UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Vinicius Santos Andrade

Uma Taxonomia de Sistemas de Tolerância a Falhas em Ambientes de Computação em Nuvem open source

São José do Rio Preto

2019

Vinicius Santos Andrade

Uma Taxonomia de Sistemas de Tolerância a Falhas em Ambientes de Computação em Nuvem open source

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós- Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Orientador: Prof. Dr. Aleardo Manacero Junior

São José do Rio Preto

2019

A553t

Andrade, Vinicius Santos
Uma Taxonomia de Sistemas de Tolerância a Falhas em Ambientes de Computação em Nuvem open source / Vinicius Santos Andrade. -- São José do Rio Preto, 2019
117 p. : tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientador: Aleardo Manacero Junior

1. Ciência da computação. 2. Sistemas distribuídos.. 3. Tolerância a falhas. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas. São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Vinicius Santos Andrade

Uma Taxonomia de Sistemas de Tolerância a Falhas em Ambientes de Computação em Nuvem open source

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós- Graduação em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de São José do Rio Preto.

Banca examinadora:

Prof. Dr. Aleardo Manacero Junior
UNESP – São José do Rio Preto
Orientador

Prof. Dr. Rafael Pasquini
UFU – Uberlândia

Prof. Dr. Rodolfo Ipolito Meneguetti
IFSP – Catanduva

São José do Rio Preto
04 de janeiro de 2019

À minha mãe Nilvani.

Agradecimentos

Primeiramente, a Deus, pela minha vida e por tudo que tens feito nela.

Ao meu orientador Prof. Dr. Aleardo Manacero Júnior, pela confiança e ensinamentos transmitidos no decorrer do mestrado.

A minha mãe, Nilvani, pelo apoio e amor incondicional, manifestado ao longo da minha vida.

A minha amiga e namorada, Ana Caroline Braguetto dos Santos, pelo apoio e paciência.

A minha tia, Renata Cristina Lopes Andrade, pelo incentivo e apoio.

Aos meus amigos, Luiz Felipe de Camargo, Ewerson Assunção Jahara, João Antonio Magri Rodrigues e Hethini do Nascimento Ribeiro, pela ajuda técnica e pela amizade.

A Prof[a]. Dr[a]. Roberta Spolon, que no câmpus da Unesp de Bauru, deu todo apoio necessário desde o início do mestrado.

Ao Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto (IBILCE-Unesp) pela oportunidade, aos professores e funcionário e a todos, que direta ou indiretamente, contribuíram para com este trabalho.

“Deixe o mundo um pouco melhor do que encontrou.”

- Robert Stephenson Smyth Baden-Powell.

Resumo

Computação em nuvem oferece serviços de computação por meio da internet, utilizando máquinas virtuais. A garantia de elasticidade e pagamento sobre demanda são os principais motivos para atrair pessoas para aderirem a esta tecnologia. Infelizmente, existem diversos fatores que podem ocasionar erros que poderão gerar falha(s) no serviço, sendo que muitas vezes tais falhas acabam ocasionando perda de dados, por exemplo. Para evitar transtornos além da própria falha do serviço é importante que o suporte de tolerância a falhas em computação em nuvem seja robusto, garantindo que o ambiente se recupere na íntegra, de maneira ágil e sem a necessidade de interferência do cliente ou até do administrador. Um ambiente de tolerância a falhas envolve inúmeras variáveis, tanto entre causas da falha como técnicas de tratamento, fazendo com que uma falha possa ser suprida de *N* maneiras. Na literatura é possível encontrar muitos trabalhos relacionados a tolerância falhas em nuvem, o que causa grande dificuldade em identificar pontos que podem ser melhorados em relação à falhas no ambiente. Neste trabalho é apresentada uma sistematização das inúmeras propostas para tratamento de falhas em uma infraestrutura de computação em nuvem encontradas na literatura, criando uma classificação por origem da falha e forma de tratamento, na qual se aponta ainda pontos fortes e fracos de cada abordagem. Essa sistematização serviu de base para a construção de um *web site* com informações objetivas sobre as soluções identificadas, formando um protótipo de um ambiente de recomendação para administradores de ambientes de computação em nuvem.

Palavras-chaves: Computação em nuvem, máquinas virtuais, tolerância a falha, revisão.

Abstract

Cloud computing offers computer services over the internet using virtual machines. The guarantee of elasticity and payment on demand are the main reasons attracting people to join the technology. Unfortunately, there are a number of factors that may cause errors that could lead to service failure(s), with such failures often resulting in loss of data, for example. In order to avoid disruptions beyond the fault itself, it is important that the fault tolerance support is robust, ensuring that the environment is recovered in full, in an agile manner and without the need for customer, or even system's manager, interference. A fault tolerant environment deals with a myriad of variables related to the failure's cause or its treatment, implying that a failure can be managed through *N* approaches. One can find several papers describing approaches to solve cloud computing failures, which causes great difficulty in identifying points that can be improved in relation to the flaws in the environment. This work presents a systematic review of the methods for fault tolerance in a cloud computing infrastructure that are found in the literature, creating a classification based in the failure's cause and the solution's approach, including their strong and weak points. This systematization became the basis of a website containing objective information about the identified solutions, creating a prototype of a recommendation system for cloud computing managers.

Keywords: Cloud computing, virtual machines, fault tolerance, survey.

Lista de ilustrações

Figura 1 – Exemplo de Computação em Nuvem (GUPTA; GUPTA, 2012). 22
Figura 2 – Definição de computação em nuvem segundo NIST (VERAS, 2015). 24
Figura 3 – Exemplo de implementação de nuvem pública, privada e híbrida (GUPTA; GUPTA, 2012). 26
Figura 4 – Arquitetura de computação em nuvem (ZHANG; CHENG; BOUTABA, 2010). 27
Figura 5 – Arquitetura do OpenStack (REDHAT, 2017). 30
Figura 6 – Arquitetura básica do OpenNebula (OPENNEBULA, 2017). 32
Figura 7 – Hospedagem de três máquinas virtuais com diferentes sistemas operacionais (BUYYA; BROBERG; GOSCINSKI, 2011). 33
Figura 8 – Cadeia de ameaças provenientes de uma falha (AVIZIENIS et al., 2004). . . 36
Figura 9 – Sistema de Injeção de Falhas (HSUEH; TSAI; IYER, 1997). 43
Figura 10 – Gráfico com total de artigos obtidos subi-divididos por base de dados. . . . 57
Figura 11 – Quantidade de artigos publicados segundo origem da falha. 58
Figura 12 – Técnicas utilizadas para falhas tolerar falhas considerando todas propostas estudadas. 59
Figura 13 – Tipos de falhas em ambientes de computação em nuvem atualizado. 60
Figura 14 – Página inicial do *site* acessado usando um celular. 105
Figura 15 – Página inicial do *site* acessado usando um *desktop*. 105
Figura 16 – Conteúdo da página “Falhas relacionadas a comunicação” do *site* acessado usando um *desktop*. 106

Lista de tabelas

Tabela 1 – Comparação entre arquitetura de computação em nuvem em relação a outras arquiteturas. 23
Tabela 2 – Dados referentes aos *hypervisors*, gestores e plataformas de computação em nuvem. 35
Tabela 3 – Fases para aplicação das técnicas de tolerância a falhas. 38
Tabela 4 – Resumo das técnicas de tolerância a falhas para sistemas específicos. 41
Tabela 5 – Strings de busca utilizadas para Cloud Computing 50
Tabela 6 – Número de artigos obtidos em cada base de acordo com cada string de busca SG*n* 51
Tabela 7 – Resultados por ano utilizando SG5 (Setembro de 2017) 51
Tabela 8 – Strings de busca utilizadas para Openstack 52
Tabela 9 – Número de artigos obtidos em cada base de acordo com cada string de busca SS*n* (Setembro de 2017) 52
Tabela 10 – Resultados por ano utilizando SS3 (Setembro de 2017) 53
Tabela 11 – Strings de busca utilizadas para Cloudstack 53
Tabela 12 – Número de artigos obtidos em cada base de acordo com cada string de busca SCS*n* 54
Tabela 13 – Resultados por ano utilizando SCS3 (Setembro de 2017) 54
Tabela 14 – *Strings* de busca utilizadas para OpenNebula 55
Tabela 15 – Número de artigos obtidos em cada base de acordo com cada string de busca SN*n* (Outubro de 2017) 55
Tabela 16 – Resultados por ano utilizando SN3 (Setembro de 2017) 56
Tabela 17 – Total de artigos obtidos com cada *string* escolhida 56
Tabela 18 – Resumo do sistema IGG. 63
Tabela 19 – Resumo do $HySARC^2$. 63
Tabela 20 – Resumo do sistema SkyCDS. 64
Tabela 21 – Resumo sistema DRAGOS. 65
Tabela 22 – Resumo Protótipo para tratamento automático de anomalias. 66
Tabela 23 – Resumo Tahoe-LAFS 67
Tabela 24 – Resumo do sistema SwiftER. 67
Tabela 25 – Resumo do sistema de armazenamento em multicamadas. 68
Tabela 26 – Resumo do sistema GFS. 68
Tabela 27 – Resumo do algoritmo híbrido baseado em MapReduce. 69
Tabela 28 – Resumo do sistema Morpho. 69
Tabela 29 – Resumo do sistema DCR2S. 70
Tabela 30 – Resumo do sistema FIR3. 71

