objetivo de implementar e utilizar tecnologias que o ajudassem a prover melhor os seus serviços [93].

Atualmente, como uma plataforma de nuvens federadas, o BioNimbuZ permite a interação entre diversos tipos de nuvens, privadas, públicas e comunitárias, oferecendo aos seus usuários a ideia de que dispõem de uma infinidade de recursos virtuais. Isto de forma transparente ao usuário, que não necessita se preocupar com os detalhes de infraestrutura de cada provedor utilizado pela plataforma BioNimbuZ [21].

A plataforma de federação, aqui apresentada, possui como uma importante característica, a flexibilidade para a inclusão de mais provedores de nuvem, pois o BioNimbuZ possui uma camada de integração formada por *plugins* que são responsáveis por mapear as requisições de usuários para um determinado provedor. Dessa forma, é fácil incluir provedores à federação de nuvens utilizada pela plataforma, e assim, a plataforma BioNimbuZ atende a requisitos de escalabilidade e de flexibilidade.

Inicialmente, o BioNimbuZ foi proposto como um sistema P2P (*peer-to-peer*) [94]. Em outras palavras, um sistema distribuído sem qualquer controle centralizado ou organização hierárquica. No entanto, com a evolução da plataforma, esta característica foi modificada, e atualmente o BioNimbuZ mantém informações distribuídos com o uso do serviço de coordenação Apache ZooKeeper[1]. Desta forma, a consulta, a atualização e a sincronização de dados referentes à federação podem ser executadas de maneira mais simples, pois o Apache ZooKeeper permite um controle centralizado de todo o sistema distribuído.

[1]Apache ZooKeeper, `https://zookeeper.apache.org/`

Posteriormente, evoluções contribuíram com a adoção de algoritmos de escalonamento de tarefas, que elegem as melhores instâncias virtuais disponíveis para a execução de tarefas dos *workflows*, visando a redução de custos [92, 95]; o uso de políticas de armazenamento que consideram a localidade dos dados de entrada, custo, latência de rede e a utilização de recursos gratuitos [89, 96, 97]; a implementação de um controlador de elasticidade, que oferece suporte ao provisionamento/desprovisionamento automático e sob demanda de máquinas virtuais, bem como a elasticidade horizontal (quantidade de recursos alocados) e vertical (capacidade de processamento e quantidade de memória) de recursos alocados pela plataforma [88]; e com um controlador que garante a não violação de SLA estabelecidos entre a plataforma BioNimbuZ e o usuário final [21].

Recentemente, com a necessidade de otimizar o armazenamento de arquivos, a plataforma BioNimbuZ teve seu serviço de armazenamento modificado, possibilitando a utilização de serviços de armazenamento mais recentes, como *Buckets*[23], também oferecidos por diversos provedores de nuvem computacional [98].

O BioNimbuZ, atualmente, apresenta-se com uma nova arquitetura, na qual é possível afirmar que que todos os seus serviços foram implementados de maneira mais modularizada, o que deu origem a uma segunda versão da plataforma [93]. A principal mudança foi quebrar a camada de núcleo em várias camadas, que em sua versão inicial consistia em um monobloco, responsável por toda a execução de *workflows*. Com isto, a nova arquitetura permite maior flexibilidade na execução de tarefas e tem como foco reduzir os custos necessários para a execução das mesmas. Uma outra grande mudança foi a utilização de microsserviços que possibilita o desacoplamento da camada responsável por abrir canal de comunicação com os diversos provedores de nuvem. A nova arquitetura também favorece a manutenibilidade dos sistema, pois agora é possível que a plataforma sofra modificações de forma independente em cada uma de suas camadas. A atual arquitetura do BioNimbuZ é detalhada na Seção 4.1.

4.1 Arquitetura do BioNimbuZ

A arquitetura do BioNimbuZ foi projetada de maneira hierárquica e distribuída, a fim de facilitar a distribuição dos seus componentes em diferentes provedores de nuvem computacional. Além disso, ela possui camadas bem divididas, o que permite maior facilidade de implantação de seu ambiente, e de integração de seus componentes na federação de nuvens. Outro fator que contribui para essa facilidade é a criação de *plugins* de nuvem

[2]Amazon S3 Buckets, `http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html`
[3]Google Cloud Storage Buckets, `https://cloud.google.com/compute/docs/disks/gcs-buckets`

como microsserviços [99], que são totalmente independentes do funcionamento global da plataforma, e podem ser encerrados ou iniciados sempre que necessário.

A comunicação entre os componentes é feita em dois níveis, uma em alto nível e a outra em baixo nível. A primeira, em alto nível, é feita entre as nuvens, permitindo uma troca de informação mais resumida. A outra, em mais baixo nível, é feita dentro da nuvem, trafegando informações de monitoramento das tarefas em execução, que podem executar em paralelo. Desta forma, a plataforma possibilita uma atualização de informações em tempo real aos seus múltiplos usuários.

Outra característica importante da arquitetura é a possibilidade de utilização de credenciais de nuvens de seus usuários, e o seu compartilhamento com grupos de usuários, permitindo que organizações, como as universidades, criem grupos de estudantes com uma mesma credencial. Assim, a tarifação é feita diretamente pelos provedores de nuvem, sem que a plataforma de federação tenha a responsabilidade sobre o controle e o processo de cobrança de cada usuário. Além do mais, seus usuários podem acessar a plataforma de maneira simplificada com o Serviço Web disponibilizado.

Para isso, a arquitetura foi estruturada em quatro camadas, as quais são: a Camada de Aplicação, que fornece mecanismos para que os usuários possam interagir com a plataforma; a Camada de Federação, com mecanismos acopláveis à plataforma que permitem a criação da federação de nuvens; a Camada de Coordenação, responsável por coordenar o fluxo e o monitoramento de execução das tarefas e, por fim, a Camada de Execução, que é responsável por efetivamente processar e monitorar as tarefas.

É importante ressaltar que na atual arquitetura, cada camada é formada por controladores, serviços e mecanismos, os quais são distribuídos de maneira a obter um melhor aproveitamento de recursos computacionais. Na Figura 4.1 é possível ter uma visão geral das camadas da arquitetura proposta. As Seções 4.2, 4.3, 4.4 e 4.5 descrevem cada uma dessas camadas em detalhes.

4.2 Camada de Aplicação

Esta camada é responsável por possibilitar a interação dos usuários com o sistema, por meio de interface web. A interface permite aos usuários fazerem o registro na plataforma e manterem suas credenciais de provedores de nuvem. Também é possível aos usuários executar aplicações e criar dependências entre elas (por exemplo, as de *workflows* científicos), e enviarem dados de entrada/saída para os provedores de nuvem cadastrados.

Assim, os dados fornecidos (registros, credenciais, tarefas, dependências, metadados dos dados de entrada/saída, dentre outros) são mantidos em banco de dados e acessado

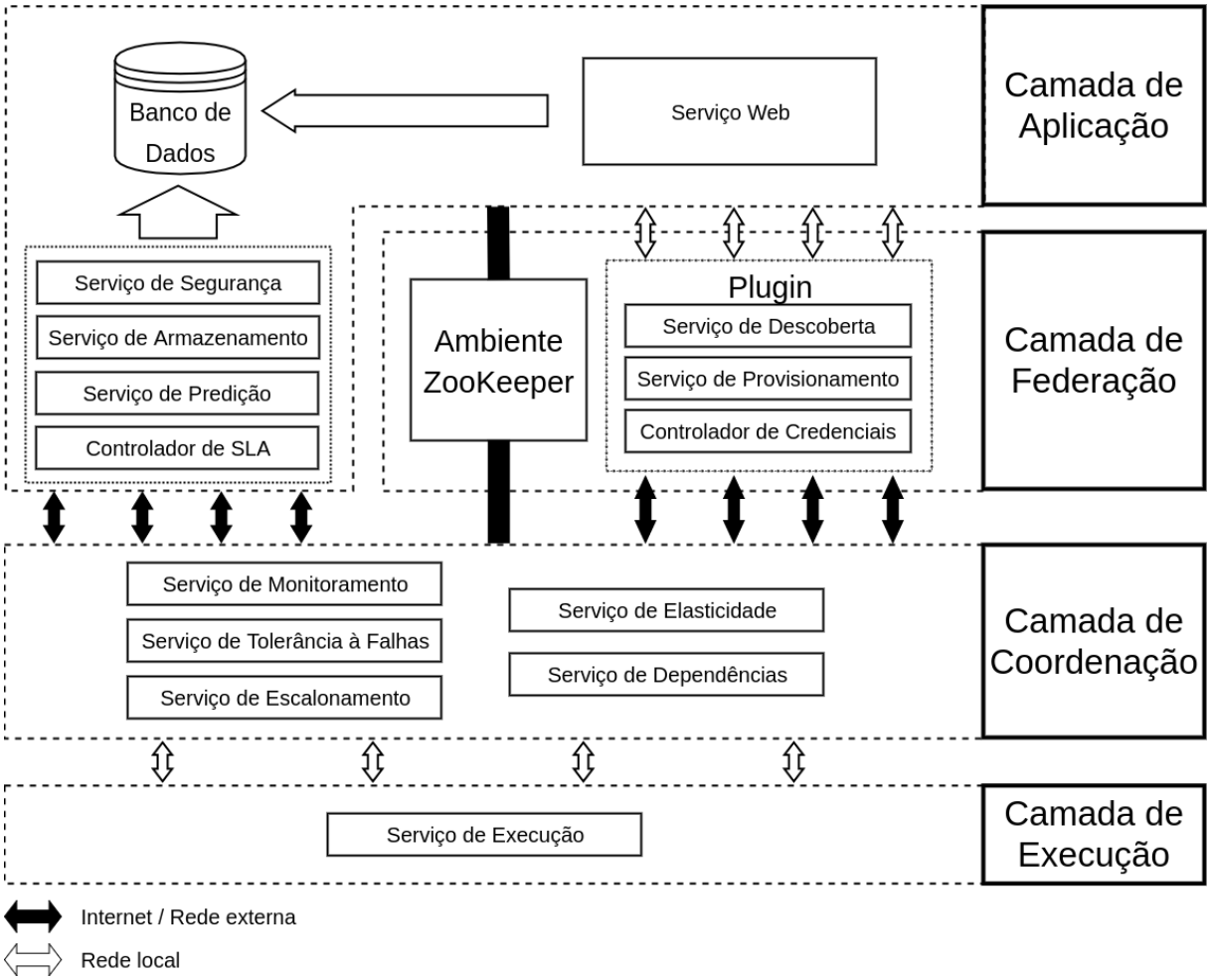

Figura 4.1: Arquitetura do BioNimbuZ Versão 2.

pelos módulos que compõem essa camada. Os serviços e os controladores dessa camada são os seguintes:

- **Serviço Web**: responsável por prover interface gráfica que permita a interação entre usuários e a plataforma por meio de páginas web. Por meio deste serviço os usuários podem acessar a plataforma e manter suas informações no banco de dados;
- **Serviço de Segurança**: encarregado por garantir os princípios básicos da segurança da informação, a confidencialidade, a integridade e a disponibilidade de informações. Em outras palavras, este serviço é o responsável por: (1) manter o sigilo de informações, impedindo que pessoas não autorizadas tenham acesso a informações sigilosas; (2) proteger as informações submetidas por usuários, bem como as informações geradas pelos *workflows*, de modo a impedir mudanças em seu conteúdo, sejam elas acidentais, indevidas ou até mesmo intencionais (por exemplo, alterações indevidas no conteúdo de arquivos de entrada/saída); e (3) garantir que as

informações sempre estajam disponíveis para pessoas autorizadas, a qualquer momento. Além disto, este serviço também é responsável por armazenar os dados de autenticação dos usuários na plataforma e as credenciais de acesso aos serviços de nuvem disponíveis. Este serviço viabiliza o acesso de múltiplos usuários à plataforma, garantindo que os dados estejam isolados entre os cientistas e suas equipes (credenciais, *workflows*, resultados, dentre outros);

- **Serviço de Armazenamento**: incumbido por implementar estratégias de armazenamento de arquivos utilizados pelos *workflows*, sejam eles arquivos de entrada ou arquivos de saída. Para isto, o serviço analisa fatores como a latência de rede, a largura de banda, a localização geográfica entre os provedores e a capacidade de armazenamento disponível em cada recurso virtual. O Serviço de Armazenamento trabalha com a estratégia de armazenamento de arquivos fornecida pelos provedores de nuvem (por exemplo, *Buckets*);

- **Serviço de Predição**: tem como objetivo auxiliar o usuário final na escolha dos recursos necessários à execução de seu *workflow*, com estimativas de tempo, custo e recursos computacionais. Para isto, o usuário deve preencher, via Serviço Web, um formulário com informações de seu *workflow*, para que o serviço efetue as estimativas mencionadas. Este serviço é constituído de quatro fases: (1) coleta de informações, na qual o usuário informa os parâmetros desejados para a execução do *workflow* (por exemplo, capacidade desejada para processamento, memória e disco); (2) avaliação da base histórica de execução de *workflows*, para otimizar a acurácia no calculo das estimativas, a base histórica contém informações de execuções passadas de todos os *workflows* já submetidos à plataforma BioNimbuZ; (3) aplicação do algoritmo GRASP [100], a fim de obter boas soluções, em tempo viável, através da aplicação de metaheurísticas baseadas em informações coletadas de fases anteriores; e por fim, (4) armazenar e informar ao usuário os dados das predições obtidas com a aplicação das três fases anteriores. As plataformas de nuvem consideradas na predição para a execução das tarefas serão apenas as relacionadas às credenciais fornecidas pelos usuários, não necessitando considerar todas as plataformas suportadas pelo sistema. Assim, se o usuário cadastrar apenas a credencial do provedor *Google Cloud Platform*, sua aplicação somente poderá executar neste provedor. Todavia, se o usuário fornecer credenciais de ambos os provedores, *Google Cloud Platform* e *Amazon Web Services*, a plataforma de federação, juntamente com as decisões deste serviço, poderá distribuir a execução das tarefas do usuário em ambos os provedores;

- **Controlador de SLA**: encarregado por implementar o ciclo de vida de SLA, o qual é formado por seis atividades: (1) descoberta de provedores, (2) definição

de SLA, (3) estabelecimento do acordo de serviço, (4) monitoramento de violação do acordo, (5) término de acordo e (6) aplicação de penalidades por violação do acordo. Para isto, o usuário deve configurar o modelo de SLA desejado. O modelo é constituído por requisitos funcionais, tais como número de núcleos do processador, tamanho de memória, tamanho de armazenamento e requisitos não funcionais, como custo máximo para a execução de um *workflow*, bem como a taxa mínima para transferência de dados durante a execução de um *workflow*. Para as requisições dos usuários, o Controlador de SLA verifica junto aos provedores de nuvem participantes da federação, se eles podem ou não suportar, naquele momento, os parâmetros configurados pelo usuário. Esta verificação é feita por meio da utilização dos *plugins*, responsáveis pela comunicação com os provedores.

4.3 Camada de Federação

A Camada de Federação é responsável por garantir a integração dos diferentes componentes do sistema que podem estar situados em diversos provedores de nuvem, formando uma federação de nuvens. A federação é obtida pela utilização do ambiente ZooKeeper [101] e dos *Plugins* que implementam as funcionalidades inerentes a determinado provedor de nuvem.

Com o ambiente fornecido pelo ZooKeeper, são mantidos dados de forma distribuída entre os Coordenadores e a Camada de Aplicação, de maneira que possam ser consultados e acompanhados em tempo real. Tais dados são, por exemplo, o estado de execução das tarefas, os endereços das máquinas monitoradas pelos Coordenadores, o consumo de recursos e os metadados dos arquivos de entrada/saída das tarefas.