Tabela 31 – Resumo do sistema MIFAS. 71
Tabela 32 – Resumo do sistema baseado no princípio 80/20. 72
Tabela 33 – Resumo do Middleware de nuvem para garantir desempenho e alta disponibilidade de aplicativos soft em tempo real. 73
Tabela 34 – Resumo do sistema mOSAIC. 73
Tabela 35 – Resumo do sistema Pilot-Data. 74
Tabela 36 – Resumo do sistema ONHelp. 75
Tabela 37 – Resumo do sistema. 76
Tabela 38 – Resumo do sistema SymVirt. 77
Tabela 39 – Resumo do sistema Mantis 78
Tabela 40 – Resumo do sistema FT-FW. 79
Tabela 41 – Resumo do sistema de controle de acesso avançado. 80
Tabela 42 – Resumo do Protótipo 80
Tabela 43 – Resumo dos sistemas pFTree-Ext e pFTree-Wt. 81
Tabela 44 – Resumo do Auto Healing Service Framework. 82
Tabela 45 – Resumo do sistema Mayflies 83
Tabela 46 – Resumo do sistema TOPSIS. 84
Tabela 47 – Resumo do sistema Group-U. 85
Tabela 48 – Resumo do sistema DORADO. 86
Tabela 49 – Resumo do sistema COSCAet-FT. 87
Tabela 50 – Resumo do sistema Cloud Inter-operation Toolkit. 87
Tabela 51 – Resumo do sistema OpenADN. 88
Tabela 52 – Resumo do sistema SpritNet. 88
Tabela 53 – Resumo do sistema FASA. 89
Tabela 54 – Resumo do sistema FTM. 90
Tabela 55 – Resumo do sistema FT-FC. 91
Tabela 56 – Resumo do sistema Promethee Scheduler. 92
Tabela 57 – Resumo da proposta de arquitetura de *cluster* virtual tolerante a falhas. . . . 93
Tabela 58 – Resumo do sistema CACS. 94
Tabela 59 – Resumo sistema Unibus. 94
Tabela 60 – Resumo do sistema Availability Knob. 95
Tabela 61 – Resumo do protótipo para sistema de migração de VM. 96
Tabela 62 – Resumo do sistema FTESW. 97
Tabela 63 – Resumo do sistema de detecção e mitigação de falhas. 97
Tabela 64 – Resumo do sistema de tolerância a falhas energética para computação de alto desempenho. 98
Tabela 65 – Resumo do sistema Zurmo Native. 99
Tabela 66 – Resumo do sistema PFT-CCKP. 100

Tabela 67 – Resumo do protótipo de replicação de VM para melhorar capacidade de sobrevivência de aplicativos de missão crítica. 102
Tabela 68 – Resumo do sistema HAStack. 103
Tabela 69 – Resumo do sistema FTStack. 104
Tabela 70 – Qualis e Número de citações dos artigos utilizados na última etapa da RSL. 117

Lista de abreviaturas e siglas

2PC	Protocolo de Confirmação de 2 fases (2PC)
AFTRC	Adaptive Fault Tolerance model in Real Time Cloud Computing
ACM	Association of Computing Machinery
API	Application Programming Interface
AWS	Amazon Web Service
BVM	Benign VM migration
CDS	Content Delivery Systems
CEO	Chief Executive Officer
CPU	Unidade Central de Processamento
DMTCP	Distributed MultiThreadedCheckpointing
EC2	Elastic Compute Cloud
E/S	Entrada/Saída
FASA	Future Automation System Architecture
FTESW	Fault-Tolerant Elastic Scheduling Algorithm for Workflow
FT-FC	Fault-tolerance in Federated Clouds
FTM	Fault Tolerance Manager
FTWS	Fault Tolerant Work flow Scheduling
GFM	Global Fault Manager
GPU	Graphics Processing Unit
HA	High Availability
HDD	Hard Disk Drive
HDFS	Hadoop Distributed File System
IAAS	Infraestructure as a Service

ID	Identidade
IEEE	Institute of Electrical and Electronics Engineering
KVM	Kernel-based Virtual Machine
LFM	Local Fault Manager
LLFT	Low Latency Fault Tolerance
MAC	Media Access Control
MIFAS	Medical Image File Accessing System
MPI	Message Passing Interface
MTBF	Mean Time Between Failures
MTTF	Mean Time to Failure
MTD	Moving Target Defense
MTTR	Mean Time to Repair
NAT	Network Address Translation
NIST	National Institute of Standards and Technology
NASA	National Aeronautics and Space Administration
NFV	Network Function Virtualization
PAAS	Plataform as a Service
PHP	Hypertext Processor
PFT-CCKP	Proactive Fault Tolerance Mechanism for Data Center Network
POP	Points of Presence
QoS	Quality of Service
SAAS	Software as a Service
RSL	Revisão Sistemática da Literatura
S3	Amazon Simple Storage Service
SAP	Systeme Anwendungen und Produkte in der Datenyarbeitung
SDN	Software Defined Networking

SFI	Software Fault Injection
SMM	Modelo métrico estatístico
SQL	Structured Query Language
SSH	Secure Shell
SwifER	Swift Erasure
Thaoe-LAFS	Tahoe-Least Authority File System
TI	Tecnologia da Informação
vCDN	Virtual Content Delivery Network
VDI	Virtual Desktop Infraestructure
VEE	Virtual Execution Environments
VM	Virtual Machine
VMI	Virtual Machine Introspection
VMM	Virtual Machine Monitor
WWW	World Wide Web

Sumário

1 INTRODUÇÃO 19
1.1 Motivação 19
1.2 Objetivos 20
1.3 Organização do texto 20

2 FUNDAMENTAÇÃO TEÓRICA 22
2.1 Computação em nuvem 22
2.2 Modelos de Serviços de Computação em nuvem 25
2.3 Arquitetura de computação em nuvem 25
2.4 Serviços de computação em nuvem 27
2.4.1 *Amazon AWS* 27
2.4.2 *Microsoft Windows Azure plataform* 28
2.4.3 Google App Engine 29
2.4.4 IBM *BlueMix* 29
2.5 Gestores de computação em nuvem 29
2.5.1 *Eucalyptus* 29
2.5.2 *OpenStack* 30
2.5.3 OpenNebula 32
2.5.4 Apache CloudStack 32
2.6 Hypervisors 33
2.6.1 *VMware* 33
2.6.2 *XenServer* 34
2.6.3 *KVM* 34
2.6.4 *Microsoft Hyper-V* 34
2.7 Sistematização sobre computação em nuvem 35
2.8 Tolerância a falhas 35
2.8.1 Conceitos básicos 36
2.8.2 Modelos de falhas 37
2.8.3 Técnicas de tolerância a falhas 38
2.8.4 Tolerância a falhas na computação em nuvem 38
2.8.5 Técnicas reativas de tolerância a falhas 39
2.8.6 Técnicas proativas de tolerância a falhas 40
2.8.7 Módulos de tolerância a falhas 41
2.8.8 Injeção de falhas no ambiente 42
2.9 Trabalhos relacionados 43

2.10 Considerações Finais . . . 44

3 REVISÃO SISTEMÁTICA . . . 46
3.1 Metodologia . . . 46
3.1.1 Bases científicas . . . 49
3.2 Resultados obtidos na primeira etapa da RSL . . . 49
3.2.1 Resultados para *Cloud Computing* . . . 50
3.2.2 Resultados para Apache Openstack . . . 52
3.2.3 Resultados para Cloudstack . . . 53
3.2.4 Resultados para OpenNebula . . . 55
3.3 Resultados obtidos na segunda etapa da RSL . . . 57
3.4 Resultados obtidos através da terceira etapa da RSL . . . 58
3.5 Considerações finais . . . 60

4 ANÁLISE DE SISTEMAS PARA TOLERÂNCIA A FALHAS EM AMBIENTES DE COMPUTAÇÃO EM NUVEM . . . 61
4.1 Metodologia de classificação . . . 61
4.2 Falhas relacionadas a dados . . . 62
4.2.1 Uso de *Checkpointing* . . . 62
4.2.2 Migração Preemptiva . . . 63
4.2.3 Migração de trabalho . . . 64
4.2.4 Uso de Autocura . . . 65
4.2.5 Uso de Replicação . . . 66
4.3 Falhas relacionadas a comunicação . . . 74
4.3.1 Uso de *Checkpointing* . . . 74
4.3.2 Uso de Migração preemptiva . . . 77
4.3.3 Uso de Autocura . . . 81
4.3.4 Uso de Replicação . . . 82
4.4 Falhas relacionadas a processos . . . 89
4.4.1 Uso de Replicação . . . 89
4.4.2 Uso de *Checkpointing* . . . 93
4.4.3 Uso de Autocura . . . 95
4.4.4 Uso de Migração de Trabalho . . . 96
4.5 Falhas relacionadas a virtualização . . . 99
4.5.1 Uso de *Checkpointing* . . . 99
4.5.2 Uso de Replicação . . . 100
4.5.3 Uso de Autocura . . . 102
4.5.4 Uso de Migração preemptiva . . . 103
4.6 Web site para consulta guiada . . . 104
4.7 Considerações finais . . . 106

5 CONCLUSÕES E TRABALHOS FUTUROS 107
5.1 Conclusões 107
5.2 Trabalhos Futuros 108

REFERÊNCIAS 109

APÊNDICE A – QUALIS E NÚMERO DE CITAÇÕES DOS ARTIGOS UTILIZADOS NA ETAPA FINAL DA RSL . 116

1 Introdução

Computação em nuvem é um modelo de sistema computacional que permite acesso via rede, por demanda, para um *pool* de recursos computacionais compartilhados (por exemplo, redes, servidores, armazenamento, aplicativos e serviços), que podem rapidamente ser provisionados e liberados com mínimo esforço de gerenciamento ou interação do provedor de serviços (MELL; GRANCE et al., 2011).

O que se tem observado é que clientes migram para computação em nuvem pelas inúmeras vantagens que o ambiente oferece, como por exemplo, alta disponibilidade, flexibilidade e recursos sob demanda. Porém, para que isso seja realidade é fundamental que o ambiente seja tolerante a falhas, isto é, que não se perca tempo de execução por problemas com o sistema ou parte dele.

De modo conceitual tem-se que falhas são originadas por erros, sendo que inúmeros erros podem ocorrer até o momento que uma falha trave completamente o sistema, por exemplo. Como falhas são inevitáveis é importante que se entenda como ocorrem e como podem ser evitadas ou corrigidas. Além disso, falhas podem ter origens e soluções muito diferentes para cada situação, o que tem gerado um grande número de contribuições de soluções que se adéquam de modo diferente a casos específicos. O trabalho aqui apresentado avalia essas contribuições, com base em uma proposta de metodologia de classificação de técnicas de tolerância a falhas, que permita a identificação da técnica mais adequada ao caso específico de um administrador de sistemas de computação em nuvem.