Os *Plugins* são implementados de acordo com a abordagem de microsserviços [99], na qual cada serviço possui pequenas responsabilidades, cada um rodando em seu próprio processo, comunicando-se por meio de mecanismos leves, geralmente uma API (*Application Programming Interface*) sobre o protocolo HTTP[4].

Dessa maneira, independentes da Camada de Aplicação, os *Plugins* fornecem uma interface comum de acesso aos provedores de nuvem, e são compostos pelos seguintes serviços e controladores:

- **Serviço de Descoberta**: é incumbido de compilar informações de latência de rede, tabelas de preços e tipos de recursos de computação e de armazenamento de determinada plataforma de nuvem. As informações são mantidas sempre em memória para uma consulta mais eficiente pela Camada de Aplicação, visto que esses dados não são sensíveis aos usuários;

[4]Protocolo HTTP, `https://www.w3.org/Protocols/`

- **Serviço de Provisionamento**: fornece para a Camada de Aplicação a possibilidade de alocar e desalocar máquinas de determinado provedor de nuvem. Também é responsável por provisionar espaços para *upload*/*download* de arquivos de entrada/saída das tarefas definidas na Camada de Aplicação;
- **Controlador de Credenciais**: é responsável por interpretar as informações de credenciais enviadas pela Camada de Aplicação por meio dos arquivos JSON[5], na forma de chave-valor, e traduzir para o mecanismo de segurança utilizado pela nuvem sob seu controle.

4.4 Camada de Coordenação

Esta camada e seus serviços são implementados pelos Coordenadores de Tarefas (*Task Coordinator* - TC), que são responsáveis por monitorar e garantir o fluxo das tarefas criadas pelos usuários, bem como as dependências existentes entre elas.

O TC é criado com as credenciais dos usuários ou com as credenciais do grupo de usuários, que são repassadas para os *Plugins*. Assim, o usuário deverá estar ciente de que para utilizar a plataforma, o custo total para a execução de suas tarefas (alocação de máquinas virtuais e espaços de armazenamento) também inclui o custo para a execução de seus Coordenadores. Esse é um custo extra para o usuário, mas que fornece benefícios como a execução mais eficiente de suas tarefas com serviços de elasticidade, provisionamento e monitoramento. Os serviços que compõem esta camada são os que seguem:

- **Serviço de Monitoramento**: responsável por monitorar o consumo de recursos computacionais utilizados por seu Coordenador, e o consumo utilizado pelos Executores de Tarefas. Esse monitoramento não servirá para a tarifação dos usuários, mas para o monitoramento e garantia de SLA. O serviço de monitoramento estará sempre em contato com os Executores de Tarefas (os quais serão detalhados na Seção 4.5), coletando os dados de execução levantados por estes;
- **Serviço de Tolerância a Falhas**: em conjunto com o Serviço de Monitoramento, este serviço detecta o término das tarefas e realiza a replicação dos resultados gerados, garantindo que sempre existam, ao menos, duas cópias desses resultados. Ao constatar que algum resultado ainda não foi duplicado ou suas cópias encontrarem-se indisponíveis, este serviço dá início ao processo de replicação;
- **Serviço de Dependências**: este serviço verifica as demandas das tarefas e suas dependências, quais as próximas tarefas a serem executadas, e se determinado lote

[5]JSON, `http://www.json.org/`

de tarefas foi concluído ou não. Além disso, este serviço também verifica se o TC necessita ser encerrado para economizar o custo;

- **Serviço de Elasticidade**: com o auxílio do Serviço de Monitoramento, este serviço analisa as informações coletadas pelos Executores de Tarefas. Desta forma, é capaz decidir quando aumentar ou reduzir os recursos computacionais (quantidade de máquinas virtuais, tamanho de memória, quantidade de processadores e capacidade de armazenamento);
- **Serviço de Escalonamento**: responsável por distribuir as tarefas que devem ser executadas na nuvem em que o TC está executando, e solicitar o início da execução das tarefas ou o lote delas.

4.5 Camada de Execução

Os Executores de Tarefas (*Task Executor* - TE) são os processos que representam esta camada e fornecem o Serviço de Execução. Assim, o Serviço de Execução é o responsável por manter o ciclo de vida das tarefas e o monitoramento das mesmas.

O TE é responsável por garantir a comunicação constante com os TCs, enviando dados de monitoramento, como o consumo de processador, o consumo de memória, e o estado de execução da tarefa. Em outras palavras, esta camada é responsável por executar e monitorar as tarefas que constituem os *workflows* definidos na Camada de Aplicação.

4.6 Fluxo de Execução Principal

A fim de facilitar o entendimento da integração entre as camadas da arquitetura do BioNimbuZ, esta seção apresenta a execução das tarefas em ambiente federado. O fluxo de execução é realizado de acordo com os seguintes passos:

1. **Acesso Web**: os usuários inicialmente acessam a plataforma web, por meio da utilização de um navegador de Internet, e solicitam seu ingresso. Em seguida, os usuários efetuam *login* na plataforma, onde serão capazes de cadastrar as informações necessárias para a execução de suas tarefas;
2. **Cadastro de Credenciais de Nuvem**: após o *login*, os usuários devem inserir suas credenciais de nuvem que serão utilizadas pelos *Plugins* das nuvens cadastradas, para que seja possível a execução de suas tarefas nesses provedores de nuvem;
3. **Criação de Tarefas**: posteriormente ao cadastro de credenciais das nuvens suportadas, os usuários podem dar início à criação de seus *workflows*. Além disso, podem

efetuar o *upload* de arquivos e a criação de grupos de usuários. Antes do cadastro de credenciais não é possível efetuar a criação de tarefas ou o *upload* de arquivos, pois esse processo consome recursos, que serão custeados pelos próprios usuários. Após a criação das tarefas, e com o auxílio do Serviço de Predição ou escolha direta de máquinas feita pelos usuários, são definidos os passos para a execução das tarefas com as dependências existentes entre si, bem como as nuvens a serem utilizadas. As tarefas criadas são armazenadas em banco de dados para posteriores consultas, e também são armazenadas no ambiente ZooKeeper, que só manterá essas informações enquanto as tarefas estiverem em execução;

4. **Coordenação de Tarefas**: definidas as nuvens iniciais que vão executar as tarefas, a Camada de Aplicação solicita aos *Plugins* respectivos, a criação dos TCs. Estes recebem informações para ingressarem no ambiente ZooKeeper. Desta forma, ao iniciarem, consultam as tarefas pendentes de execução, e solicitam a criação dos respectivos TEs;

5. **Monitoramento de Tarefas**: os TCs estão sempre sendo atualizados pelos TEs em determinado intervalo de tempo. Esse mecanismo será necessário para detectar quedas, e para que o Serviço de Elasticidade identifique a necessidade do aumento ou da redução de recursos;

6. **Término de Execução**: por fim, a Camada de Aplicação é notificada quando há o término, seja ele com sucesso ou não, de tarefas em execução por meio de trocas de mensagens. Os dados gerados pela execução das tarefas (tempo total de execução e localização dos arquivos de saída) são armazenados em banco para posterior consulta.

A Figura 4.2 ilustra a relação existente entre os componentes integrantes da arquitetura. Assim, como pode ser observado em (1), os usuários acessam a plataforma pelo navegador e solicitam seu ingresso. Depois que possuem acesso, (2) são capazes de cadastrar suas credenciais de nuvem, suas tarefas e grupos em banco de dados. Em seguida, (3) a Camada de Aplicação solicita que o *Plugin* da nuvem, na qual a tarefa será executada, faça o provisionamento de recursos necessários em (4). Depois da inicialização dos recursos virtuais, a Camada de Aplicação envia informações em (5) para conexão com o ambiente ZooKeeper para os Coordenadores, que fazem a conexão com o ambiente em (6). Por fim, em (7) é estabelecida a comunicação dos Executores com seus Coordenadores. Caso um lote de tarefas seja executado em nuvens diferentes, os Coordenadores das nuvens estabelecem comunicação em (8) para sincronização do fluxo de execução.

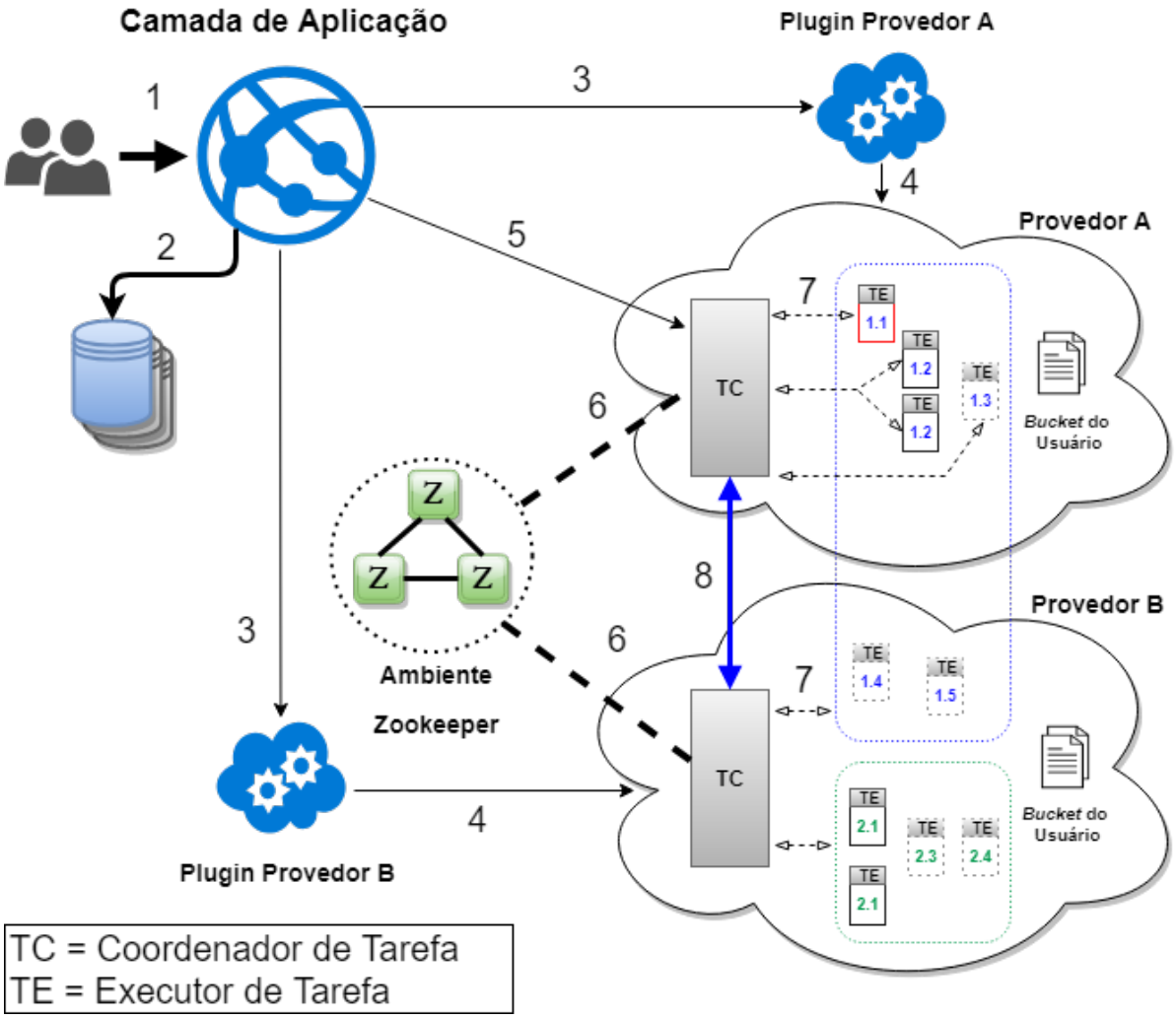

Figura 4.2: Comunicação entre os Diferentes Componentes.

4.7 Considerações Finais

Este capítulo discorreu sobre o sistema BioNimbuZ, que é uma plataforma de federação de nuvens computacionais híbridas para a execução de *workflows* de Bioinformática, tratando de descrever a atual arquitetura da plataforma, bem como cada uma de suas camadas. Também foi ilustrado como é o funcionamento do fluxo de execução principal da plataforma.

Conforme foi apresentado neste capítulo, a plataforma BioNimbuZ já possui um serviço responsável pela "tolerância a falhas" em sua arquitetura. Todavia, o serviço hoje existente apenas implementa o princípio básico da tolerância a falhas (*i.e.*, redundância de dados), e ainda o aplica apenas para os arquivos de saída relacionados à execução de *workflows*. O atual serviço de tolerância a falhas deixa de fora o controle da própria plataforma, ou seja, se uma falha por queda ocorre na Camada de Aplicação, por exemplo, todo o serviço se

tornaria indisponível aos usuários da plataforma, o que o torna incapaz de tolerar falhas e permanecer com sua correta operação. Esta deficiência justifica o motivo pelo qual a plataforma BioNimbuZ foi escolhida para a integração do modelo de tolerância a falhas proposto. O Capítulo 5 apresenta o modelo de tolerância a falhas proposto neste trabalho.

Capítulo 5

Modelo Proposto para Tolerância a Falhas

Neste capítulo será apresentado o modelo proposto neste trabalho, para tolerância a falhas em ambientes distribuídos, o qual é um modelo de tolerância a falhas que funciona de maneira integrável a qualquer tipo de sistema, ou seja, um mecanismo implementado como biblioteca que pode ser integrado em sistemas centralizados ou em sistemas distribuídos.

5.1 Modelo Proposto

O modelo proposto tem como principal objetivo facilitar a utilização de políticas e de técnicas de tolerância a falhas para tratamento de falhas por queda e de falhas por tempo em aplicações que fazem uso de ambientes de nuvens computacionais. Visto que a utilização de tolerância a falhas neste tipo de ambiente está diretamente relacionada a custos, o modelo proposto também tem como objetivo permitir que usuários finais possam escolher e configurar (*i.e.*, tolerância a falhas parametrizável pelo usuário) um nível de tolerância a falhas que atenda as suas necessidades de acordo com o seu negócio.

Para avaliar o modelo será realizado uma análise experimental com injeção de falhas, objetivando salientar, de acordo com as métricas de tolerância a falhas [57], a eficácia e o *overhead* do modelo proposto.

Para explorar e avaliar a melhoria da confiabilidade da tolerância a falhas, a estrutura do modelo foi constituída por vários componentes. A Figura 5.1 ilustra os componentes e os relacionamentos do modelo proposto.