1.1 Motivação

Para a computação em nuvem, vista como um negócio, é crucial que o provedor cumpra o contrato de nível de serviço (SLA) estabelecido com seus clientes. O não cumprimento desse contrato pode acarretar em uma degradação da qualidade do serviço hospedado e, consequentemente, na insatisfação de clientes. Serviços oferecidos por meio da nuvem atendem um grande nicho de mercado, desde usuários "comuns" até grandes empresas como, por exemplo, a Netflix, que em agosto de 2008 iniciou a migração, concluída em janeiro de 2016, de seus serviços de *streaming* para a nuvem da Amazon Web Services (AWS) (NETFLIX, 2017).

Assim como qualquer outro sistema computacional, ambientes de computação em nuvem estão suscetíveis a erros, os quais podem levar a falhas de serviço. Tais falhas podem gerar inúmeros prejuízos, tanto para o cliente quanto para o fornecedor de serviços de computação em nuvem. Portanto, é importante que o serviço fornecido possa ocorrer sem interrupções ou prejuízos de desempenho, ou seja, é importante que o sistema seja tolerante a falhas.

A busca em manter um ambiente tolerante a falhas deve ser constante, visto que existem inúmeras falhas que podem ser resolvidas de inúmeras formas. Isso tem sido trabalhado de modos diversos, com propostas que muitas vezes não trazem um ganho real em relação a outros métodos já existentes, o que é um erro. Infelizmente esse erro tem sido repetido continuamente, o que gerou uma enorme quantidade de abordagens para o tratamento de falhas, causando dificuldades em identificar quais técnicas devem ser usadas.

Neste trabalho é apresentado um *web site* de busca guiada desenvolvido com base em informações extraídas a partir de uma revisão sistemática da literatura. Com a revisão criou-se um *survey* da área de tolerância a falhas em computação em nuvem, contendo soluções para tratar falhas em diferentes ambientes e situações. A forma sintetizada deste *survey* permitiu a criação do *web site* de busca[1].

1.2 Objetivos

Considerando que existem vários ambientes distintos de computação em nuvem e diversas abordagens sugeridas para o tratamento de tolerância a falhas para tais ambientes, o objetivo do projeto apresentado aqui é uma sistematização das propostas de tolerância a falhas que resulte em um *survey* e na geração de um ambiente de consulta guiada para leitura das técnicas relevantes para cada caso.

Com a sistematização de soluções propostas se pretende obter um documento que possa ser utilizado por administradores de sistemas de computação em nuvem. Esse documento, ao apontar que técnicas se aplicam em que problemas, incluindo vantagens e desvantagens de cada técnica, facilita o processo de suporte para esses administradores no tratamento de falhas.

Por fim, com as informações obtidas, será gerado um *web site* para armazenar algumas dessa informações. Esse ambiente virtual é um protótipo de um ambiente de recomendação para tolerância a falhas em nuvem. Em um primeiro momento a recomendação ocorrerá por meio de busca guiada, para que administradores de sistemas de computação em nuvem possam ter melhores condições para gerenciar o tratamento de falhas em seus ambientes.

1.3 Organização do texto

O próximo capítulo aborda os temas computação em nuvem e tolerância a falhas, com suas principais definições. Já no Capítulo 3 se apresenta a metodologia usada para a revisão sistemática de literatura (RSL), bem como estatísticas globais obtidas de sua aplicação.

A revisão sistemática das soluções identificadas no capítulo anterior é feita então no Capítulo 4, em que cada solução é descrita, apresentando que falhas tratam, de que forma e

[1] https://quejo18.github.io/

quais são seus pontos fortes e fracos. Neste capítulo também se introduz o *web site* para consulta criado.

Por fim são apresentadas as conclusões deste trabalho e indicações de possíveis trabalhos futuros a ele relacionados no Capítulo 5. Além do corpo básico da dissertação é incluído um Apêndice em que se lista o número de citações que cada trabalho referenciado no Capítulo 4 recebeu e, quando disponível, o Qualis da publicação em que o trabalho foi publicado.

2 Fundamentação teórica

Este capítulo apresenta conceitos teóricos importantes para a fundamentação e entendimento de computação em nuvem (seção 2.1), suas classes de serviços (seção 2.2), sua arquitetura (seção 2.3), alguns provedores de serviços (2.4), gestores de nuvem (seção 2.5), ferramentas de virtualização (seção 2.6) e tolerância a falhas (seção 2.8).

2.1 Computação em nuvem

O termo computação em nuvem foi introduzido em 2006, quando o então CEO (*Chief Executive Officer*) da Google, Eric Schmidt, utilizou o termo para descrever os serviços da própria empresa, e pouco tempo depois, quando a Amazon utilizou o mesmo termo para lançar seu serviço EC2. O termo foi de fato popularizado com o trabalho de George Gilder intitulado *"The Information Factories"* (GILDER, 2006). A Figura 1 ilustra um ambiente de computação em nuvem. Nele os usuários se conectam a nuvem com seus próprios computadores ou aparelhos portáteis, usando a *internet*.

Figura 1 – Exemplo de Computação em Nuvem (GUPTA; GUPTA, 2012).

(Traduzido e adaptado pelo autor)

A computação em nuvem é um termo abrangente para descrever uma categoria de serviços de computação sob demanda (*on-demand*) que eram inicialmente oferecidos por provedores

comercias, como Amazon, Google e Microsoft. Sua infraestrutura de computação é vista como uma "nuvem", em que as empresas e indivíduos acessam aplicativos de qualquer lugar do mundo por meio da internet (BUYYA et al., 2009). A Tabela 1 apresenta um comparativo entre as arquiteturas de TI, em que se observa que computação em nuvem se diferencia pelo pagamento apenas do que for utilizado, sem necessidade de aquisição de equipamentos.

Tabela 1 – Comparação entre arquitetura de computação em nuvem em relação a outras arquiteturas.

Arquitetura	Tecnologia	Economia	Modelo de negócio
Mainframe	Computação centralizada	Otimizado para eficiência por causa do alto custo	Alto custo de *hardware* e *software*
Cliente/ Servidor	Computação distribuída	Otimizado para agilidade devido ao baixo custo	Licença perpétua para S.O e aplicativos
Computação em Nuvem	Grandes datacenters	Otimizado para eficiência e agilidade	Paga pelo uso

Fonte: (HARMS; YAMARTINO, 2010) (tradução nossa).

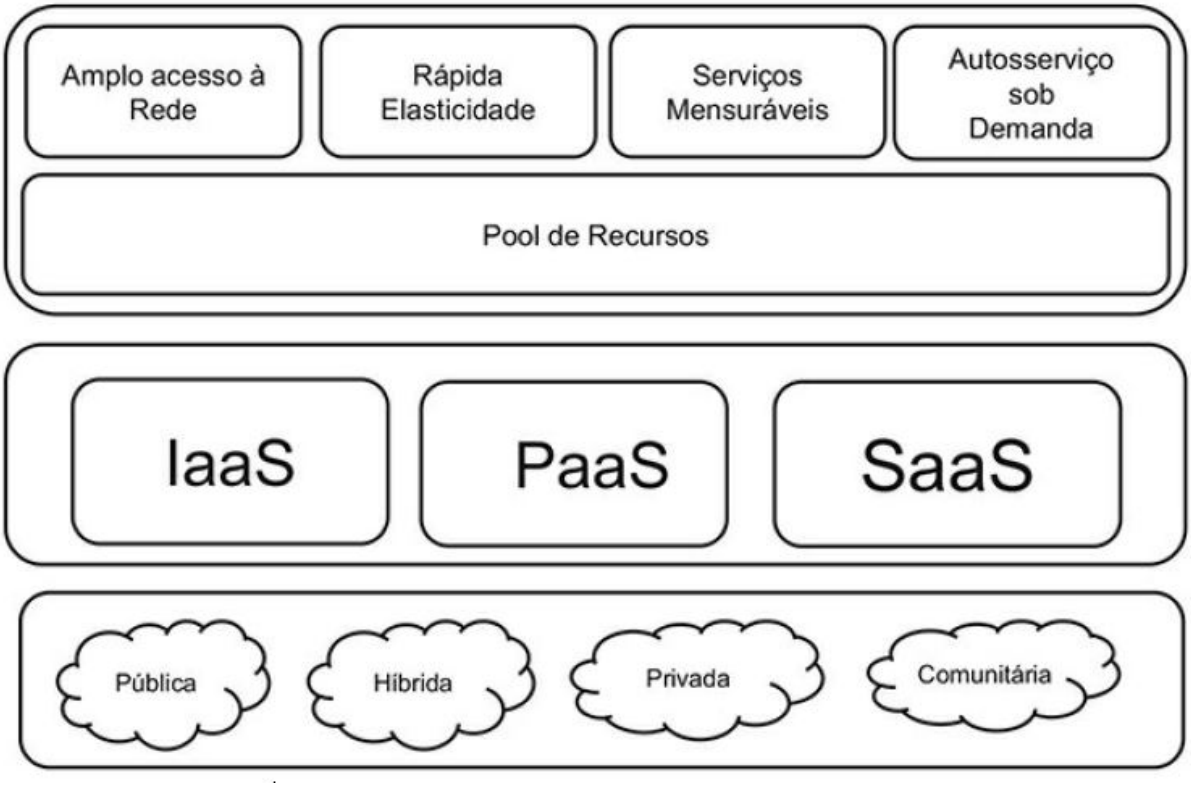
É essencial que um ambiente em nuvem apresente as seguintes características essenciais (MELL; GRANCE et al., 2011):

- **Atendimento sob demanda:** funcionalidades computacionais são providas automaticamente sem a interação humana com o provedor de serviço.
- **Amplo acesso a serviço de rede:** recursos computacionais estão disponíveis na *internet* e são acessados via mecanismos padronizados, para que possam ser utilizados por dispositivos heterogêneos (celulares, notebooks, estações de trabalho, entre outros).
- ***Pool* de recursos:** recursos computacionais (físicos ou virtuais) do provedor são alocados para utilização conforme a demanda do usuário. A alocação é feita de forma dinâmica.
- **Elasticidade rápida:** os recursos computacionais devem ser rápidos e elasticamente providos, bem como rapidamente liberados (em certos casos automaticamente). O usuário dos recursos deve ter a impressão de que ele possui recursos ilimitados, que podem ser adquiridos (comprados) a qualquer momento e em qualquer quantidade.
- **Serviços mensuráveis:** os sistemas de gerenciamento utilizados pela computação em nuvem controlam, otimizam e monitoram automaticamente os recursos para cada tipo de serviço (processamento, largura de banda, armazenamento, entre outros). O monitoramento garante transparência para o provedor de serviços e consumidor do serviço utilizado.