Na arquitetura do modelo de tolerância a falhas proposto, o *Plugin* de Tolerância a Falhas (*FTPlugin*), corresponde à biblioteca que será integrada em aplicações clientes, com o objetivo de fornecer uma interface de interação entre o coordenador de tolerância a falhas e a aplicação cliente. O *FTPlugin* é composto por dois serviços, que são:

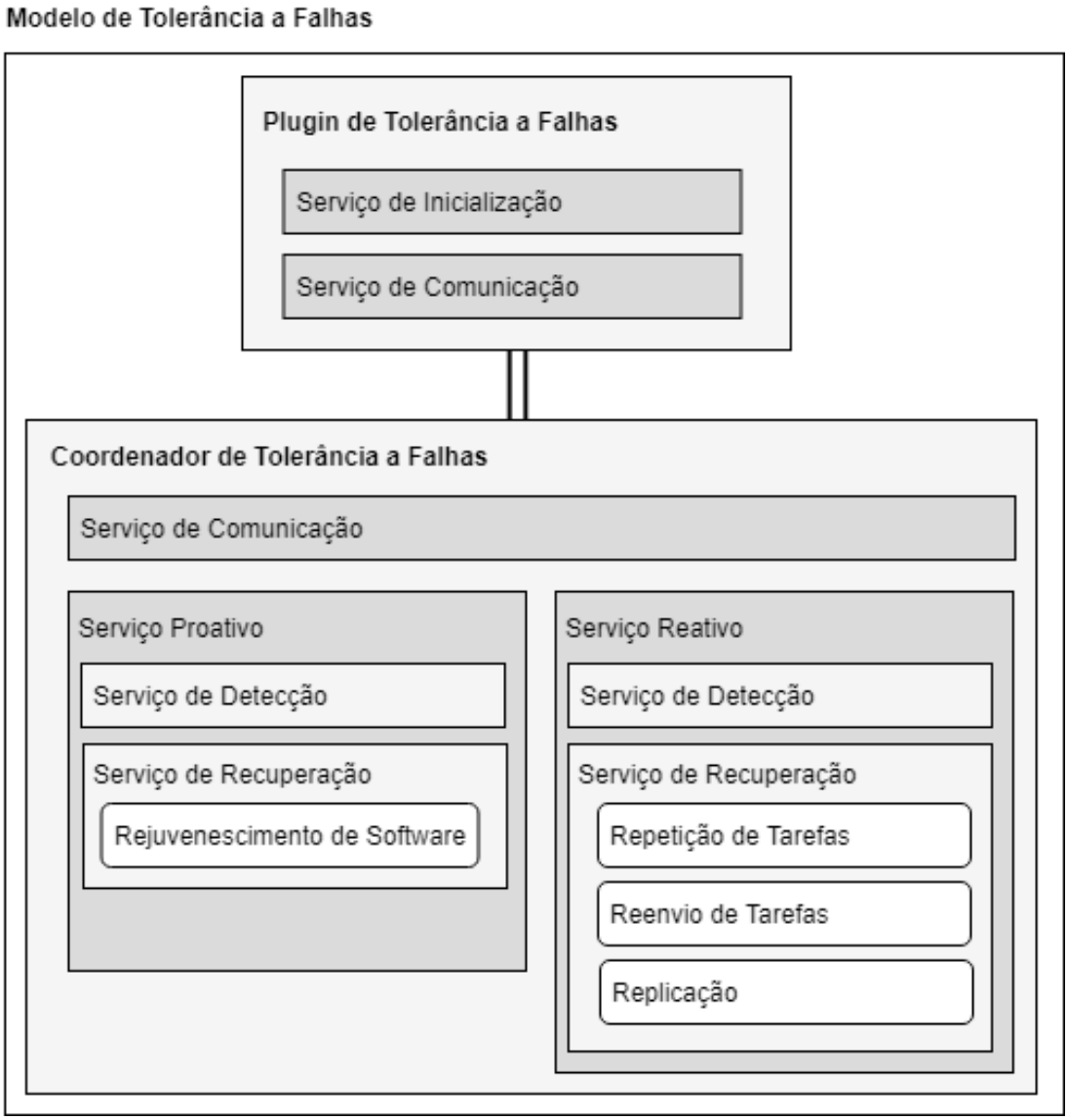

Figura 5.1: Arquitetura do Modelo de Tolerância a Falhas Proposto.

- **Serviço de Inicialização** (*BootstrapService*): responsável por (1) receber e validar os parâmetros que configuram o nível de tolerância a falhas desejado pela aplicação cliente; (2) configurar e inicializar o serviço de comunicação; e (3) criar o coordenador de tolerância a falhas, caso os parâmetros recebidos sejam válidos e o serviço de comunicação tenha sido iniciado com sucesso.
- **Serviço de Comunicação** (*CommServiceThread*): responsável por estabelecer o canal de comunicação entre o *FTPlugin* e o coordenador de tolerância a falhas. A troca de mensagens oferecida por este serviço ocorre por meio da utilização de notação JSON sobre o protocolo HTTP, isto segundo o estilo arquitetural *Restful Web Services* [102].

Já o Coordenador de Tolerância a Falhas (*FTCoordinator*) é responsável por configurar, monitorar e aplicar o nível de tolerância a falhas solicitado pela aplicação cliente. O *FTCoordinator* corresponde a um aplicativo independente do *FTPlugin*, ou seja, se trata de um processo totalmente apartado, tanto da aplicação cliente, quanto do *FTPlugin* que lhe deu origem. Para cumprir com seus objetivos, o *FTCoordinator* é composto pelos serviços:

- **Serviço de Comunicação** (*CommServiceController*): responsável por possibilitar a troca de mensagens entre o *FTPlugin* e o seu coordenador. Ele é similar ao serviço de comunicação do *FTPlugin*, ou seja, possui as mesmas características;
- **Serviço de Tolerância a Falhas Proativo** (*PFTService*): encarregado por iniciar e configurar os serviços responsáveis pela detecção e aplicação da técnica de tolerância a falhas de rejuvenescimento de software;
- **Serviço de Tolerância a Falhas Reativo** (*RFTService*): encarregado por iniciar e configurar os serviços responsáveis pela detecção e aplicação das técnicas de tolerância a falhas: repetição, reenvio de tarefas e replicação;
- **Serviço de Detecção de Falhas**: este serviço é implementado separadamente pelos serviços *PFTService* e *RFTService*. Em casos de configuração da política proativa, é incumbido de inspecionar periodicamente o nível de utilização de processador e de memória da instância monitorada, e quando verificado que os níveis definidos na parametrização foram excedidos, ou quando o *timeout* configurado for atingido, aplicar a técnica de rejuvenescimento de software por meio do serviço de recuperação. Caso nenhum dos atributos necessários à aplicação desta técnica sejam informados, entende-se que a aplicação cliente não deseja a utilização da mesma. Já em casos de configuração da política reativa, é responsável por monitorar o *FTPlugin* por meio da utilização do protocolo de *heartbeat* adaptativo, no qual o tempo de *timeout* é recalculado com base no tempo de resposta de uma determinada instância de máquina virtual, ou seja, com base na latência. Este protocolo será aplicado em um nível, tendo como referência o tempo necessário para comunicação direta com o *FTPlugin*. Caso nenhum dos atributos necessários à aplicação destas técnicas seja informado, entende-se que a aplicação cliente não deseja a utilização das mesmas;
- **Serviço de Recuperação** (*RecoveryService*): é encarregado por aplicar as técnicas implementadas de tolerância a falhas proativas e reativas, são elas, o rejuvenescimento de software, a repetição, o reenvio de tarefas e a replicação, isto de acordo com a parametrização recebida da aplicação cliente.

Para configurar o nível de tolerância a falhas é necessário criar entidades de negócio, responsáveis por armazenar as informações que configuram o nível desejado pela aplicação cliente. A Figura 5.2 mostra os detalhes de como as entidades de negócio se relacionam, em um diagrama de classes, com o objetivo de configurar o nível de tolerância a falhas. Os métodos *get/set* foram suprimidos neste diagrama para simplificar e facilitar o entendimento.

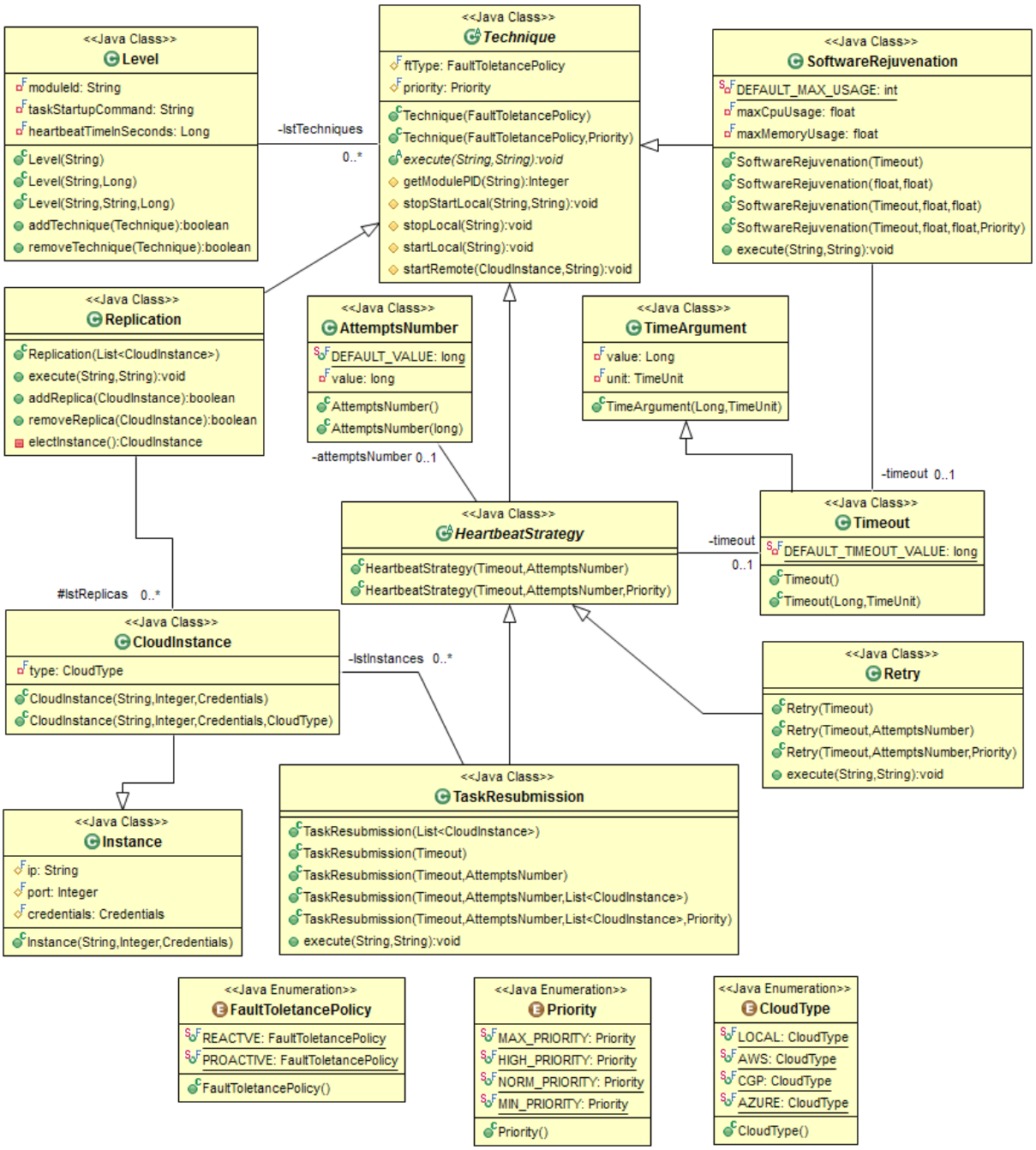

Figura 5.2: Diagrama de Classes da Configuração do Nível de Tolerância a Falhas.

Assim, as principais entidades de negócio e seus atributos ou propriedades são descritas a seguir:

- **Nível de tolerância a falhas** (*Level*): entidade de negócio responsável por armazenar informações de configuração do nível de tolerância a falhas desejado. Os seus atributos são os que seguem:
  - *moduleId*: corresponde ao identificador do processo (PID) que necessita ser monitorado;
  - *taskStartupCommand*: responsável por armazenar a linha de comando necessária à inicialização da tarefa monitorada;
  - *heartbeatTimeInSeconds*: corresponde ao tempo em segundos que o serviço de detecção deve utilizar para monitorar a aplicação cliente por meio do protocolo de *heartbeat*. Esta propriedade é necessária apenas quando configuradas as técnicas de repetição e reenvio de tarefas;
  - *lstTechniques*: é o conjunto ou lista de técnicas desejadas pela aplicação cliente.
- **Técnica de tolerância a falhas** (*Technique*): entidade de negócio responsável por armazenar informações da técnica desejada. Esta entidade possui como classes filhas as entidades *SoftwareRejuvenation*, *Retry*, *TaskResubission* e *Replication*. Os seus parâmetros ou atributos são os que seguem:
  - *ftType*: armazena o tipo da política de tolerância a falhas relacionada à técnica configurada;
  - *priority*: propriedade que define qual técnica, dentre a repetição e o reenvio de tarefas, tem maior prioridade, no caso de as duas serem configuradas simultaneamente. Para este atributo, valores menores indicam maior prioridade;
  - *maxCpuUsage*: define o valor percentual máximo para o consumo de processador para a aplicação da técnica de rejuvenescimento de software;
  - *maxMemoryUsage*: define o valor percentual máximo para o consumo de memória para a aplicação da técnica de rejuvenescimento de software;
  - *DEFAULT_MAX_USAGE*: percentual padrão, com o valor de 95%, para o consumo de memória e de processador para a aplicação da técnica de rejuvenescimento de software. Esta propriedade é utilizada apenas quando a aplicação cliente não informa os limites desejados para o consumo de processador e de memória, mas ainda assim deseja a aplicação da técnica em questão;

– *timeout*: propriedade utilizada pelas técnicas de rejuvenescimento de software, repetição e reenvio de tarefas. Para a técnica de rejuvenescimento de software esta propriedade define o tempo máximo que o PFTService deve aguardar para aplicar a técnica, isto independente dos níveis de utilização de processador e de memória. Já para as técnicas de repetição e de reenvio de tarefas, esta propriedade define a tolerância para que o serviço de detecção acuse uma falha, ou seja, uma falha por tempo ou uma falha por queda será detectada quando o tempo sem comunicação com o *FTPlugin* for maior que o valor da propriedade *heartbeatTimeInSeconds* somada ao valor definido para *timeout* somados a latência;

– *lstInstances*: corresponde ao conjunto ou lista de instâncias virtuais disponíveis para a aplicação das técnicas de reenvio de tarefas e replicação. A entidade de negócio referente às instâncias de máquinas virtuais possuem os atributos IP, porta e credenciais de acesso.

Uma outra característica do modelo de tolerância a falhas proposto é possibilitar o consumo de recursos computacionais sob demanda, mesmo para a execução das técnicas de tolerância a falhas. Em outras palavras, a execução das técnicas de tolerância a falhas configuradas pela aplicação cliente, não depende da existência de instâncias virtuais disponíveis, ou seja, o atributo "conjunto de instâncias disponíveis" não é mandatório. Caso seja verificado no *FTCoordinator* que não existem instâncias disponíveis à execução das técnicas, uma mensagem de solicitação de recursos é enviada ao *FTPlugin*, que por sua vez notifica a necessidade para a aplicação cliente, que fica responsável por provisionar um novo recurso virtual em um de seus provedores de serviço de nuvem computacional. Esta característica permite aos usuários otimizarem o custo necessário para tolerar falhas, ou seja, cabe ao usuário final decidir, com base no tipo de negócio de suas aplicações, como os recursos computacionais serão utilizados pelo modelo. Desta forma, a característica de consumo sob demanda do paradigma da computação em nuvem é mantido até mesmo para tolerar falhas.

A Figura 5.3 apresenta um diagrama de sequência que expressa a interação que corresponde à inicialização do *FTPlugin* e à inicialização do processo correspondente ao *FTCoordinator*.

A chamada ao último método *imalive* do diagrama de sequência ilustrado na Figura 5.3, ocorre de forma periódica, conforme a configuração do atributo de *heartbeatTimeInSeconds*, dessa forma, a frequência com que o modelo de tolerância falhas realiza a operação de *heartbeat* pode variar de acordo com a parametrização efetuada pelo usuário. Esta interação representa a ponte de comunicação entre o *FTPlugin* e o *FTCoordinator*.

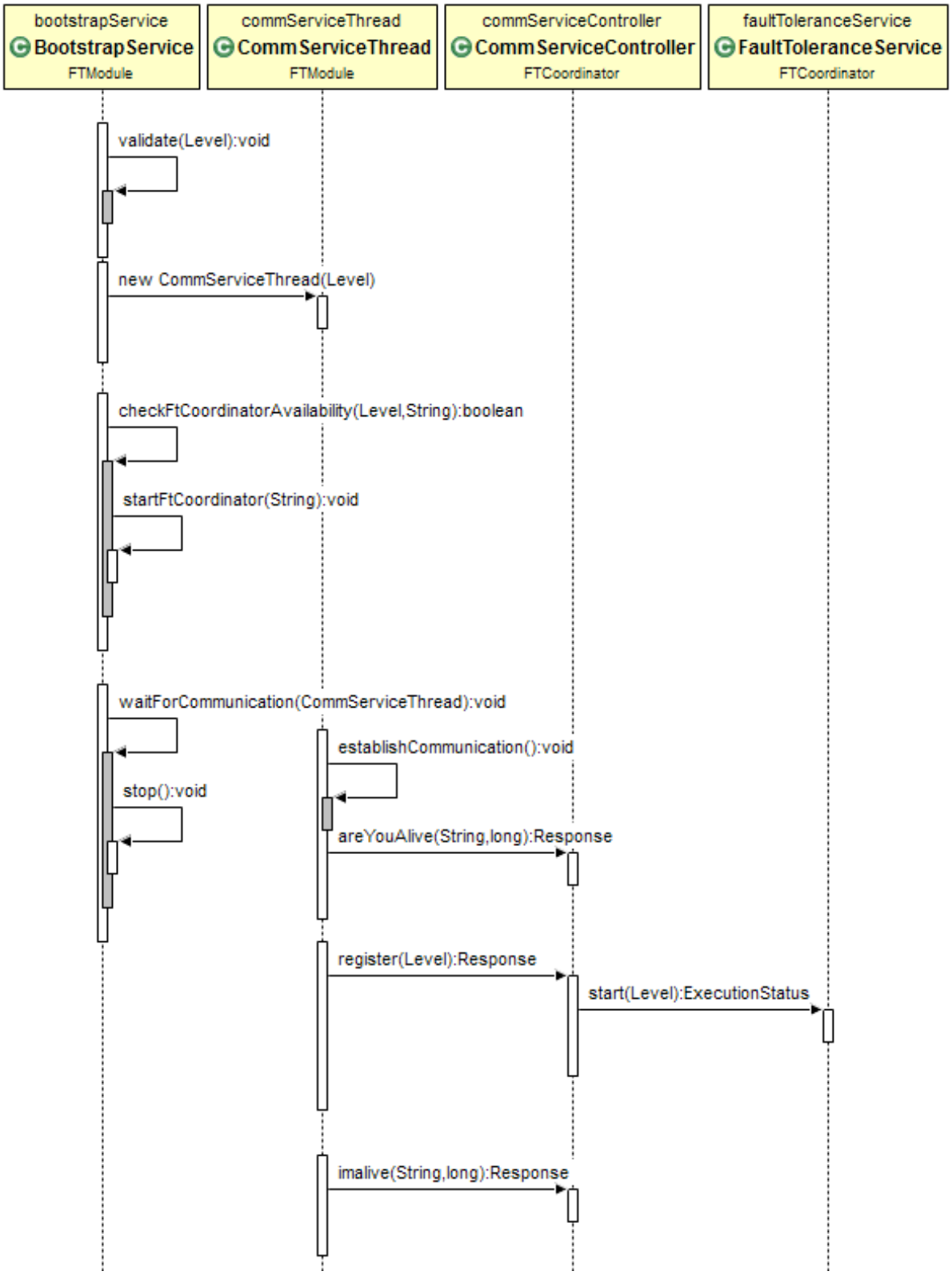

Figura 5.3: Diagrama de Sequência de Interação entre o *FTPlugin* e o *FTCoordinator*.

Já Figura 5.4 apresenta o diagrama de sequência que expressa a interação de implementação do *FTCoordinator*.

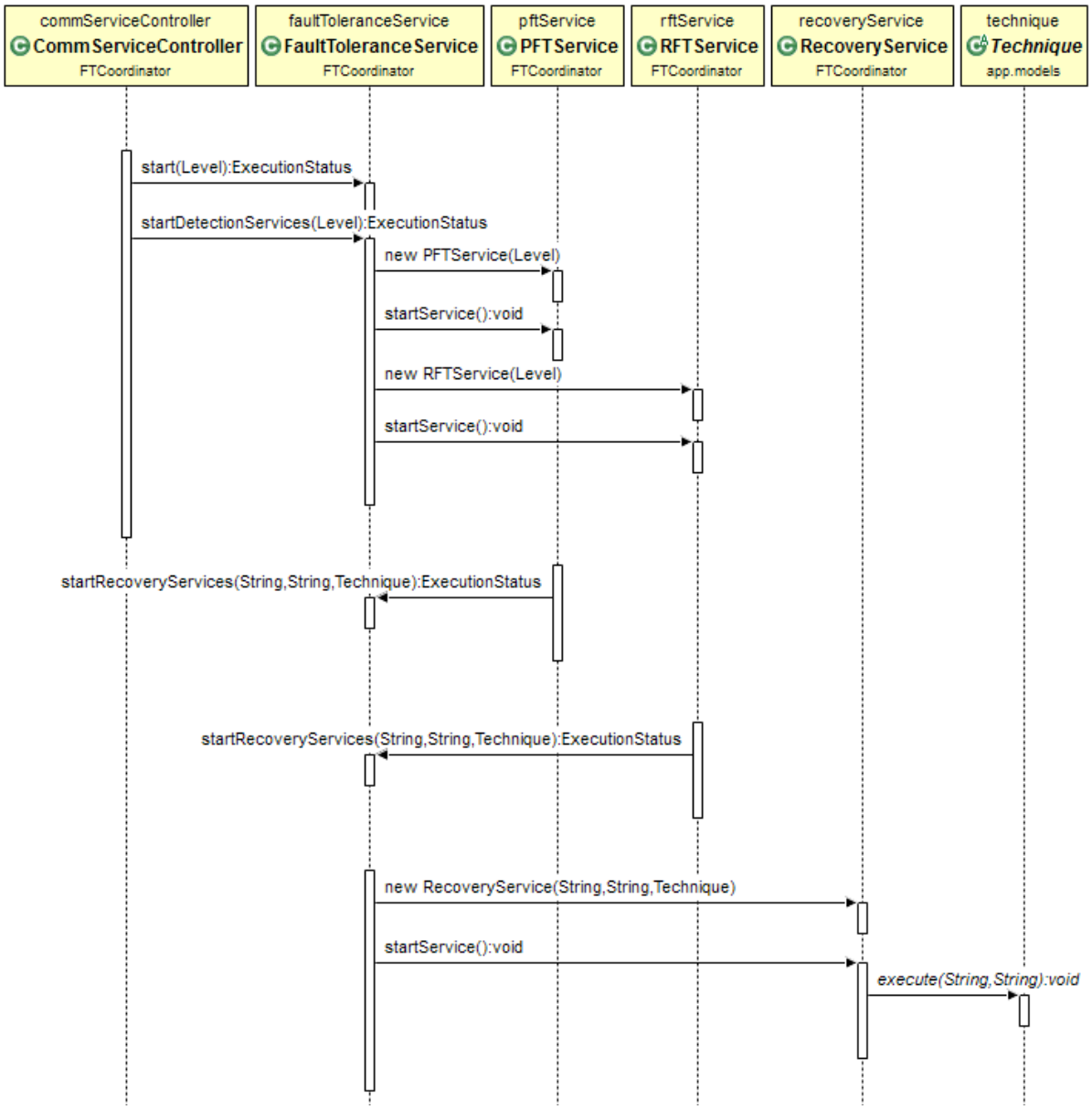

Figura 5.4: Diagrama de Sequência de Interação do *FTCoordinator*.

A criação e a configuração dos serviços de tolerância a falhas que implementam as políticas proativas e reativas, exemplificadas na Figura 5.4, ocorrem de acordo com o nível de tolerância a falhas configurado pela aplicação cliente, ou seja, o *PFTService* apenas será utilizado quando o conjunto ou a lista das técnicas configuradas possuir a técnica de rejuvenescimento de software, e o serviço de *RFTService* apenas será utilizado quando o nível de tolerância a falhas possuir ao menos uma das técnicas reativas implementadas por este modelo, o que possibilita ao modelo proposto trabalhar com uma ou ambas as políticas de tolerância a falhas.

Assim sendo, a Figura 5.5 ilustra, em três passos, a interação entre uma aplicação

cliente, o *FTPlugin* e o *FTCoordinator*. O *FTPlugin*, como dito anteriormente, é equivalente a uma biblioteca que será integrada na aplicação cliente com o objetivo de fornecer uma interface de interação com o coordenador de tolerância a falhas e o *FTCoordinator*, como dito, fica responsável por configurar, monitorar e aplicar o nível de tolerância a falhas configurado pela aplicação cliente.

O primeiro passo corresponde à integração entre a aplicação cliente e o *FTPlugin*. Já o segundo passo corresponde à inicialização do *FTCoordinator* pelo *FTPlugin*. Por fim, o terceiro e último passo ilustra a ponte de comunicação entre o *FTPlugin* e o *FTCoordinator*. Neste caso, assume-se que a aplicação cliente, o *FTPlugin* e o *FTCoordinator* são executadas no mesma instância de máquina virtual.

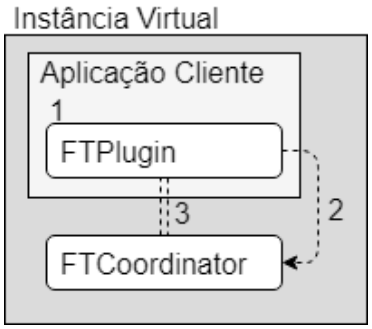

Figura 5.5: Interação entre Aplicação Cliente, *FTPlugin* e *FTCoordinator*.

A Figura 5.6 exemplifica o modelo proposto aplicado a um sistema distribuído que encontra-se em um ambiente de federação de nuvens. O cenário ilustrado é composto por uma aplicação (Instância Principal - ip) que supostamente parametriza o nível de tolerância a falhas com o objetivo de manter a aplicação em execução em conjunto com três réplicas (Instância Secundária - is). Para selecionar instâncias candidatas, o modelo busca na coleção de instâncias disponíveis (*pool* de instâncias contornadas pela linha tracejada), aquelas que melhor atendam aos requisitos da aplicação, para então replicar a aplicação cliente.

Ainda é possível que cada instância replicada possua seu próprio nível de tolerância a falhas, supondo que as três réplicas criadas estejam configuradas para utilizar a técnica de reenvio de tarefas, e em caso de detecção de falhas, o *FTCoordinator*, responsável pela instância que falhou, busca por instâncias disponíveis e então, reenvia a réplica para a instância escolhida. Em outras palavras, com a utilização do modelo proposto é possível favorecer a dependabilidade até mesmo das réplicas criadas pelo mesmo. O trecho de código 5.1 ilustra como é simples realizar a integração do modelo proposto em uma aplicação cliente que deseja configurar o nível de tolerância a falhas com as técnicas de rejuvenescimento de software e repetição.

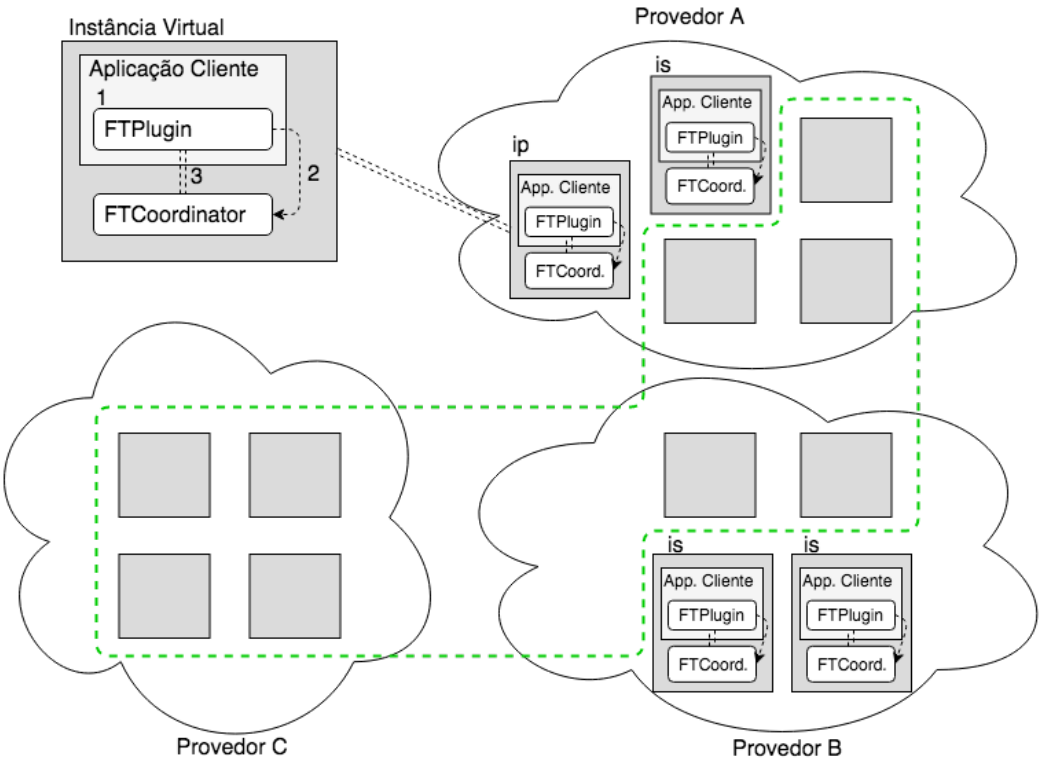

Figura 5.6: Visão Geral do Modelo Aplicado (ip: instância principal, is: instância secundária).

```
1   @OnApplicationStart
2   public class FaultToleranceConfigurationJob {
3
4       public void doJob() {
5
6           final FaultTolerancePlugin ftPlugin =
7               BootstrapService.getInstance();
8           final Level ftLevel =
9               new Level(STARTUP_COMMAND, HEARTBEAT_TIME);
10          ftLevel.setModuleId(RUNTIME_MODULE_ID);
11          ftLevel.addTechnique(bindSoftwareRejuvenation());
12          ftLevel.addTechnique(bindRetry());
13          ftPlugin.start(
14              ftLevel,
15              STARTUP_COORDINATOR_COMMAND);
16      }
17
18
```

```
19          private static SoftwareRejuvenation bindSoftwareRejuvenation() {
20
21              final Long rejuvanationTimeout = 60 * 60 * 24L; // 24 HOURS
22              final int maxAllowedCpuUsage = 97; // 97%
23              final int maxAllowedMemoryUsage = 97; // 97%
24              final Timeout timeout = new Timeout(
25                  rejuvanationTimeout,
26                  DEFAULT_TIME_UNIT);
27              return new SoftwareRejuvenation(
28                  timeout,
29                  maxAllowedCpuUsage,
30                  maxAllowedMemoryUsage);
31          }
32
33          private static Retry bindRetry() {
34
35              final Timeout timeout = new Timeout(1L, DEFAULT_TIME_UNIT);
36              return new Retry(
37                  timeout,
38                  new AttemptsNumber(DEFAULT_ATTEMPTS_NUMBER));
39          }
40      }
41
```

Trecho de Código 5.1: Integração com o Modelo de Tolerância a Falhas Proposto.