A Figura 2 exemplifica os conceitos referentes à computação em nuvem (MELL; GRANCE et al., 2011 apud VERAS, 2015), com destaque para a camada de serviços (IaaS, PaaS

e SaaS) que é descrita a seguir, na Seção 2.2, e seus modelos de arquitetura (pública, privada, etc) serão descritos na Seção 2.3.

Figura 2 – Definição de computação em nuvem segundo NIST (VERAS, 2015).



(Traduzido e adaptado pelo autor)

Para que o modelo de computação em nuvem atenda as necessidades e expectativas dos consumidores é essencial que as ofertas de serviços em nuvem sejam (BUYYA; BROBERG; GOSCINSKI, 2011):

- ***Self-service*:** os consumidores esperam que o serviço seja sob demanda e de modo praticamente instantâneo. Para isso o provedor de serviços deve disponibilizar uma plataforma em que o cliente consiga facilmente solicitar, personalizar, usar e pagar serviços sem intervenção humana;
- **Uso medido e faturado:** os serviços de computação em nuvem devem ser tarifados a curto prazo (por exemplo, por hora), permitindo ao usuário liberar os recursos assim que necessário. A medição é feita de acordo com o serviço (armazenamento, processamento e largura de banda) e deve frequentemente ser relatada ao usuário, proporcionando assim transparência;
- **Elástico:** o usuário tem a ideia de que possui recursos ilimitados. Os usuários esperam que a nuvem ofereça rapidamente recursos adicionais e que os mesmos possam ser

provisionados automaticamente, quando a carga da aplicação aumentar, e liberados quando a carga diminuir (escalar para cima e para baixo); e

- **Personalizável:** muitas vezes a necessidade do usuário é bastante variável, por essa razão, é fundamental que os recursos oferecidos pela nuvem sejam altamente personalizáveis.

2.2 Modelos de Serviços de Computação em nuvem

A computação em nuvem pode oferecer diferentes tipos de serviços como *hardware*, *software*, dados e infraestrutura. De acordo com Mell, Grance et al. (2011) esses serviços podem ser classificados e agrupados nas seguintes categorias:

- **SaaS (*Software* como Serviço):** neste modelo o consumidor utiliza aplicativos que são executados em uma infraestrutura em nuvem. As aplicações são acessíveis a partir de dispositivos que utilizam uma interface como um navegador *web* ou interface de programa, por exemplo, para autenticar o acesso do cliente. Toda parte de infraestrutura é gerenciada pelo provedor de serviços, isso inclui armazenamento, sistema operacional, rede, e até mesmo recursos de aplicativos contratados, com possíveis exceções de configuração especificas a seu usuário;

- **PaaS (Plataforma como Serviço):** neste modelo, o consumidor tem acesso a um ambiente de desenvolvimento e implementação na nuvem. O consumidor gerencia os aplicativos e serviços desenvolvidos e o provedor de serviço gerencia todo o resto (armazenamento, rede, servidores, etc); e

- **IaaS (Infraestrutura como Serviço):** neste modelo, são oferecidos ao consumidor recursos fundamentais de computação (processamento, armazenamento, rede, entre outros), de maneira que o consumidor possa implementar e executar *software* de forma arbitrária. O consumidor tem controle sobre o sistema operacional, armazenamento e aplicativos implementados, além do controle de maneira limitada sobre recursos de rede como *firewalls*.

2.3 Arquitetura de computação em nuvem

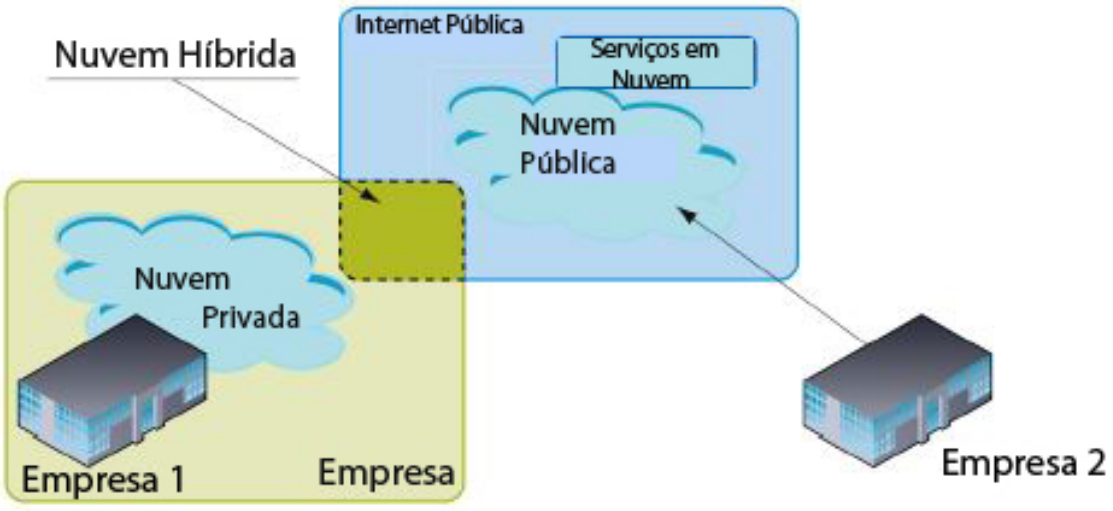
De acordo com Mell, Grance et al. (2011) um ambiente em nuvem pode ser implementado de quatro maneiras:

- **Nuvem Privada *(Private Cloud)*:** a infraestrutura de nuvem é provisionada para uso exclusivo de uma única organização que compreende vários consumidores (distintas unidades da mesma empresa, por exemplo). Pode ser de propriedade, gerenciado e operado pela organização, um terceiro grupo, ou alguma combinação de ambos;

- **Nuvem Comunitária *(Community Cloud)*:** a infraestrutura de nuvem é utilizada por um conjunto de organizações que possui interesses em comum. Ela pode ser de propriedade, gerenciada e operada por uma ou mais organizações da comunidade, por um terceiro, ou por alguma combinação de ambos;
- **Nuvem Pública *Public Cloud*:** neste modelo a infraestrutura de nuvem é provida e aberta para o público geral. Pode ser gerenciada por organização governamental, acadêmica ou comercial; e
- **Nuvem Híbrida *(Hybrid Cloud)*:** a infraestrutura de nuvem é composta por duas ou mais infraestruturas de nuvem distintas (privadas, comunitárias ou públicas) que permanecem como entidades exclusivas, mas que são unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações entre as nuvens.

A Figura 3 exemplifica as arquiteturas de computação em nuvem citadas anteriormente.

Figura 3 – Exemplo de implementação de nuvem pública, privada e híbrida (GUPTA; GUPTA, 2012).

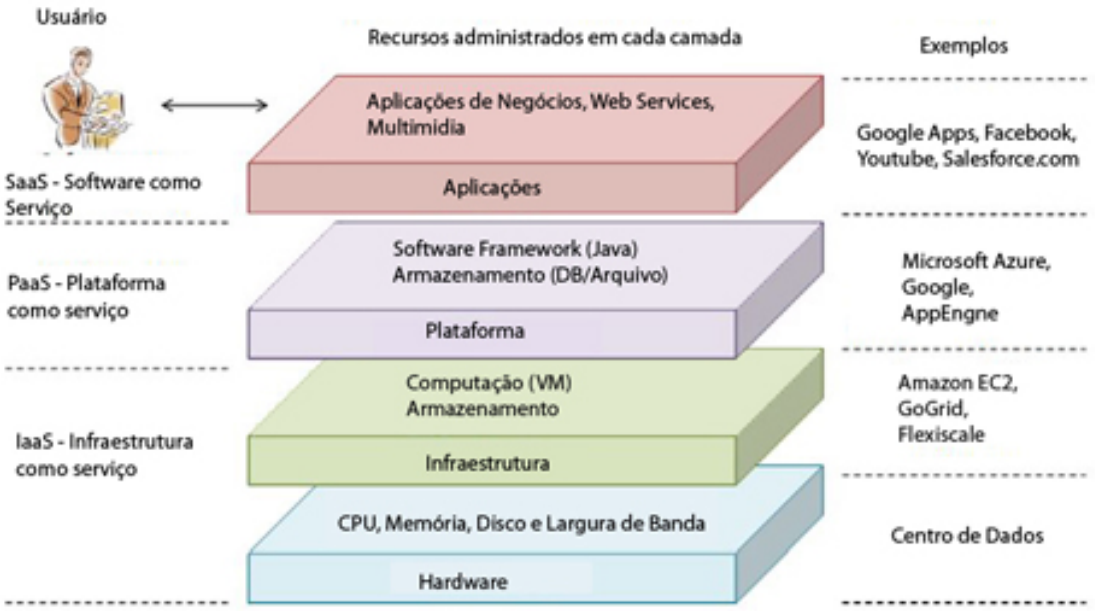


(Traduzido e adaptado pelo autor)

De modo geral, a arquitetura do ambiente de computação em nuvem pode ser dividida em quatro camadas: *hardware*, infraestrutura, plataforma e aplicação. Na Figura 4 pode-se observar o que cada camada gerencia e quais os modelos de serviços utilizam a mesma (ZHANG; CHENG; BOUTABA, 2010).

- **Camada de *Hardware*:** é responsável pela gestão de recursos físicos da nuvem, o que inclui servidores físicos, *switches*, sistemas de energia e refrigeração. Na prática, a camada de *hardware* é tipicamente implementada em *data centers*, que geralmente contém milhares de servidores que são organizados em *racks* e interconectados por meio de roteadores ou

Figura 4 – Arquitetura de computação em nuvem (ZHANG; CHENG; BOUTABA, 2010).