Na linha de número 6, é possível observar a integração do *FTPlugin* com aplicações cliente. Já as linhas de número 8 a 12, configuram o nível de tolerância a falhas desejado. Por fim, a linha de número 13 inicia o *FTPlugin*, o que estabelece uma ponte de comunicação com o *FTCoordinator*. Desta forma, é possível notar que a estratégia adotada para favorecer o nível de tolerância a falhas pode ser configurada de várias formas, combinando e utilizando as técnicas que melhor atendam um determinado modelo de negócio.

Assim sendo, em relação aos trabalhos relacionados apresentados na Seção 3.7, nota-se que o modelo proposto é o único que implementa ambas as políticas de tolerância a falhas, proativa e reativa, com foco no consumidor de serviços de federação de nuvem computacional. Além disto, o modelo proposto também é o único que permite a parametrização do nível de tolerância a falhas desejado. Em adicional, o modelo proposto é o que implementa o maior número de técnicas de tolerância a falhas. A Tabela 5.1 apresenta uma comparação entre os trabalhos relacionados e o modelo proposto.

Tabela 5.1: Comparação dos Trabalhos Relacionados.

Proposta	Políticas	Técnicas	Voltado	Parametrizável	Modelo de Nuvem
Zhao *et al.* [74]	Reativa	Replicação	Provedores	Não	Única
Zhang *et al.* [75]	Reativa	Replicação	Provedores	Não	Única
Veronese *et al.* [76]	Reativa	Replicação	Provedores	Não	Única
Garraghan *et al.* [77]	Reativa	Replicação	Consumidores	Não	Única e Federada
Jhawar *et al.* [78]	Reativa	Repetição e Migração	Provedores	Não	Única
Das *et al.* [79]	Proativa Reativa	Balanceamento de Carga, Replicação e Ponto de Checagem/Reinício	Provedores	Não	Única
Joshi *et al.* [80]	Reativa	Balanceamento de Carga e Replicação	Provedores	Não	Única
Bala *et al.* [81]	Proativa	Aprendizado de máquina	Provedores	Não	Única
Chen *et al.* [82]	Reativa	Repetição	Provedores	Não	Única
Modelo Proposto	Proativa Reativa	Rejuvenescimento de Software, Repetição, Reenvio de Tarefas, Replicação e Tratamento de exceções definido pelo usuário	Consumidores	Sim	Única e Federada

Como análise experimental, o modelo proposto será integrado ao sistema BioNimbuZ, uma plataforma de federação de nuvens computacionais híbridas para a execução de *workflows* de Bioinformática, com o objetivo de fornecer tolerância a falhas a todas as suas camadas. Mais detalhes sobre a utilização do modelo proposto na plataforma BioNimbuZ são apresentados na Seção 5.2.

5.2 Integração do Modelo Proposto ao BioNimbuZ

Conforme apresentado na seção anterior, o BioNimbuZ já possui um serviço responsável pela "tolerância a falhas" em sua arquitetura. Todavia, o serviço hoje existente apenas implementa o princípio básico da tolerância a falhas (*i.e.*, redundância de dados), e ainda o aplica apenas para os arquivos de saída relacionados à execução de *workflows*. O atual serviço de tolerância a falhas deixa de fora o controle da própria plataforma, ou seja, se uma falha por queda ocorrer na Camada de Aplicação, por exemplo, todo o serviço se torna indisponível aos usuários da plataforma.

Conforme citado anteriormente, o gerenciamento e o tratamento de falhas em sistemas federados se torna cada vez mais complexo. Visto isso, percebe-se que a plataforma BioNimbuZ não possui abordagem e nem mecanismos eficientes para a detecção e a recuperação do sistema para falhas que venham a ocorrer durante a sua operação, e por este motivo, esta plataforma foi escolhida para a realização da análise experimental, onde implementou-se o modelo proposto neste trabalho.

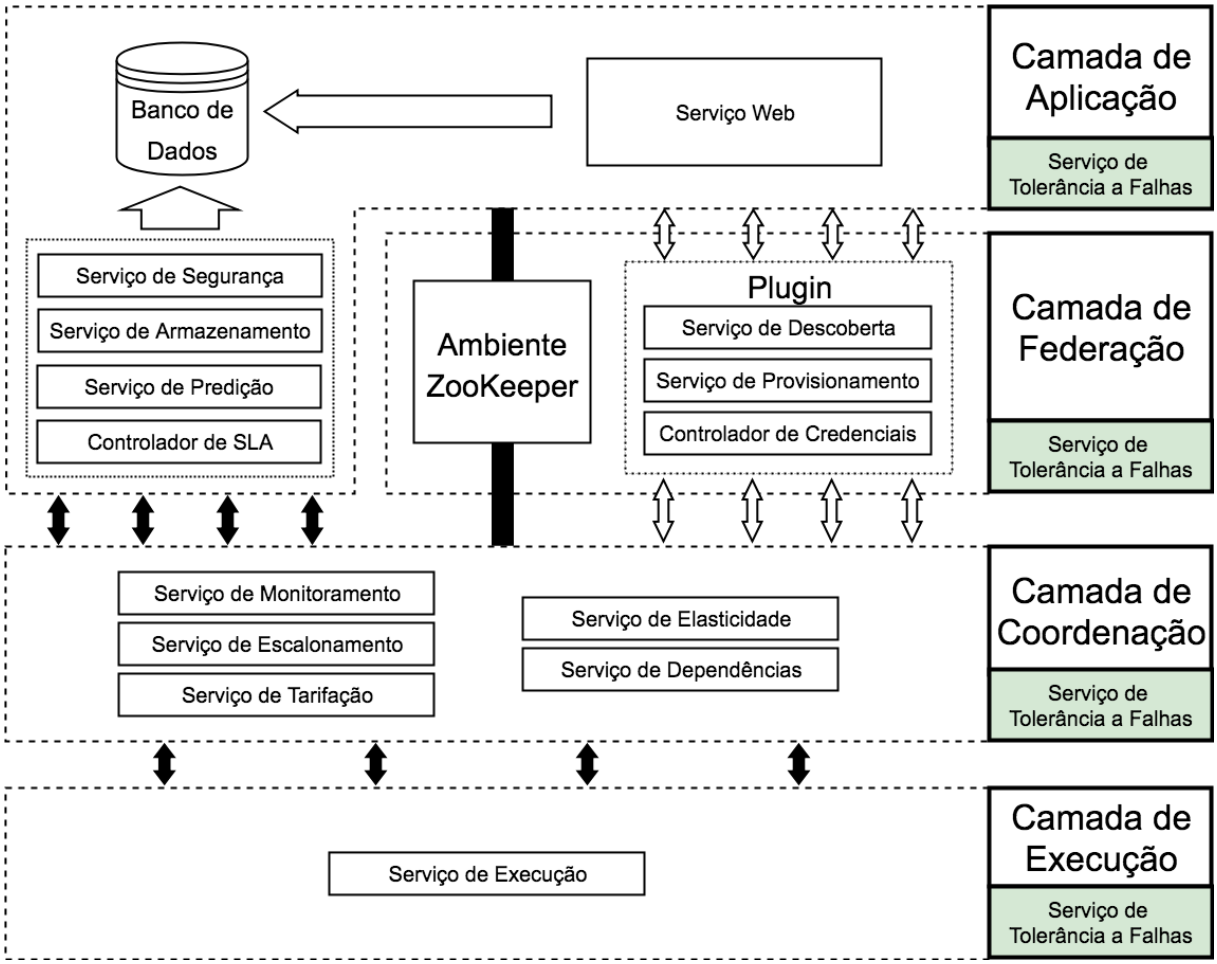

Figura 5.7: Arquitetura do BioNimbuZ Integrado ao Modelo de Tolerância a Falhas.

A Figura 5.7 ilustra a forma como o modelo proposto neste trabalho foi integrado à arquitetura da plataforma BioNimbuZ. Neste caso, o modelo de tolerância a falhas foi integrado em todos as quatro camadas no BioNimbuZ, assim, pretende-se fornecer um nível de tolerância a falhas parametrizável à toda plataforma. Em outras palavras, com a utilização do modelo proposto, é possível configurar um nível de tolerância a falhas diferente para cada uma das quatro camadas existentes, por exemplo, na Camada de Aplicação (1) e na Camada de Federação (2), que consistem em sistemas com a característica de permanecer disponível 24 horas por dia, é plausível configurar o nível de tolerância a falhas com as técnicas de rejuvenescimento de software e de repetição. Já para a Camada de Coordenação (3), que consiste em um sistema que é executado sob demanda em um determinado provedor de nuvem, é possível configurá-lo apenas com a técnica de repetição, por fim, (4) a Camada de Execução, que também consiste em um sistema executado sob demanda, mas em diversos provedores de nuvem, pode-se configurar o nível de tolerância a falhas para utilizar a técnica de reenvio de tarefas e replicação, e é para esta camada que será disponibilizada a opção de configuração do nível de tolerância a falhas até mesmo

para os usuários finais, que poderão prover uma parametrização que melhor atenda aos seus modelos de negócio. Dessa forma, o modelo proposto garante, com esta configuração, a implementação de tolerância a falhas mais flexível, eficiente e otimizada, pois permite que seja usada a técnica mais adequada de acordo a necessidade. Isto apenas é possível porque o modelo é multi-estratégico, flexível e modular, diferenciando-o de todos os trabalhos relacionados, apresentados na Seção 3.7.

A Figura 5.8 exemplifica a integração do modelo proposto e o fluxo de execução do BioNimbuZ, citado anteriormente na Seção 4.6 e ilustrado na Figura 4.2. Desta forma, é possível observar como é na prática a integração do *plugin* e do coordenador de tolerância a falhas (*FTPlugin* e *FTCoordinator*) nas quatro camadas do BioNimbuZ. É importante ressaltar que a parametrização do nível de tolerância a falhas para usuários finais apenas se aplica na Camada de Execução, pois esta é a camada responsável por executar as tarefas criadas na interface gráfica, onde se possibilita que os próprios usuários configurem o nível de tolerância a falhas desejado para cada tarefa, as demais camadas são parametrizadas de maneira sistêmica, ou seja, apenas os desenvolvedores do BioNimbuZ tem acesso à sua configuração.

Assim, é possível perceber que no cenário apresentado nas Figuras 5.7 e 5.8, os níveis de tolerância a falhas foram consideravelmente aumentados, tanto para a plataforma BioNimbuZ quanto para as tarefas que constituem os *workflows*, tarefas estas executadas pela Camada de Execução.

Dessa forma, o modelo implementado se destaca pela variedade de técnicas disponibilizadas para aplicar a tolerância a falhas, bem como por possibilitar que até mesmo usuários finais configurem o nível de tolerância a falhas que atenda às suas expectativas de custo de acordo com o seu modelo de negócio. Além disto, o modelo ainda é capaz de permitir que os serviços de um sistema distribuído implementem, dentre as técnicas disponibilizadas pelo modelo, aquela que melhor atenda a sua necessidade.

5.3 Considerações Finais

Este capítulo apresentou os detalhes do modelo de tolerância a falhas implementado neste trabalho, que tem como principal objetivo facilitar a utilização de políticas e técnicas de tolerância a falhas para tratar falhas por queda e falhas por tempo, servindo aplicações clientes como uma biblioteca.

Além disso, o modelo proposto permite que o nível de tolerância a falhas possa ser parametrizado por usuários finais. Possibilitando a configuração do modelo para trabalhar com as políticas de tolerância a falhas proativa e reativa.

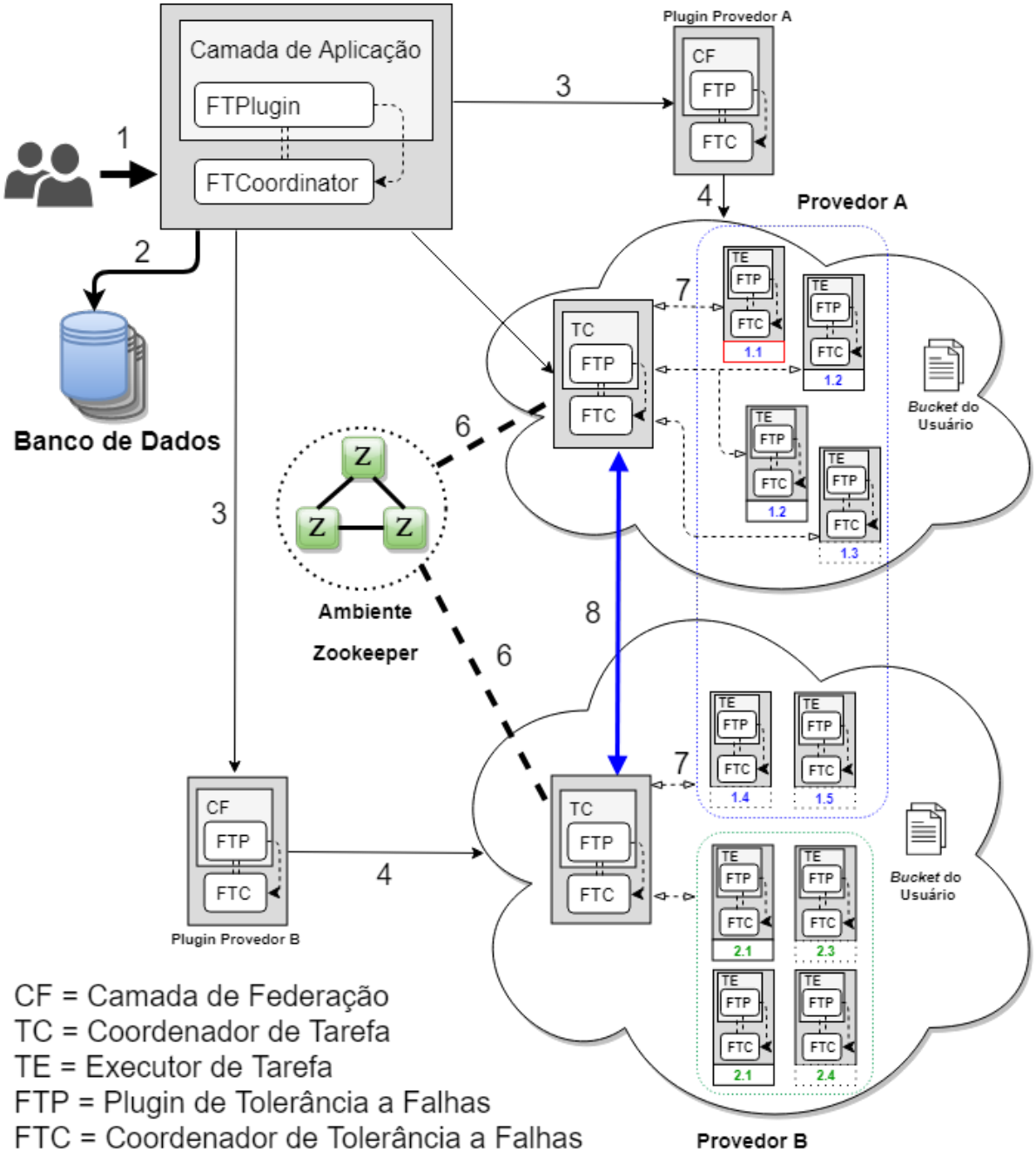

Figura 5.8: Comunicação entre os Diferentes Componentes e o Modelo de Tolerância a Falhas Proposto Neste Trabalho.