(Traduzido e adaptado pelo autor)

outros meios. Questões típicas da camada de *hardware* incluem, configuração de tolerância a falhas, gestão do tráfego de alimentação e gestão de recursos de resfriamento;

- **Camada de infraestrutura:** também conhecida como a camada de virtualização, a camada de infraestrutura cria um *pool* de recursos de armazenamento e computação, dividindo os recursos físicos por virtualização. A camada de infraestrutura é essencial na computação em nuvem, uma vez que muitas funcionalidades, como atribuição dinâmica de recursos, são disponibilizadas por meio de tecnologias de virtualização;
- **Camada de plataforma:** consiste nos sistemas operacionais e *frameworks* de aplicação disponibilizados pela nuvem; e
- **Camada de Aplicação:** consiste em aplicativos de nuvem reais. Diferente das aplicações tradicionais, as aplicações em nuvem podem alavancar o recurso de auto escala, garantindo melhor desempenho, disponibilidade e menor custo operacional.

2.4 Serviços de computação em nuvem

Serviços de computação em nuvem são soluções de T.I oferecidas pela infraestrutura da nuvem. Neste modelo, têm-se um serviço que será contratado pelo usuário.

2.4.1 *Amazon AWS*

A *Amazon Web Services* (AWS) é um conjunto de serviços de computação em nuvem que surgiu em 2006, possuindo uma oferta única em serviços de computação em nuvem com o

foco em IaaS. Os serviços AWS permitem o acesso a recursos de computação e outros serviços de infraestrutura sob demanda. A proposta dos serviços AWS é fornecer serviços do tipo IaaS com flexibilidade, efetividade, escalabilidade, elasticidade e segurança (VERAS, 2015).

Os elementos centrais dos serviços de infraestrutura da Amazon fornecem blocos de construção mais comuns para aplicações, que são:

- **Computação:** *Amazon Elastic Comput Cloud* (EC2) é um ambiente virtual que permite aumentar ou diminuir a capacidade de recursos de computação com base na demanda, além de facilitar o fornecimento de novas instâncias de servidor.
- **Armazenamento:** é possível armazenar qualquer coisa que o aplicativo necessite no *Amazon Simple Storage Service* (S3) e tirar vantagem do armazenamento escalável, confiável, de alta disponibilidade e de baixo custo.

A estrutura da AWS conta com quatro níveis de suporte para os usuários (AMAZON, 2017):

- **Básico:** gratuito durante 12 meses, mas sem tempo de resposta estabelecido;
- **Desenvolvedor:** pago, com tempo de resposta menor que 12 horas;
- **Comercial:** pago, com tempo de resposta menor que uma hora; e
- ***Enterprise:*** pago, com tempo de resposta menor que 15 minutos.

2.4.2 *Microsoft Windows Azure plataform*

O Windows Azure é um ambiente de nuvem pago que surgiu em 2009. Ele oferece as três categorias de serviços de computação em nuvem (IaaS, PaaS e SaaS) (CARUTASU et al., 2016). Os principais serviços trazidos pelo Windows Azure são classificados da seguinte forma (MICROSOFT AZURE, 2017):

- **Azure Máquinas Virtuais:** traz suporte para instalação de ambientes em Linux, Windows, *SQL Server*, Oracle, IBM e SAP em nuvem;
- **Azure Serviços em Nuvem:** é possível desenvolver, empacotar e implementar aplicativos e serviços na nuvem, garantindo sempre o balanceamento de carga e monitoramento de integridade.
- **Azure *Web Site*:** permite aos desenvolvedores compilar, implementar e gerenciar rapidamente sites e aplicativos *Web* avançados; e
- **Azure Serviços móveis:** permite desenvolvimento e compilação de aplicativos com base nativa em iOS, Android, Windows e Mac.

2.4.3 Google App Engine

É uma plataforma para criação de aplicativos *Web* dimensionáveis e *back-ends* móveis, usando linguagens conhecidas como Java, Python, PHP ou Go. Com o *App Engine* se tem acesso a serviços incorporados e APIs, como armazenamento de dados no NoSQL, API de autenticação do usuário, comum para a maioria dos aplicativos. A plataforma conta com escalonamento automático, gerando assim maior otimização na utilização de recursos e economia. Além disso, possui (GOOGLE, INC, 2017):

- Segurança para aplicação: utilização de certificados SSL/TSL;
- Controle de versão do aplicativo: versionamento da aplicação, além da criação de ambientes de desenvolvimento, teste, cenário e produção.
- Flexibilidade: com ambientes de execução personalizados, é possível integrar qualquer biblioteca ao App Engine.

2.4.4 IBM *BlueMix*

IBM Bluemix é uma implementação da arquitetura de nuvem aberta da IBM. Ela é baseada em Cloud Foundry, isto é, uma PaaS de código aberto, o que permite criar e implementar aplicativos rapidamente na nuvem. O Bluemix garante integração direta a ambientes em nuvem públicas, privadas e híbridas. Possui infraestrutura segura, escalável, flexível, e fornece soluções corporativas customizadas (IBM, 2017).

2.5 Gestores de computação em nuvem

Para facilitar o processo de implementação de um sistema de computação em nuvem existem os gestores de computação em nuvem. Estes são *softwares* responsáveis por fazer a integração das diversas camadas de *software* e *hardware* necessárias. Esses gestores podem ser específicos para um determinado modelo de serviço de computação em nuvem (IaaS, SaaS e PaaS) ou dar suporte para mais de um deles. Os gestores mais reconhecidos são:

2.5.1 *Eucalyptus*

Eucalyptus é uma infraestrutura paga e de *software* de código aberto *(open-source)* para implementar computação em nuvem. Começou com um projeto de pesquisa no Departamento de Ciências de Computação, na Universidade da Califórnia, Santa Barbara. Com o sistema *Eucalyptus* é possível implementar uma estrutura para criação de máquinas virtuais e controle sobre os recursos existentes. Também é possível construir sua própria nuvem privada por traz do *firewall* da sua companhia. A principal característica do Eucalyptus é sua API ser compatível com o *Amazon Web Service* (AWS) (EUCALYPTUS SYSTEMS, INC., 2017).

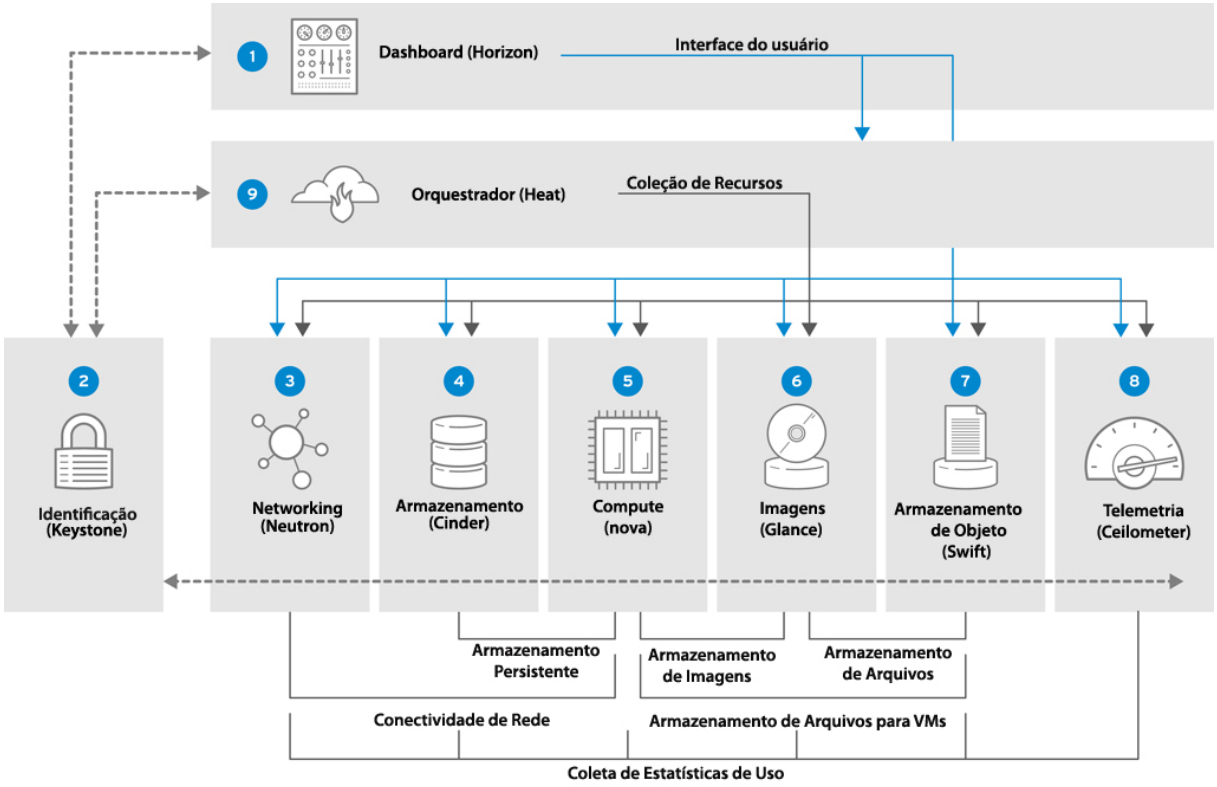
2.5.2 *OpenStack*

Seu projeto teve início com a Rackspace e a NASA, e seu lançamento ocorreu no ano de 2010. É um ambiente *open-source* para criação de nuvens públicas e privadas. Controla grande *pool* de recursos de computação, armazenamento e rede em um *data center*, gerenciados a partir de um painel de controle ou API *OpenStack*. É indicado para infraestruturas heterogêneas pelo fato de ser um ambiente *open-source*. Esta nuvem suporta diversos *hypervisors* como XenServer, KVM, VMware e *Hyper-V* (OPENSTACK, 2017).

OpenStack é um gestor bastante utilizado e constantemente explorado por grandes empresas como IBM e VMware (THONGTANUNAM et al., 2016).

Sua arquitetura é composta por vários subcomponentes, desenvolvidos de forma independente. Sendo assim, para se obter certas funcionalidades e características relacionadas a nuvem IaaS é necessária a instalação e configuração desses subcomponentes (SCHARF et al., 2015). A Figura 5 fornece uma visão de alto nível da arquitetura do Openstack.

Figura 5 – Arquitetura do OpenStack (REDHAT, 2017).