Dentre as principais técnicas de tolerância a falhas descritas no Capítulo 3, o modelo proposto fornece a implementação das técnicas: (1) rejuvenescimento de software; (2) repetição de tarefas; (3) reenvio de tarefas; e (4) replicação de tarefas. Desta forma, pretende-se elevar os níveis de dependabilidade de plataformas que se beneficiem do modelo de nuvem computacional, em especial, nuvens federadas. Também tratou-se da

plataforma de nuvens federadas para a execução de *workflows* de Bioinformática, o Bio-NimbuZ, e a integração do modelo implementado no mesmo.

Capítulo 6

Resultados

6.1 Visão Geral

Neste capítulo serão apresentados os testes realizados na plataforma de federação de nuvem BioNimbuZ e os resultados obtidos, em ralação ao aumento da confiabilidade e da disponibilidade. O objetivo é avaliar o modelo de tolerância a falhas proposto neste trabalho.

Para tal, foi configurada uma federação de nuvem computacional composta pelos provedores de nuvem *Google Cloud Platform* e *Amazon Web Services*, na qual a plataforma BioNimbuZ foi executada e integrada a um mecanismo de injeção de falhas, e a um mecanismo de inspeção de falhas, desenvolvidos especificamente para analisar a eficácia do modelo proposto.

A análise dos resultados terá como foco expressar as medidas de avaliação de dependabilidade, relacionadas a disponibilidade e a confiabilidade, detalhadas na Seção 3.3.1.

Para qualificar a disponibilidade e a confiabilidade da plataforma BioNimbuZ, serão utilizadas as métricas de tempo médio para o defeito (MTTF, do inglês *Mean Time To Failure*), de tempo médio entre os defeitos (MTBF, do inglês *Mean Time Between Failures*) e o tempo médio de reparo (MTTR, do inglês *Mean Time To Repair*). Além disto este capítulo também trata do *overhead* agregado à utilização do modelo de tolerância a falhas.

6.2 Estratégia de Execução dos Testes

Como estratégia para a execução dos testes, a plataforma BioNimbuZ foi configurada com um nível de tolerância a falhas que nos permitiu utilizar todas as técnicas de tolerância propostas no modelo. Para isto, as camadas existentes na plataforma BioNimbuZ foram configuradas como se segue:

- **Camada de Aplicação**: por ser uma camada que tem como característica permanecer disponível 24 horas por dia, o nível de tolerância a falhas foi configurado com as técnicas de rejuvenescimento de software e de repetição;
- **Camada de Federação**: por possuir as mesmas características da camada anterior, esta camada também foi configurada com as técnicas de rejuvenescimento de software e de repetição;
- **Camada de Coordenação**: por consistir em um sistema que é executado sob demanda em um determinado provedor de nuvem, esta camada foi configurada com a técnica de repetição;
- **Camada de Execução**: também consiste em um sistema executado sob demanda, todavia é executado em diversos provedores de nuvem, assim, configurou-se o nível de tolerância a falhas para utilizar a técnica de reenvio de tarefas e de replicação.

A Tabela 6.1 resume a configuração das políticas e das técnicas de tolerância a falhas realizada em cada uma das camadas da plataforma BioNimbuZ, conforme dito acima.

Tabela 6.1: Configuração de Tolerância a Falhas por Camada.

Camada	**Política**	**Técnica de Tolerância a Falhas**
Aplicação	Proativa e Reativa	Rejuvenescimento de software e Repetição
Federação	Proativa e Reativa	Rejuvenescimento de Software e Repetição
Coordenação	Reativa	Repetição
Execução	Reativa	Reenvio de Tarefas e Replicação

Já a Tabela 6.2 ilustra como foi efetuada a configuração do mecanismo de injeção de falhas. Estas configurações foram utilizadas da mesma forma em todas as camadas da plataforma BioNimbuZ.

Tabela 6.2: Configuração da Injeção de Falhas.

Tempo de Avaliação	**MTTF**	**MTBF**	**Número de Falhas**
01 hora	05 minutos	05 minutos	11 falhas
02 horas	05 minutos	05 minutos	23 falhas
03 horas	05 minutos	05 minutos	35 falhas
04 horas	05 minutos	05 minutos	47 falhas

Para analisar a eficácia do modelo, também foi desenvolvido um mecanismo de inspeção de falhas, para computar o tempo de disponibilidade, de indisponibilidade e de

MTTR. Para isso, o mecanismo de inspeção teve como configuração, realizar requisições HTTP periódicas, em intervalos de um segundo, para cada uma das quatro camadas do BioNimbuZ. Com isto, foi possível avaliar na plataforma BioNimbuZ, dentro de determinados períodos, o tempo de resposta, o número de requisições respondidas com sucesso e o número de requisições respondidas com erro, que são insumos para os cálculos da confiabilidade e da disponibilidade.

Para a realização dos testes foram selecionadas, dentre as instâncias de máquinas virtuais existentes nos provedores de nuvem utilizados, aquelas que possuíam menor custo de utilização e a maior semelhança de hardware. A Tabela 6.3 detalha o tipo de instância utilizada durante os testes em cada um dos provedores. As regiões e as zonas utilizadas para a criação das máquinas virtuais foram as mesmas para ambos os provedores, região *us-east-1* e zona *us-east-1b*.

Tabela 6.3: Ambiente de Testes.

Provedor	**Tipo de instância**	**Números de CPUs**	**Memória RAM**	**Custo por Hora**
Google Cloud Platform	n1-standard-1	1 (2.3 GHz)	3.75 GB	0.0475
Amazon Web Services	t2.micro	1 (3.3 GHz)	1.00 GB	0.0116

Finalmente, a Tabela 6.4 apresenta detalhadamente a parametrização utilizada para configurar em cada uma das técnicas utilizadas o nível de tolerância falhas desejado para a execução dos testes.

Tabela 6.4: Parametrização do Nível de Tolerância a Falhas.

	Rejuvenescimento de Software	**Repetição**	**Reenvio de Tarefas**	**Replicação**
maxCpuUsage	97%			
maxMemoryUsage	97%			
heartbeatTimeInSeconds		01 segundo	01 segundo	
priority		0	1	
timeout	02 horas	01 segundo	01 segundo	

Com isto, define-se como ocorreram as baterias de testes na plataforma BioNimbuZ em ambos os provedores de nuvem computacional que formam a federação do ambiente testado.

6.3 Resultados Obtidos

Esta Seção, tem como objetivo apresentar os resultados obtidos nos cálculos de disponibilidade e confiabilidade. Com isto, deseja-se expressar o aumento dos índices de disponibilidade e de confiabilidade para a plataforma BioNimbuZ. Os cálculos para o percentual

de disponibilidade e de confiabilidade apresentados aqui, foram obtidos conforme a Equação 3.2 e a Equação 3.4, apresentadas anteriormente no Capítulo 3.

6.3.1 Tempo Médio de Reparo - MTTR

A Figura 6.1 exibe o resultado em relação ao MTTR, da média obtida em milissegundos, para cada uma das camadas da plataforma BioNimbuZ em ambos os provedores utilizados, *Google Cloud Platform* (GCP) e *Amazon Web Services* (AWS).

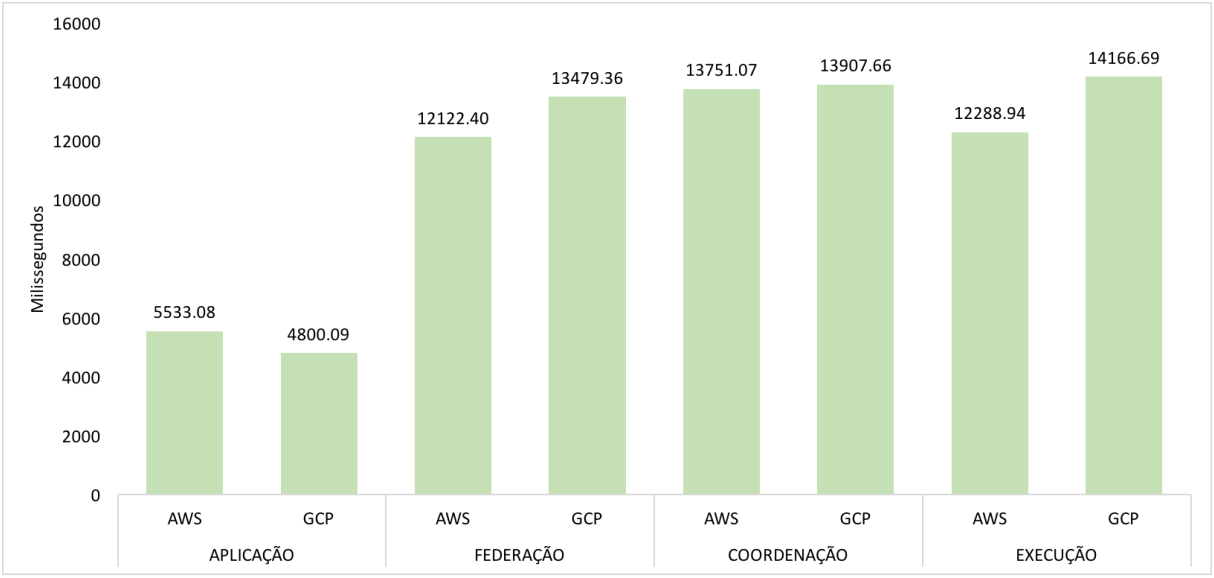

Figura 6.1: Tempo Médio de Reparo - MTTR.

Com a parametrização efetuada para o nível de tolerância a falhas, o tempo para a detecção de falhas é de apenas um segundo. Todavia, o tempo de recuperação depende do tempo necessário para inicializar a tarefa que falhou, este fato justifica a variação obtida para o tempo médio de reparo em cada uma das camadas da plataforma BioNimbuZ.

É possível observar que no provedor de nuvem *Amazon Web Services*, o valores alcançados para tempo médio de reparo foram melhores que os alcançados com o provedor *Google Cloud Platform* na maioria dos casos. O motivo não foi investigado a fundo, mas o fato é curioso, visto que os recursos utilizados na *Amazon Web Services* possuem menor capacidade de memória, conforme descrito na Tabela 6.3.

Por fim, a Figura 6.2 expressa os percentuais computados para a disponibilidade e para a confiabilidade, também em cada uma das camadas analisadas, e também em ambos os provedores de nuvem utilizados.

Os resultados obtidos após a execução dos testes mostram que com o modelo de tolerância a falhas proposto, é possível garantir para a plataforma BioNimbuZ, com a para-

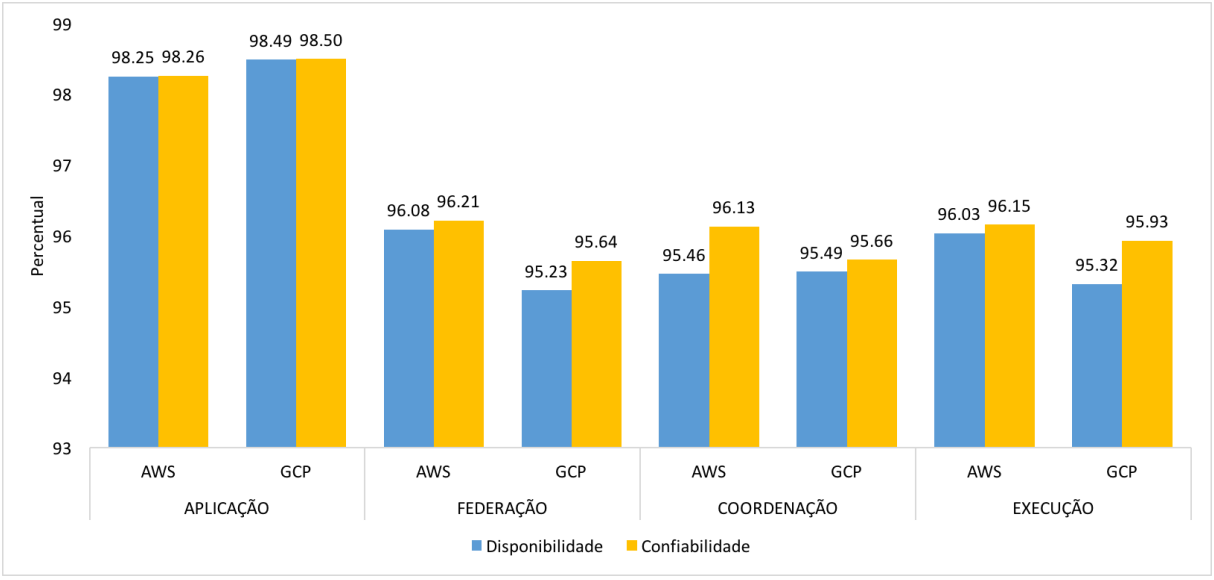

Figura 6.2: Percentual de Disponibilidade e de Confiabilidade.

metrização utilizada, o mínimo de 95,23% de disponibilidade e 95,64% de confiabilidade, mesmo com a injeção periódica de falhas a cada 5 minutos.

Assim, é dedutível que os indicadores de disponibilidade e de confiabilidade variam de acordo o tempo de injeção de falhas, para avaliar o modelo, a frequência configurada para a injeção de falhas é relativamente alta, o que nos mostra, que em situações reais os percentuais dos indicadores aumentariam de forma significativa. Em outras palavras, os resultados provam que o modelo é capaz de permitir a correta operação do sistema mesmo sob a presença constante de falhas. Desta forma, elava-se o nível de tolerância a falhas da plataforma, que contava apenas com o princípio básico da tolerância a falhas (*i.e.*, redundância de dados), aplicando-o apenas para os arquivos de saída relacionados à execução de suas tarefas. Desta forma, o sistema se manteria disponível apenas até a injeção da primeira falha em qualquer uma de suas camadas.

Aqui, também é possível observar, na maioria dos casos, que os níveis alcançados para a disponibilidade e para a confiabilidade foram melhores no provedor de nuvem *Amazon Web Services*. Todavia, este comportamento já era esperado, tendo visto que este provedor ofereceu melhor tempo médio de reparo, conforme apresentado na Figura 6.1.

6.4 *Overhead* Associado

Para analisar o *overhead*, relacionado ao tempo de inicialização, causado pela integração do modelo de tolerância a falhas proposto na plataforma BioNimbuZ, foram utilizados dois cenários para execução dos testes. No primeiro cenário foi calculado o tempo necessário para inicialização de cada uma das camadas do BioNimbuZ, sem a integração do modelo

proposto, obtendo assim, a média de tempo necessário para a inicialização de cada uma das quatro camadas. O segundo cenário, consiste na mesma apuração, mas agora fazendo o uso do modelo proposto.

Para a apuração do *overhead*, as camadas do BioNimbuZ foram modificadas temporariamente com o objetivo de remover os trechos que necessitam realizar comunicação com os provedores de nuvem, pois esta característica influenciaria no resultado, já que a latência de comunicação com os provedores pode sofrer variações de forma não prevista. A Figura 6.3 detalha o resultado obtido para o *overhead* associado à utilização do modelo de tolerância a falhas.