(Traduzido e adaptado pelo autor)

De acordo com RedHat (2017), será descrita a seguir a funcionalidade de cada subcomponente do OpenStack presente na Figura 5.

- **Horizon - *Dashboard* (Item 1 na Figura 5):** é responsável pela interface *web* para os serviços do Openstack. Com essa interface é possível efetuar o gerenciamento de diversos

recursos da nuvem;

- **Keystone - Idenficação (Item 2):** aqui é feita a autenticação dos serviços oferecidos pela nuvem;
- **Neutron - *Networking* (Item 3):** é o módulo responsável pelos serviços de rede, desde a criação até configuração de seus recursos;
- **Cinder - Armazenamento de bloco (Item 4):** fornece armazenamento em nível de blocos, que pode ser montado como um volume pelas instâncias;
- **Nova - Compute (Item 5):** este módulo é responsável por disponibilizar máquinas virtuais de acordo com a demanda do cliente;
- **Glance - Imagens (Item 6):** faz o gerenciamento das imagens de VM utilizadas pelo Nova;
- **Swift - Armazenamento de Objetos (Item 7):** este módulo fornece o serviço de armazenamento de objetos em que os clientes e administradores podem armazenar ou efetuar a busca de seus arquivos;
- **Ceilometer - Telemetria (Item 8):** Fornece medidas em relação aos recursos que são utilizados na nuvem;
- **Heat - Orquestrador (Item 9):** Responsável por fornecer modelos para criar e gerenciar automaticamente a pilha de recursos da nuvem.

Dentre as vantagens oferecidas pelo Openstack, pode-se destacar as seguintes (REDHAT, 2017):

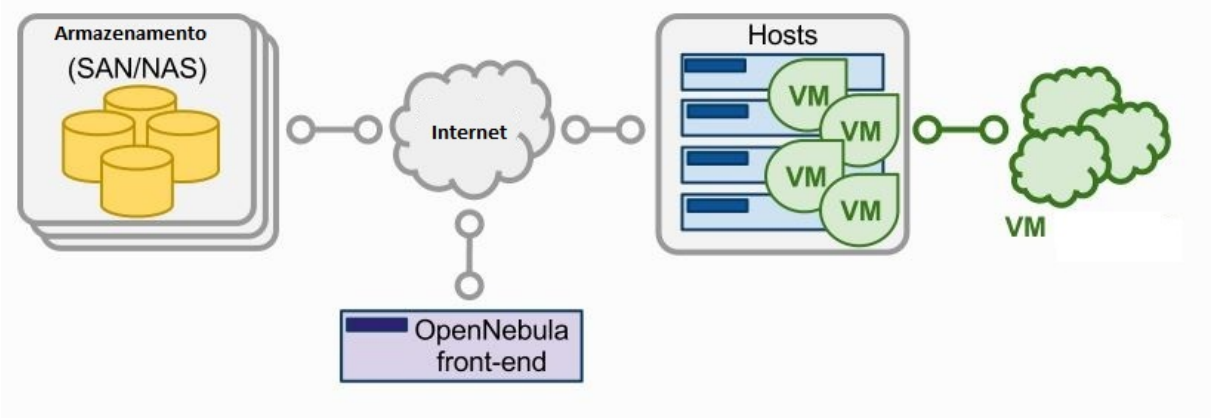
- Um único modelo fornece acesso a todas as APIs de serviço subjacentes;
- Os modelos são modulares e orientados a recursos;
- Os modelos podem ser recursivamente definidos e reutilizáveis, como pilhas aninhadas, portanto, sua infra-estrutura pode ser definida e reutilizada de maneira modular;
- A implementação de recursos é plugável, permitindo assim o uso de recursos personalizados;
- Os recursos são auto-dimensionados, sendo assim, podem ser adicionados ou removidos do *cluster* de acordo com o uso; e
- A funcionalidade básica de alta disponibilidade está disponível.

2.5.3 OpenNebula

Lançada em 2008, oferece solução simples, mas rica em recursos e de alta flexibilidade, para o gerenciamento de *data centers* virtualizados para permitir nuvens IaaS privadas, públicas e hibridas (OPENNEBULA, 2017).

Seu modelo de infraestrutura básico conta com um *cluster front-end*, um conjunto de *hosts* onde as VMs serão executadas e um *datastore* que é responável pelo armazenamento de dados, sendo tudo unido por pelo menos uma rede física (OPENNEBULA, 2017). A Figura 6 exemplica o modelo básica de arquitetura do OpenNebula.

Figura 6 – Arquitetura básica do OpenNebula (OPENNEBULA, 2017).



(Traduzido e adaptado pelo autor)

2.5.4 Apache CloudStack

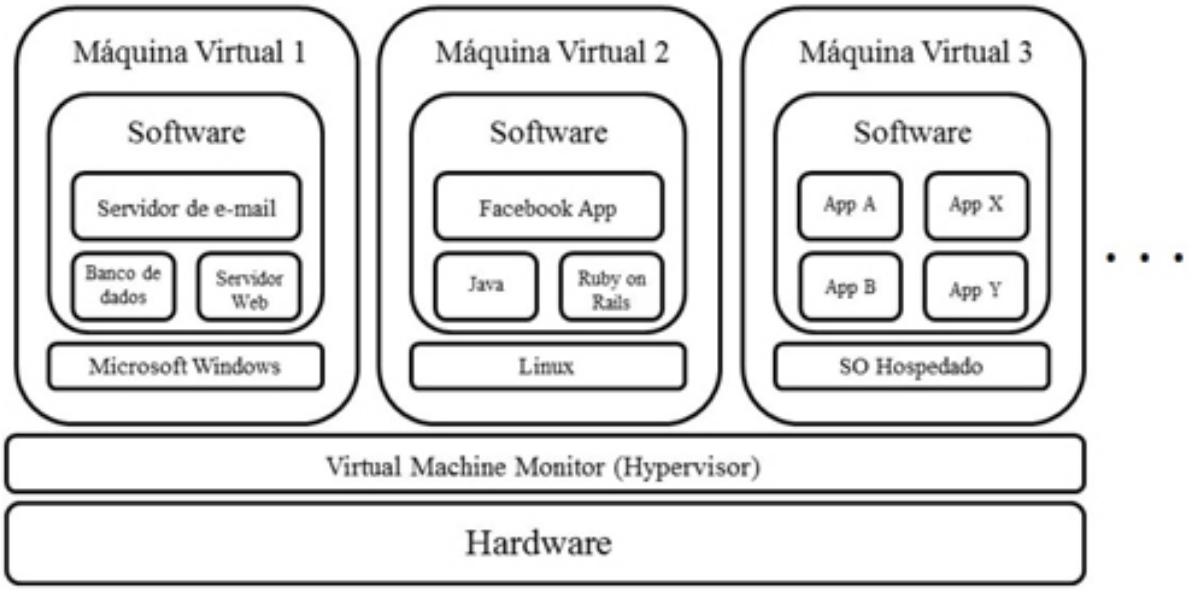
Esta nuvem *open-source* surgiu em 2010, projetada para implementar e gerenciar grandes redes de máquinas virtuais, como uma plataforma de computação em nuvem de infraestrutura de Serviço (IaaS), altamente disponível e escalável. É usada por vários provedores que oferecem serviços em nuvem pública e por diversas empresas que oferecem ambiente de nuvem privada ou como parte de uma solução de nuvem híbrida. Atualmente possui suporte aos principais *hypervisors*: *Vmware*, KVM, *Citrix XenServer*, *Xen Cloud Plataform* (XCP), servidor Oracle VM e *Microsoft Hyper-V* (CLOUDSTACK, 2017).

Este gestor é conceitualmente construído em pilhas de serviços e agentes internos. O primeiro serviço é o controlador de nuvem. O segundo é o servidor de armazenamento para hospedar volumes. O terceiro serviço (armazenamento secundário) é para manusear imagens ISO, modelos e instantâneos, sendo implementado utilizando um serviço de sistema de arquivos distribuídos. O último é um serviço de rede que implementa vários recursos avançados como LAN virtual, Rede Privada Virtual e encapsulamento de roteamento genérico (VOGEL et al., 2016).

2.6 Hypervisors

Os VMM (*Virtual Machine Monitor*), mais conhecidos como *hypervisors*, formam uma camada de *software* localizada entre o *hardware* "real" e as máquinas virtuais, sendo responsável por fornecer recursos (armazenamento, CPU, memória, etc.) da máquina física para a virtual. Essa estrutura pode ser vista na Figura 7, que ilustra um caso de virtualização em que três máquinas virtuais, com sistemas operacionais distintos, são hospedadas no mesmo *host*. Existem diversos *hypervisors* disponíveis para os usuários, sendo que aqui serão apresentados os *hypervisors* mais conhecidos e utilizados atualmente.

Figura 7 – Hospedagem de três máquinas virtuais com diferentes sistemas operacionais (BUYYA; BROBERG; GOSCINSKI, 2011).



(Traduzido e adaptado pelo autor)

2.6.1 *VMware*

VMware é a maior plataforma de virtualização do mundo. Está disponível em versões gratuitas e pagas. Sua principal ferramenta, a *vSphere*, traz inúmeras vantagens em ambientes virtualizados, tais como (VMWARE, 2017):

- Consolidação de servidores;
- Virtualização de qualquer aplicativo;
- Dimensionamento para atender às mais diversas demandas (nuvem, *big data*, redes sociais, entre outras);
- Garante alto desempenho, independente do aplicativo;
- Alocação dinâmica de recursos;

- Desempenho superior de aplicativos desktop com o *VMware Horizon* com NVIDIA GRID vGPU;
- Transferência de máquina virtual inteira em execução de um servidor físico para outro, sem tempo de inatividade; e
- Eficiente sistema de tolerância a falhas, permitindo recuperação rápida do ambiente *(vShare Fault Tolerance)*.

2.6.2 *XenServer*

É uma plataforma de virtualização *open-source* voltada para o ramo empresarial, com suporte para nuvem. Oferece todos os recursos necessários para qualquer implementação de virtualização de servidor e *datacenter* em ambiente Intel ou AMD. Também conta com inúmeros recursos de gestão, como por exemplo (XENSERVER, 2017):

- Migração ao vivo de VM: permite migrar VMs sem interrupções no sistema;
- Proteção contra falhas no *host*: reinicia automaticamente as VMs caso haja falhas em nível de VM, *hypervisor* ou servidor; e
- Conjunto de recursos heterogêneos: permite que o conjunto de servidores seja heterogêneos.

2.6.3 *KVM*

O KVM (Kernel-based Virtual Machine) é uma solução completa de virtualização para Linux em *hardware* x86. Consiste em um módulo de *kernel* que fornece a infraestrutura de virtualização central e um módulo específico do processador. Com este virtualizador é possível utilizar máquinas virtuais Linux ou Windows, com *hardware* virtualizado privado (placa de rede, placa gráfica, hd, etc). É um *software open-source* e está incluso nas versões oficiais do *kernel* do Linux desde a versão 2.6.20 (KVM, 2017).

2.6.4 *Microsoft Hyper-V*

Permite a criação de um ambiente virtualizado, utilizando a tecnologia de virtualização do *Windows Server*. O *Hyper-V* (MICROSOFT MSDN, 2017) fornece a infraestrutura necessária para virtualizar aplicativos e cargas de trabalho para dar suporte a uma variedade de metas comercias voltadas para a melhoria da eficiência e redução de custos, como:

- Estabelecer ou expandir um ambiente de nuvem privada;
- Aumentar a utilização de *hardware*;

- Melhorar a continuidade dos negócios;
- Estabelecer ou expandir uma infraestrutura VDI; e
- Aumentar a eficiência em atividades de desenvolvimento e testes.

2.7 Sistematização sobre computação em nuvem

Existe uma quantidade considerável de serviços e gestores de computação em nuvem, assim como de *hypervisors*. A Tabela 2 sistematiza as informações básicas em relação a categoria de serviço, se são soluções de código aberto ou não, e ainda se o sistema é gratuito, de cada *hypervisor*, gestor e *software* de computação em nuvem. Essas informações, embora não diretamente relacionadas ao foco deste trabalho, são de total importância para escolha da plataforma e *hypervisor* a ser utilizado. É importante destacar os sistemas considerados nesta tabela como gratuitos, não possuem necessariamente apenas versões gratuitas.

Tabela 2 – Dados referentes aos *hypervisors*, gestores e plataformas de computação em nuvem.

Sistema	Categoria	Código Aberto	Gratuito	Classificação
Amazon AWS	IaaS	Não	Não	*Software*
Microsoft Windows Azure	IaaS, PaaS, SaaS	Não	Não	*Software*
Google App Engine	PaaS	Não	Não	*Software*
IBM BlueMix	PaaS	Não	Não	*Software*
OpenStack	IaaS	Sim	Sim	Gestor
Apache CloudStack	IaaS	Sim	Sim	Gestor
Eucalyptus	IaaS	Sim	Não	Gestor
OpenNebula	IaaS	Sim	Sim	Gestor
KVM	-	Sim	Sim	*Hypervisor*
XenServer	-	Sim	Sim	*Hypervisor*
VmWare	-	Não	Não	*Hypervisor*
Microsoft Hyper-V	-	Não	Não	*Hypervisor*

Fonte: Elaborado pelo autor.

2.8 Tolerância a falhas

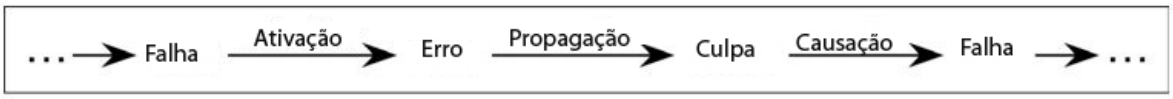
Em ambientes distribuídos um aspecto importante é o da confiabilidade, ou seja, garantia de que o sistema estará disponível sempre que necessário, com a não ocorrência de interrupções. Como essas interrupções surgem por falhas no sistema, é vital que se entenda como elas ocorrem e como podem ser tratadas.

2.8.1 Conceitos básicos

Segundo Tanenbaum e Steen (2007), a existência de um defeito é ocasionada pela ocorrência de algum erro no sistema. A causa desse erro é uma falha do sistema, sendo que a identificação da falha e sua correção são aspectos importantes em qualquer sistema computacional, em especial em sistemas distribuídos como são as nuvens.

Várias falhas podem ocorrer antes de gerar um erro. Uma falha que gera um erro é considerada uma falha ativa, caso contrário, é considerada uma falha latente. Um erro é sucessivamente transformado em outros erros, propagando-se através de componentes internos e externos, gerando por fim falhas em serviços. A Figura 8 ilustra essa cadeia de ameaças provenientes de uma falha (AVIZIENIS et al., 2004).

Figura 8 – Cadeia de ameaças provenientes de uma falha (AVIZIENIS et al., 2004).



(Traduzido e adaptado pelo autor)

Computação tolerante a falhas compreende ambientes em que com o surgimento de erros, o sistema funcione e seja capaz de se estabilizar com base em suas ferramentas e componentes voltados para esse tratamento, garantindo sempre a disponibilidade, a confiabilidade e a segurança (SHOOMAN, 2001).

Uma característica de sistemas distribuídos que os distingue de sistemas de uma única maquina é a noção parcial de falha. Em um ambiente distribuído uma falha pode afetar o funcionamento de alguns componentes e, ao mesmo tempo, deixar outros totalmente ilesos. Já em sistemas convencionais, uma falha em sistemas quase sempre é total, no sentido de que afetará todos os componentes e pode facilmente fazer o sistema parar por inteiro (TANENBAUM; STEEN, 2007). Tanenbaum e Steen (2007) definem alguns requisitos úteis relacionados a sistemas distribuídos tolerantes a falhas:

- **Disponibilidade:** é definida como a propriedade de um sistema estar funcionando corretamente e pronto para ser usado imediatamente;
- **Confiabilidade:** é a propriedade de um sistema poder funcionar continuamente sem falhas, isto é, sem interrupções durante um período de tempo relativamente longo;
- **Segurança:** refere-se à situação em que, se um sistema deixar de funcionar corretamente durante um determinado tempo, não haverá maiores danos para os seus usuários; e
- **Capacidade de manutenção:** refere-se à facilidade de realizar manutenção em um sistema caso ele falhe.

Para alcançar tais requisitos, existem algumas métricas tradicionalmente usadas quando trata-se de sistemas de tolerância a falhas. De acordo com Koren e Krishna (2007) elas estão diretamente relacionadas com confiabilidade, são o *Mean Time to Failure* (MTTF), e o *Mean Time Between Failures* (MTBF). O primeiro é o tempo médio que o sistema opera até ocorrer uma falha, enquanto o segundo é o tempo médio entre duas falhas consecutivas. A diferença entre os dois é devido ao tempo necessário para reparar o sistema após a primeira falha, denominado *Mean Time to Repair* (MTTR). Com sua introdução chega-se na Equação 2.1:

$$MTBF = MTTF + MTTR \tag{2.1}$$

2.8.2 Modelos de falhas

Quando um sistema apresenta algum tipo de falha, significa que o sistema não está disponibilizando os serviços de acordo com o projetado. Porém, nem sempre o problema está diretamente relacionado aos servidores que estão fornecendo o serviço, pois pode ser que tal serviço dependa de outros servidores para funcionar adequadamente. Tanenbaum e Steen (2007) descrevem diversos tipos de falhas, tais como:

- **Falha por queda:** ocorre quando um servidor para de funcionar prematuramente, deixando de produzir saída, de tal modo que sua parada pode ser detectada por outros processos. Um exemplo desse modelo de falha é quando um sistema operacional para de repente e a única solução é reinicia-lo;
- **Falha por omissão:** acontece quando um servidor deixa de responder a uma requisição. Neste caso o servidor não fica ciente de que alguma mensagem foi enviada a ele. Tais falhas podem ser provenientes de erros de *software* tais como laços infinitos ou gerenciamento inadequado. Pode ocorrer também quando o *buffer* de envio transborda e o servidor não estiver preparado para tal situação;
- **Falha por temporização:** acontece quando a resposta se encontra fora de um intervalo de tempo real especificado, isto é, o servidor responde tarde demais. Essa falha é, muitas vezes originada, por falta de desempenho;
- **Falha de resposta:** ocorre quando a resposta do servidor a uma requisição está simplesmente incorreta. Por exemplo, um mecanismo de busca que retorne sistematicamente páginas *Web* não relacionados com qualquer uma das palavras utilizadas para a busca; e
- **Falhas arbitrárias:** também conhecidas como falhas bizantinas. Quando ocorrem, o cliente deve se preparar para forte possibilidade de interrupção do serviço, uma vez que pode ocorrer de um servidor estar gerando saídas que nunca deveriam ser geradas, mas que não podem ser detectadas como incorretas.

2.8.3 Técnicas de tolerância a falhas

Um ambiente tolerante a falhas tem maior garantia de disponibilidade, segurança e confiabilidade, garantindo que o ambiente se recupere e mantenha sua funcionalidade de maneira íntegra mesmo após uma falha. A tolerância normalmente é obtida com a aplicação de técnicas de recuperação, que podem ser classificadas de várias formas. A mais tradicional é a classificação em quatro fases de aplicação: detecção, confinamento, recuperação e tratamento, apresentadas na Tabela 3 (ANDERSON; LEE, 1981).

Tabela 3 – Fases para aplicação das técnicas de tolerância a falhas.

Fases	Mecanismos
Detecção de erros	Replicação Testes de limites de tempo Cão de guarda (watchdog timers) Testes reversos Codificação: paridade, códigos de detecção de erros Teste de razoabilidade, de limites e de compatibilidades Testes estruturais e de consistência Diagnóstico
Avaliação de danos	Ações atômicas Operações primitivas auto encapsuladas Isolamento de processos Regras do tipo tudo que não é proibido Hierarquia de processos Controle de recursos
Recuperação do erro	Técnicas de recuperação por retorno Técnicas de recuperação por avanço
Tratamento da falha e continuidade do serviço	Diagnóstico Reparo

Fonte: (ANDERSON; LEE, 1981) (traduzido e adaptado pelo autor).

2.8.4 Tolerância a falhas na computação em nuvem

As técnicas de tolerância a falhas permitem que um sistema continue executando quando uma de suas partes cair. Se o sistema não estiver totalmente operacional, as soluções de tolerância a falhas permitem que ele continue operando com capacidade reduzida em vez de ficar inoperante após uma falha (AMIN; SINGH; SETHI, 2015).