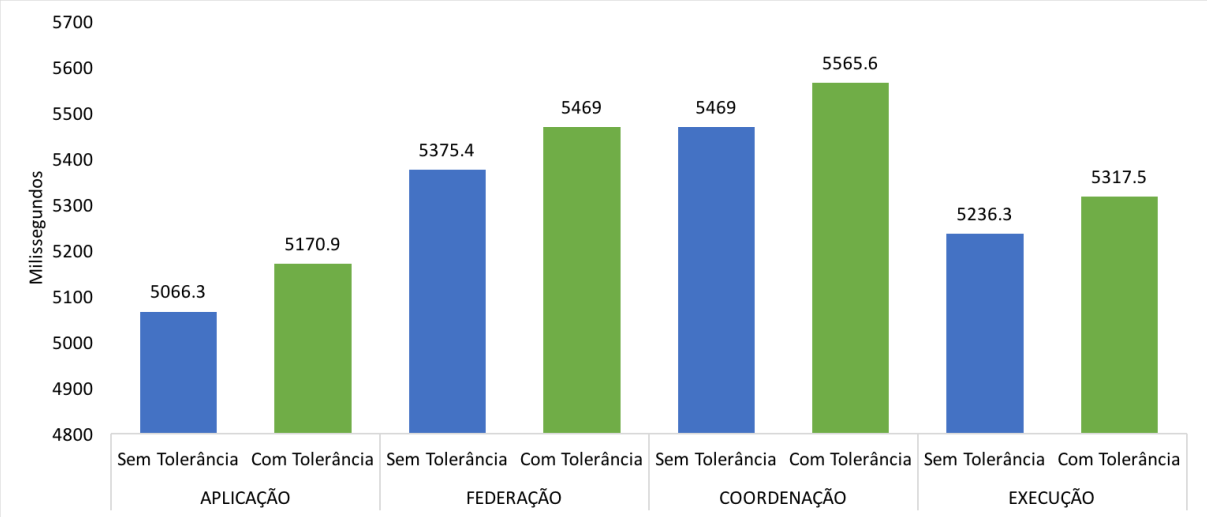

Figura 6.3: *Overhead* Associado.

Calculando o percentual relacionado ao *overhead* associado à utilização do modelo de tolerância a falhas em cada uma das quatro camadas da plataforma BioNimbuZ, tem-se 2.06% de *overhead* para a Camada de Aplicação, 1.74% para a Camada de Federação, 1.77% para Camada de Coordenação e 1.55% para a Camada de Execução.

Desta forma, Os resultados desta análise mostram que o modelo de tolerância a falhas não agrega *overhead* significativo para a plataforma BioNimbuZ, garantindo a sua eficácia na implementação da multi-estratégia de tolerância a falhas em uma plataforma real de federação de nuvem computacional.

6.5 Considerações Finais

Este capítulo apresentou os testes efetuados na plataforma BioNimbuZ, bem como os resultados obtidos para o tempo médio de reparo e para os níveis de confiabilidade e de disponibilidade, com a utilização do modelo proposto neste trabalho. Também foi apresentado o *overhead* associado à utilização do modelo.

Por fim, este capítulo permitiu observar que os resultados apresentaram o aumento dos níveis de confiabilidade e de disponibilidade associados à integração do modelo proposto à plataforma BioNimbuZ.

Capítulo 7

Conclusões

Este trabalho tratou de vários aspectos relacionados a dependabilidade de sistemas distribuídos, detalhando os desafios e as limitações existentes para a adoção do modelo de nuvem computacional. A complexidade presentes neste tipo de ambiente torna evidente que a tolerância a falhas é uma questão de suma importância tanto para provedores quanto para consumidores. Tendo observado tais fatos, diversos aspectos referentes a tolerância a falhas foram abordados, pois para enfrentar esses desafios e favorecer a dependabilidade, é crucial que técnicas de tolerância a falhas sejam utilizadas para que sistemas sejam capazes de tolerar falhas e permanecer com sua correta operação.

Diante disto, este trabalho apresentou um modelo multi-estratégico de tolerância a falhas para ambientes distribuídos, que implementa políticas proativas e reativas da tolerância a falhas. O modelo proposto teve como principal objetivo possibilitar a configuração do nível de tolerância a falhas mais adequado ao modelo de negócio de de cada usuário final. Dentre as principais técnicas de tolerância a falhas existentes, foram implementadas as técnicas de rejuvenescimento de *software*, repetição de tarefas, reenvio de tarefas e replicação. Isto com o objetivo de elevar os níveis de dependabilidade de sistemas distribuídos.

Para cumprir com tais objetivos, também foram apresentados neste trabalho detalhes da plataforma BioNimbuZ, que foi escolhida para a análise experimental, por ser tratar de uma plataforma de federação de nuvens computacionais híbridas para a execução de *workflows* de Bioinformática.

Os resultados dos testes realizados com a análise experimental, para o modelo de tolerância a falhas proposto, possibilitaram observar o aumento no nível de confiabilidade e de disponibilidade da plataforma BioNimbuZ, na qual observou-se um valor mínimo de 95,23% de disponibilidade e 95,64% de confiabilidade.

Também foi evidenciado que o modelo proposto é o único que implementa ambas as políticas de tolerância a falhas, proativa e reativa, com foco no consumidor de serviços

de federação de nuvem computacional. Além disto, o modelo proposto também é o único que permite a parametrização do nível de tolerância a falhas desejado. Em adicional, o modelo proposto é o que implementa o maior número de técnicas de tolerância a falhas.

Além disso, este trabalho mostra que o modelo de tolerância a falhas proposto pode ser facilmente integrado a qualquer tipo de sistema, seja ele distribuído ou não, por meio da biblioteca *FTPlugin*.

Dentre as possibilidades de trabalhos futuros ou de continuidade deste trabalho, é proposto o incremento do modelo com a implementação de novas técnicas de tolerância a falhas, tais como a auto-reparação e o ponto de checagem/reinício. Além disso, é possível aprofundar o modelo para que o mesmo seja capaz de identificar ou eleger automaticamente a melhor técnica de tolerância a falhas para um determinado tipo de falha ou para um determinado tipo de tarefa, fazendo o uso de aprendizagem de máquina ou da observação de dados históricos.

Referências

[1] Toosi, Adel Nadjaran, Rodrigo N. Calheiros e Rajkumar Buyya: *Interconnected cloud computing environments: Challenges, taxonomy, and survey.* ACM Comput. Surv., 47(1):7:1–7:47, maio 2014, ISSN 0360-0300. `http://doi.acm.org/10.1145/2593512`. ix, 4, 9, 10, 13, 14, 16, 21

[2] Thomas J. Bittman, from Gartner information technology research: *The evolution of the cloud computing market.* `http://blogs.gartner.com/thomas_bittman/2008/11/03/the-evolution-of-the-cloud-computing-market/`, 2008. Acessado em: 07/04/2017. ix, 12, 13

[3] Celesti, A., F. Tusa, M. Villari e A. Puliafito: *How to enhance cloud architectures to enable cross-federation.* Em *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, páginas 337 –345. IEEE, jul. 2010. ix, 12, 13, 15

[4] Weber, Taisy Silva: *Um roteiro para exploração dos conceitos básicos de tolerância a falhas.* Relatório técnico, Instituto de Informática UFRGS, 2002. ix, 4, 19, 20

[5] Avizienis, A., J. C. Laprie, B. Randell e C. Landwehr: *Basic concepts and taxonomy of dependable and secure computing.* IEEE Transactions on Dependable and Secure Computing, 1(1):11–33, Jan 2004, ISSN 1545-5971. ix, 19, 22, 23, 24, 26, 27

[6] Eric Bauer, Randee Adams(auth.): *Reliability and Availability of Cloud Computing.* Wiley-IEEE Press, 2012, ISBN 9781118177013,9781118393994. ix, 24, 25

[7] Barborak, Michael, Anton Dahbura e Miroslaw Malek: *The consensus problem in fault-tolerant computing.* ACM Comput. Surv., 25(2):171–220, junho 1993, ISSN 0360-0300. `http://doi.acm.org/10.1145/152610.152612`. ix, 21, 27, 29

[8] San Murugesan, Irena Bojanova (eds.): *Encyclopedia of Cloud Computing.* Wiley-IEEE Press, 1ª edição, 2016, ISBN 1118821971,9781118821978. x, 1, 4, 9, 10, 11, 17, 25

[9] Tanenbaum, Andrew S. e Maarten van Steen: *Distributed Systems: Principles and Paradigms (2Nd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006, ISBN 0132392275. x, 20, 21, 22, 27, 28

[10] Mell, Petter e Timothy Grance: *The NIST definition of cloud computing.* Em *Recommendations of the National Institute of Standards and Technology.* Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011. 1, 8, 10, 11

[11] L. M. Vaquero, L. Rodero-Merino, J. Caceres e M. Lindner: *A break in the clouds: towards a cloud definition.* ACM SIGCOMM Computer Communication Review, 39(1):50–55, 2009. 1

[12] Platform, Google Cloud: *Resource Quotas.* `https://cloud.google.com/compute/quotas`, 2017. Acessado em: 07/03/2017. 1, 12

[13] Services, Amazon Web: *AWS Service Limits.* `http://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html`, 2017. Acessado em: 07/03/2017. 1, 12

[14] Azure, Microsoft: *Azure subscription and service limits, quotas, and constraints.* `https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits`, 2017. Acessado em: 03/04/2017. 1, 12

[15] Buyya, Rajkumar, Rajiv Ranjan e Rodrigo N. Calheiros: *InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services*, páginas 13–31. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, ISBN 978-3-642-13119-6. `http://dx.doi.org/10.1007/978-3-642-13119-6_2`. 1, 12

[16] Wang, Jianwu, Moustafa Abdelbaky, Javier Diaz-Montes, Shweta Purawat, Manish Parashar e Ilkay Altintas: *Kepler+ cometcloud: dynamic scientific workflow execution on federated cloud resources.* Procedia Computer Science, 80:700–711, 2016. 1, 3, 12

[17] Saldanha, H. V.: *BioNimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de Bioinformática.* Tese de Mestrado, Programa de Pós-Graduação em Informática, Departamento de Ciência de Computação, Universidade de Brasília, 2012. `http://repositorio.unb.br/handle/10482/12046`. 1, 40

[18] Kaur, Jasbir e Supriya Kinger: *Analysis of different techniques used for fault tolerance.* International Journal of Computer Science and Information Technologies, 5(3), 2014, ISSN 20975-9646. 1, 32

[19] Ferreira Leite, Alessandro Ferreira Leite: *A user-centered and autonomic multi-cloud architecture for high performance computing applications.* Theses, Université Paris Sud - Paris XI, dezembro 2014. `https://tel.archives-ouvertes.fr/tel-01127070`. 2, 3, 8, 12

[20] Bipin B. Nandi, Himadri Sekhar Paul, Ansuman Banerjee e Sasthi C. Ghosh: *Fault tolerance as a service.* IEEE Sixth International Conference on Cloud Computing, páginas 446–453, 2013, ISSN 2159-6182. 2

[21] Rosa, Michel, Breno Moura, Guilherme Vergara, Lucas Santos, Edward de Oliveira Ribeiro, Maristela Holanda, Maria Emilia Telles Walter e Aletéia Patrícia Favacho de Araújo: *Bionimbuz: A federated cloud platform for bioinformatics applications.* Em *IEEE International Conference on Bioinformatics and*

Biomedicine, BIBM 2016, Shenzhen, China, December 15-18, 2016, páginas 548–555, 2016. http://dx.doi.org/10.1109/BIBM.2016.7822580. 3, 40, 41

[22] Hoffa, Christina, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman e John Good: *On the use of cloud computing for scientific workflows*. Em *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, páginas 640–645. IEEE, 2008. 3

[23] Ludäscher, Bertram, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A Lee, Jing Tao e Yang Zhao: *Scientific workflow management and the kepler system*. Concurrency and Computation: Practice and Experience, 18(10):1039–1065, 2006. 3

[24] Services, Amazon Web: *Summary of the Amazon S3 Service Disruption in the Northern Virginia*. https://aws.amazon.com/message/41926/, 2017. Acessado em: 07/03/2017. 3, 31

[25] Services, Amazon Web: *Summary of the Amazon EC2 and Amazon RDS service disruption in the US east region*. https://aws.amazon.com/message/65648/, 2011. Acessado em: 07/03/2017. 3

[26] Services, Amazon Web: *Summary of the AWS service event in the US east region*. https://aws.amazon.com/message/67457/, 2012. Acessado em: 07/03/2017. 3

[27] Platform, Google Cloud: *GCE networking in us-central1 zones is experiencing disruption*. https://status.cloud.google.com/incident/compute/17006, 2017. Acessado em: 07/03/2017. 3, 31

[28] Platform, Google Cloud: *Network packet loss to Compute Engine us-west1 region*. https://status.cloud.google.com/incident/compute/17005, 2017. Acessado em: 07/03/2017. 3

[29] Platform, Google Cloud: *New VMs are experiencing connectivity issues*. https://status.cloud.google.com/incident/compute/17003, 2017. Acessado em: 07/03/2017. 3

[30] Azure, Microsoft: *RCA - Cooling Event - Japan East, Cloud Services and Virtual Machines - East US 2*. https://azure.microsoft.com/en-us/status/history/, 2017. Acessado em: 03/04/2017. 3, 31

[31] Barril, J. F. H., J. Ruyter e Qing Tan: *A view on internet of things driving cloud federation*. Em *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, páginas 221–226. IEEE, July 2016. 4, 16, 17

[32] Kurze, Tobias, Markus Klems, David Bermbach, Alexander Lenk, Stefan Tai e Marcel Kunze: *Cloud federation*. Em *Proceedings of the 2nd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2011)*, volume 1971548541. Citeseer, 2011. 4, 16

[33] Jhawar, Ravi e Vincenzo Piuri: *Chapter 1 - fault tolerance and resilience in cloud computing environments*. Em Vacca, John R. (editor): *Cyber Security and {IT} Infrastructure Protection*, páginas 1 – 28. Syngress, Boston, 2014, ISBN 978-0-12-416681-3. http://www.sciencedirect.com/science/article/pii/B978012416681300001X. 7, 32

[34] Foster, I., Yong Zhao, I. Raicu e S. Lu: *Cloud computing and grid computing 360-degree compared*. Em *Grid Computing Environments Workshop, 2008. GCE'08*, páginas 1 –10, nov. 2008. 7, 8

[35] Armbrust, Michael, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica e Matei Zaharia: *A view of cloud computing*. Commun. ACM, 53(4):50–58, abril 2010, ISSN 0001-0782. http://doi.acm.org/10.1145/1721654.1721672. 8

[36] Buyya, Rajkumar, Chee Shin Yeo, Srikumar Venugopal, James Broberg e Ivona Brandic: *Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility*. Future Generation Computer Systems, 25(6):599 – 616, 2009, ISSN 0167-739X. http://www.sciencedirect.com/science/article/pii/S0167739X08001957. 8, 12

[37] Rimal, B. P., C. Eunmi e I. Lumb: *A Taxonomy and Survey of Cloud Computing Systems*. Em *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, páginas 44–51, aug. 2009. 10, 11

[38] Kwok, T., T. Nguyen e L. Lam: *A software as a service with multi-tenancy support for an electronic contract management application*. Em *2008 IEEE International Conference on Services Computing*, volume 2, páginas 179–186. IEEE, July 2008. 10

[39] Zhang, Qi, Lu Cheng e Raouf Boutaba: *Cloud computing: state-of-the-art and research challenges*. Journal of Internet Services and Applications, 1(1):7–18, 2010, ISSN 1869-0238. http://dx.doi.org/10.1007/s13174-010-0007-6. 10

[40] Baghban, H., M. Moradi, C. H. Hsu, J. Chou e Y. C. Chung: *Byzantine fault tolerant optimization in federated cloud computing*. Em *2016 IEEE International Conference on Computer and Information Technology (CIT)*, páginas 658–661, Dec 2016. 12, 32

[41] Nair, S. K., S. Porwal, T. Dimitrakos, A. J. Ferrer, J. Tordsson, T. Sharif, C. Sheridan, M. Rajarajan e A. U. Khan: *Towards secure cloud bursting, brokerage and aggregation*. Em *2010 Eighth IEEE European Conference on Web Services*, páginas 189–196, Dec 2010. 12

[42] Grozev, Nikolay e Rajkumar Buyya: *Inter-cloud architectures and application brokering: taxonomy and survey*. Software: Practice and Experience, 44(3):369–390, 2014. 12

[43] Liaqat, Misbah, Victor Chang, Abdullah Gani, Siti Hafizah Ab Hamid, Muhammad Toseef, Umar Shoaib e Rana Liaqat Ali: *Federated cloud resource management: Review and discussion*. Journal of Network and Computer Applications, 77:87 – 105, 2017, ISSN 1084-8045. http://www.sciencedirect.com/science/article/pii/S1084804516302387. 12, 14

[44] Kogias, D. G., M. G. Xevgenis e C. Z. Patrikakis: *Cloud federation and the evolution of cloud computing*. Computer, 49(11):96–99, Nov 2016, ISSN 0018-9162. 13

[45] Goiri, I., J. Guitart e J. Torres: *Characterizing cloud federation for enhancing providers' profit*. Em *2010 IEEE 3rd International Conference on Cloud Computing*, páginas 123–130. IEEE, July 2010. 14

[46] Bouabdallah, R., S. Lajmi e K. Ghedira: *Resources provisioning within cloud federation*. Em *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, páginas 003978–003983. IEEE, Oct 2016. 14

[47] Barroso, Luiz André, Jimmy Clidaras e Urs Hölzle: *The datacenter as a computer: An introduction to the design of warehouse-scale machines, second edition*. Synthesis Lectures on Computer Architecture, 8(3):1–154, 2013. http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024. 18

[48] Tanenbaum, Andrew S. e Herbert Bos: *Distributed Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 1st edição, 1994. 19

[49] Laprie, Jean Claude: *Dependable computing and fault-tolerance*. Digest of Papers FTCS-15, páginas 2–11, 1985. 19

[50] Lee, Peter A e Thomas Anderson: *Fault tolerance: principles and practice*, volume 3. Springer Science & Business Media, 2012. 19, 30

[51] Gil-Tomás, Daniel, Joaquín Gracia-Morán, J Carlos Baraza-Calvo, Luis J Saiz-Adalid e Pedro J Gil-Vicente: *Injecting intermittent faults for the dependability assessment of a fault-tolerant microcomputer system*. IEEE Transactions on Reliability, 65(2):648–661, 2016. 20

[52] Guclu, Sila Ozen, Tanir Ozcelebi e Johan Lukkien: *Distributed fault detection in smart spaces based on trust management*. Procedia Computer Science, 83:66 – 73, 2016, ISSN 1877-0509. http://www.sciencedirect.com/science/article/pii/S1877050916301235. 20

[53] Kopetz, Hermann e Paulo Veríssimo: *Real time and dependability concepts*. Em Mullender, Sape (editor): *Distributed Systems (2Nd Ed.)*, páginas 411–446. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993, ISBN 0-201-62427-3. http://dl.acm.org/citation.cfm?id=302430.302446. 22

[54] Jonsson, Erland: *An integrated framework for security and dependability*. Em *Proceedings of the 1998 Workshop on New Security Paradigms*, NSPW '98, páginas 22–29, New York, NY, USA, 1998. ACM, ISBN 1-58113-168-2. http://doi.acm.org/10.1145/310889.310903. 23

[55] Koren, Israel e C. Mani Krishna: *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edição, 2007, ISBN 0120885255, 9780120885251. 23, 25

[56] Shooman, Martin L.: *Fault Tolerance, Analysis, and Design*, páginas 145–201. John Wiley & Sons, Inc., 2002, ISBN 9780471224600. http://dx.doi.org/10.1002/047122460X.fmatter_indsub. 24

[57] Bilal, Kashif, Osman Khalid, Saif Ur Rehman Malik, Muhammad Usman Shahid Khan, Samee U. Khan e Albert Y. Zomaya: *Fault Tolerance in the Cloud*, páginas 291–300. John Wiley & Sons, Ltd, 2016, ISBN 9781118821930. http://dx.doi.org/10.1002/9781118821930.ch24. 24, 31, 32, 51

[58] Bilal, K., S. U. R. Malik, S. U. Khan e A. Y. Zomaya: *Trends and challenges in cloud datacenters*. IEEE Cloud Computing, 1(1):10–20, May 2014, ISSN 2325-6095. 26, 31

[59] Lamport, Leslie, Robert Shostak e Marshall Pease: *The byzantine generals problem*. ACM Trans. Program. Lang. Syst., 4(3):382–401, julho 1982, ISSN 0164-0925. http://doi.acm.org/10.1145/357172.357176. 27, 29, 37

[60] Schlichting, Richard D. e Fred B. Schneider: *Fail-stop processors: An approach to designing fault-tolerant computing systems*. ACM Transactions on Computer Systems (TOCS), 1(3):222–238, agosto 1983, ISSN 0734-2071. http://doi.acm.org/10.1145/357369.357371. 27

[61] Laranjeira, Luiz A, Miroslaw Malek e Roy Jenevein: *On tolerating faults in naturally redundant algorithms*. Em *Reliable Distributed Systems, 1991. Proceedings., Tenth Symposium on*, páginas 118–127. IEEE, 1991. 27

[62] Lee, P. A. e T. Anderson: *Fault Tolerance: Principles and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edição, 1990, ISBN 0387820779. 30, 31

[63] Garraghan, P., P. Townend e J. Xu: *Byzantine fault-tolerance in federated cloud computing*. Em *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*, páginas 280–285, Dec 2011. 31

[64] Tchana, A., L. Broto e D. Hagimont: *Approaches to cloud computing fault tolerance*. Em *2012 International Conference on Computer, Information and Telecommunication Systems (CITS)*, páginas 1–6. IEEE, May 2012. 31, 32

[65] Ganesh, A., M. Sandhya e S. Shankar: *A study on fault tolerance methods in cloud computing*. Em *2014 IEEE International Advance Computing Conference (IACC)*, páginas 844–849, Feb 2014. 32

[66] Ganga, K. e S. Karthik: *A fault tolerent approach in scientific workflow systems based on cloud computing*. Em *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, páginas 387–390, Feb 2013. 32

[67] Bala, Anju e Inderveer Chana: *Fault tolerance-challenges, techniques and implementation in cloud computing.* IJCSI International Journal of Computer Science Issues, 9(1):1694–0814, 2012. 32

[68] Agarwal, H. e A. Sharma: *A comprehensive survey of fault tolerance techniques in cloud computing.* Em *2015 International Conference on Computing and Network Communications (CoCoNet)*, páginas 408–413. IEEE, Dec 2015. 32, 34

[69] Patra, Prasenjit K., Harshpreet Singh e Gurpreet Singh: *Fault tolerance techniques and comparative implementation in cloud computing.* International Journal of Computer Applications, 64(14), 2013. `https://search.proquest.com/docview/1288810772?accountid=26646`, Copyright - Copyright Foundation of Computer Science 2013; Last updated - 2013-09-13. 32

[70] Singh, Amritpal e Supriya Kinger: *An efficient fault tolerance mechanism based on moving averages algorithm.* International Journal of Advanced Research in Computer Science and Software Engineering, 3(7), 2013. 32

[71] Vallee, G., K. Charoenpornwattana, C. Engelmann, A. Tikotekar, C. Leangsuksun, T. Naughton e S. L. Scott: *A framework for proactive fault tolerance.* Em *2008 Third International Conference on Availability, Reliability and Security*, páginas 659–664, 2008. 32

[72] Hasan, Moin e Major Singh Goraya: *Fault tolerance in cloud computing environment: A systematic survey.* Computers in Industry, 99:156 – 172, 2018, ISSN 0166-3615. `http://www.sciencedirect.com/science/article/pii/S0166361517304438`. 34

[73] Chalermarrewong, T., T. Achalakul e S. C. W. See: *The design of a fault management framework for cloud.* Em *2012 9th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, páginas 1–4. IEEE, May 2012. 35

[74] Zhao, W., P. M. Melliar-Smith e L. E. Moser: *Fault tolerance middleware for cloud computing.* Em *2010 IEEE 3rd International Conference on Cloud Computing*, páginas 67–74. IEEE, July 2010. 36, 62

[75] Zhang, Y., Z. Zheng e M. R. Lyu: *Bftcloud: A byzantine fault tolerance framework for voluntary-resource cloud computing.* Em *2011 IEEE 4th International Conference on Cloud Computing*, páginas 444–451. IEEE, July 2011. 36, 62

[76] Veronese, G. S., M. Correia, A. N. Bessani, L. C. Lung e P. Verissimo: *Efficient byzantine fault-tolerance.* IEEE Transactions on Computers, 62(1):16–30, Jan 2013, ISSN 0018-9340. 36, 37, 62

[77] Garraghan, PM, PM Townend e J XU: *Using byzantine fault-tolerance to improve dependability in federated cloud computing.* International Journal of Software and Informatics, 7(2):221 – 237, July 2013. `http://eprints.whiterose.ac.uk/76286/`. 36, 37, 62

[78] Jhawar, R., V. Piuri e M. Santambrogio: *Fault tolerance management in cloud computing: A system-level perspective*. IEEE Systems Journal, 7(2):288–297, June 2013, ISSN 1932-8184. 36, 37, 62

[79] Das, P. e P. M. Khilar: *Vft: A virtualization and fault tolerance approach for cloud computing*. Em *2013 IEEE Conference on Information Communication Technologies*, páginas 473–478. IEEE, April 2013. 36, 37, 62

[80] Joshi, Sagar C. e Krishna M. Sivalingam: *Fault tolerance mechanisms for virtual data center architectures*. Photonic Network Communications, 28(2):154–164, 2014, ISSN 1387-974X. http://dx.doi.org/10.1007/s11107-014-0463-1. 36, 37, 62

[81] Bala, Anju e Inderveer Chana: *Intelligent failure prediction models for scientific workflows*. Expert Systems with Applications, 42(3):980 – 989, 2015, ISSN 0957-4174. http://www.sciencedirect.com/science/article/pii/S0957417414005533. 36, 38, 62

[82] Chen, W., R. F. da Silva, E. Deelman e T. Fahringer: *Dynamic and fault-tolerant clustering for scientific workflows*. IEEE Transactions on Cloud Computing, 4(1):49–62, Jan 2016, ISSN 2168-7161. 36, 38, 62

[83] Jhawar, R., V. Piuri e M. Santambrogio: *A comprehensive conceptual system-level approach to fault tolerance in cloud computing*. Em *2012 IEEE International Systems Conference SysCon 2012*, páginas 1–5. IEEE, March 2012. 37

[84] Jhawar, R. e V. Piuri: *Fault tolerance management in iaas clouds*. Em *2012 IEEE First AESS European Conference on Satellite Telecommunications (ESTEL)*, páginas 1–6. IEEE, Oct 2012. 37

[85] Guan, Q., Z. Zhang e S. Fu: *Ensemble of bayesian predictors for autonomic failure management in cloud computing*. Em *2011 Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, páginas 1–6. IEEE, July 2011. 38

[86] Moura, B. R.: *Controlador de SLA em uma plataforma de nuvens federadas*. Tese de Mestrado, Programa de Pós-Graduação em Informática, Departamento de Ciência de Computação, Universidade de Brasília, 2017. http://repositorio.unb.br. 40

[87] Rosa, M. J. F.: *Predição de tempo e dimensionamento de recursos para workflows científicos em nuvens federadas*. Tese de Mestrado, Programa de Pós-Graduação em Informática, Departamento de Ciência de Computação, Universidade de Brasília, 2017. http://repositorio.unb.br. 40

[88] Vergara, G. F.: *Arquitetura de um controlador de elasticidade para nuvens federadas*. Tese de Mestrado, Programa de Pós-Graduação em Informática, Departamento de Ciência de Computação, Universidade de Brasília, 2017. http://repositorio.unb.br. 40, 41

[89] Azevedo, D. R. e T. B. Freitas Júnior: *BioCirrus: uma nova política de armazenamento para a plataforma BioNimbuZ de nuvem federada.* `http://bdm.unb.br/handle/10483/13199`, 2015. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 40, 41

[90] Costa, H. H. P. M.: *Controle de Acesso na Plataforma de Nuvem Federada BioNimbuZ.* `http://bdm.unb.br/handle/10483/11141`, 2015. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 40

[91] Ramos, V. A.: *Um sistema gerenciador de workflows científicos para a plataforma de nuvens federadas BioNimbuZ.* `http://bdm.unb.br/handle/10483/13145`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 40

[92] Barreiros Júnior, W. O.: *Escalonador de tarefas para o plataforma de nuvens federadas BioNimbuZ usando beam search iterativo multiobjetivo.* `http://bdm.unb.br/handle/10483/13146`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 40, 41

[93] Souza, F. L.: *Uma Nova Arquitetura para Plataforma de Nuvens Federadas.* Tese de Mestrado, Programa de Pós-Graduação em Informática, Departamento de Ciência de Computação, Universidade de Brasília, 2018. `http://repositorio.unb.br`. 40, 41

[94] Stoica, Ion, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek e Hari Balakrishnan: *Chord: A scalable peer-to-peer lookup protocol for internet applications.* IEEE/ACM Trans. Netw., 11(1):17–32, fevereiro 2003, ISSN 1063-6692. `http://dx.doi.org/10.1109/TNET.2002.808407`. 40

[95] Oliveira, Gabriel SS de, Edward Ribeiro, Diogo A Ferreira, Aletéia PF Araújo, Maristela T Holanda e Maria Emilia MT Walter: *Acosched: A scheduling algorithm in a federated cloud infrastructure for bioinformatics applications.* Em *Bioinformatics and Biomedicine (BIBM), 2013 IEEE International Conference on*, páginas 8–14. IEEE, 2013. 41

[96] Lima, D., B. Moura, G. Oliveira, E. Ribeiro, A. Araujo, M. Holanda, R. Togawa e M. E. Walter: *A storage policy for a hybrid federated cloud platform: A case study for bioinformatics.* Em *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, páginas 738–747. IEEE, May 2014. 41

[97] Gallon, Ricardo, Maristela Holanda, Aletéia Araújo e Maria E Walter: *Storage policy for genomic data in hybrid federated clouds.* Em *Brazilian Symposium on Bioinformatics*, páginas 107–114. Springer, 2014. 41

[98] Santos, L. F. N.: *Novo Serviço de Armazenamento na Plataforma de Nuvem Federada BioNimbuZ.* `http://bdm.unb.br/handle/`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 41

[99] Fowler, Martin e James Lewis: *Microservices (accessed on 25/03/2014). 2014.* URL: http://martinfowler.com/articles/microservices.html, 2014. 42, 45

[100] Feo, Thomas A. e Mauricio G. C. Resende: *Greedy randomized adaptive search procedures*. Journal of Global Optimization, 6(2):109–133, 1995, ISSN 1573-2916. http://dx.doi.org/10.1007/BF01096763. 44

[101] Junqueira, Flavio e Benjamin Reed: *ZooKeeper: distributed process coordination*. " O'Reilly Media, Inc.", 2013. 45

[102] Leonard Richardson, Sam Ruby, David Heinemeier Hansson: *Restful Web Services*. O'Reilly Media, 1ª edição, 2007, ISBN 0596529260,9780596529260. 52