A avaliação de tolerância a falhas em ambientes de computação em nuvem deve ser feita seguindo alguns parâmetros (PATRA; SINGH; SINGH, 2013):

- ***Throughput*:** calcula o número de tarefas de uma execução que foram concluídas. Deve atender integralmente a demanda para garantir o desempenho.

- **Tempo de resposta:** é o tempo necessário para um algoritmo específico responder. Esse parâmetro deve ser minimizado;
- **Escalabilidade:** é a capacidade da técnica de tolerância a falhas funcionar independentemente do número de nós. Este parâmetro deve ser, sempre que possível, melhorado;
- **Performance:** verifica a eficácia do sistema. O desempenho do sistema tem de ser aumentado a um custo razoável, reduzindo o tempo de resposta, e mantendo os atrasos aceitáveis;
- **Disponibilidade:** é a possibilidade de um item estar funcionando em uma determinada instância de tempo sob circunstâncias definidas;
- **Usabilidade:** o ambiente, ao ser utilizado por um usuário, deve atingir os objetivos como eficácia, eficiência e satisfação; e
- **Sobrecarga associada:** é a sobrecarga resultante da implementação de um algoritmo. Sobrecargas podem ser impostas por causa de movimentos de tarefas entre processos ou uma comunicação entre processadores. Para a eficiência da técnica de tolerância a falhas, o custo geral deve ser minimizado.

A obtenção de tolerância a falhas envolve técnicas para atuação tanto em nível de tarefa como em nível de fluxo de trabalho para resolver as falhas. Essas técnicas, em se tratando de computação em nuvem, podem ser reativas ou proativas (BALA; CHANA, 2012). De modo geral, propostas de solução podem seguir uma técnica específica ou compor duas ou mais técnicas. Como essas técnicas podem variar de acordo com a plataforma de computação em nuvem utilizada, o que será apresentado a seguir é uma visão geral de cada técnica, não levando em consideração nenhuma plataforma de computação em nuvem.

2.8.5 Técnicas reativas de tolerância a falhas

Técnicas reativas de tolerância a falhas são usadas para reduzir o impacto proveniente de falhas em um sistema quando as falhas realmente ocorrerem, sendo aplicadas portanto depois de sua ocorrência. Técnicas reativas incluem:

- ***Checkpointing*:** envolve a gravação de pontos de verificação da execução, sendo que a tarefa com falha é reiniciada a partir do ponto de verificação mais recente, sendo eficiente para aplicações de longa duração;
- **Replicação:** executa várias réplicas da tarefa em recursos distintos até que ela seja executada em sua totalidade, não sendo interrompida;
- **Migração de trabalho:** o trabalho de uma tarefa, quando por algum motivo ela não pode ser completamente executada em uma máquina, é migrado para uma nova máquina;

- **SGuard:** é baseado em recuperação de *rollback*, isto é, desfazer e/ou descartar todas as alterações feitas durante determinada transação;
- **Repetir:** esta é a técnica mais simples em nível de tarefa. O usuário reenvia a tarefa para o mesmo recurso da nuvem;
- **Ressubmissão de tarefa:** em tempo de execução, a tarefa com falha é submetida novamente à mesma máquina em que estava operando ou qualquer outra máquina;
- **Manipulação de exceções definidas pelo usuário:** aqui o usuário define ações específicas a serem tomadas na ocorrência de uma falha de tarefa para o fluxo de trabalho; e
- **Resgate do fluxo de trabalho:** permite que o sistema continue funcionando após a falha de qualquer tarefa até que se torne impossível avançar sem atender a tarefa falhada.

2.8.6 Técnicas proativas de tolerância a falhas

A tolerância a falhas proativa prediz as falhas proativamente e substitui os componentes suspeitos por outros componentes de trabalho. As técnicas podem ser (BALA; CHANA, 2012):

- **Rejuvenescimento de *software*:** o sistema planeja reinicializações periódicas, começando em um novo estado toda vez que reinicializa;
- **Autocura:** a tarefa é dividida em partes. Quando várias instâncias de um aplicativo estão sendo executadas em várias máquinas virtuais, ela trata automaticamente a falha proveniente de um aplicativo em execução; e
- **Migração preemptiva:** nesta técnica a aplicação é constantemente observada e analisada. A migração antecipatória de uma tarefa depende do mecanismo de controle de retorno.

A Tabela 4 apresenta apresenta alguns exemplos da aplicação de técnicas de tolerância a falhas descritas anteriormente em sistemas específicos. Nela é possível observar que mesmo situações semelhantes levam a soluções diversas, comprovando a dificuldade do processo de escolha por uma solução.

Tabela 4 – Resumo das técnicas de tolerância a falhas para sistemas específicos.

Sistema	Técnica de tolerância a falha	Políticas	Estrutura de programação	Ambiente	Falha Detectada	Tipo de aplicação
HAProxy	Auto cura, migração de trabalho, replicação	Reativa/ Proativa	Java	Máquina Virtual	Processo/ falha no nó	Balanceamento de carga e tolerância a falhas
SHelp	Checkpoint	Reativa	SQL, Java	Máquina Virtual	Falha na aplicação	Tolerância a falhas
Assure	Checkpoint, repetir, auto cura	Reativa/ Proativa	Java	Nuvem	Host, falha na rede	Tolerância a falhas
Hadoop	Migração de trabalho, Sguard	Reativa/ Proativa	Java, HTML, CSS	Nuvem	Aplicação/ falha no nó	Dados intensivos
Amazon EC2	Replicação, Sguard, resubmissão de tarefa	Reativa/ Proativa	Amazon Machine Image	Nuvem	Aplicação/ falha no nó	Balanceamento de carga e tolerância a falhas

Fonte: (BALA; CHANA, 2012) (Traduzida e adaptada pelo autor).

2.8.7 Módulos de tolerância a falhas

Com base nas técnicas citadas nas seções 2.8.5 e 2.8.6 é possível implementar os seguintes módulos de tolerância a falhas (CHERAGHLOU; KHADEM-ZADEH; HAGHPARAST, 2016):

- **AFTRC (*Adaptive Fault Tolerance model in Real Time Cloud Computing*):** neste sistema de ambiente proativo, a decisão é tomada com base na confiabilidade dos nós de processamento;
- **LLFT (*Low Latency Fault Tolerance*):** este modelo possui um *middleware* de tolerância a falhas de baixa latência, fornecendo um ambiente tolerante a falhas para aplicativos distribuídos implementados em nuvem. Este *middleware* replica o aplicativo por replicação semi-ativa ou processo de replicação semi-passiva, protegendo o aplicativo contra diversos tipos de falhas;
- **FTM (*Fault Tolerance Manager*):** nesta metodologia o usuário pode especificar e aplicar o nível desejado de tolerância a falhas. Este ambiente pode ser visto como um conjunto de vários componentes de serviços *web*, em que cada um possui uma funcionalidade específica;
- **FTWS (*Fault Tolerant Workflow Scheduling*):** é um algoritmo de escalonamento de fluxo de trabalho a falhas que fornece um ambiente tolerante a falhas por mecanicismos de replicação e reenvio de tarefa com base em sua prioridade;

- ***Candy*:** sua disponibilidade é baseada em componentes. Garante alta disponibilidade do serviço em nuvem; e
- **FTCloud:** é uma estrutura baseada em classificação de componentes para estrutura de aplicações em nuvem. Sua proposta é com a identificação de componentes significativos em uma aplicação em nuvem, efetuar a governança automática para a tolerância a falhas.

2.8.8 Injeção de falhas no ambiente

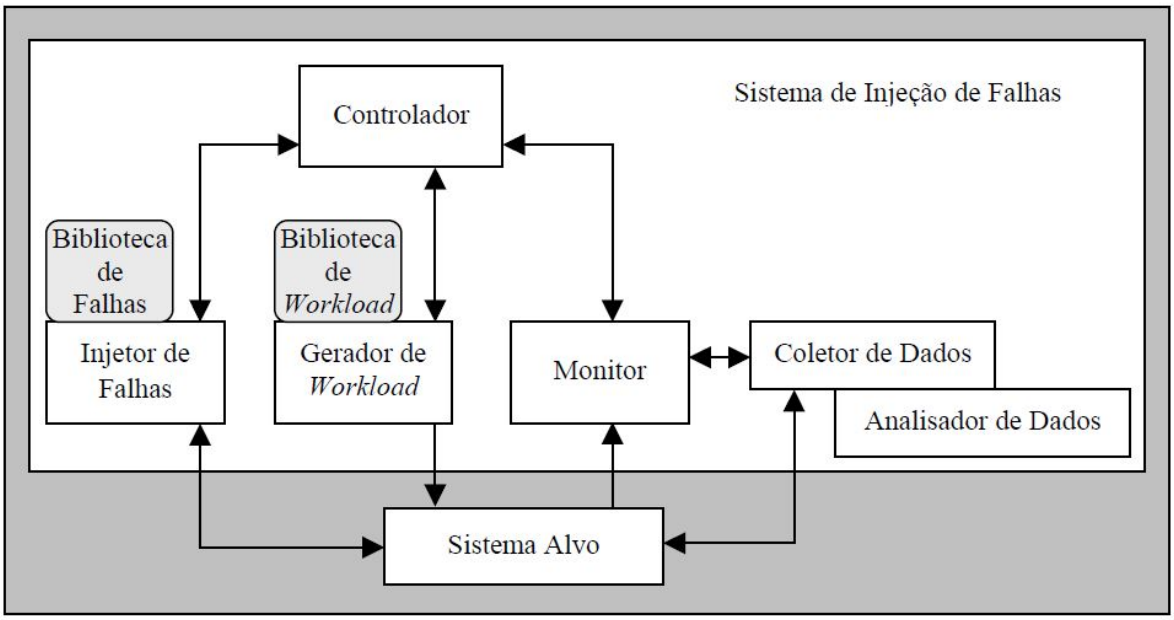
Técnicas para tolerância a falhas necessitam ser testadas na presença de falhas para serem validadas. Como não é prático esperar a ocorrência de uma falha para fazer esses testes, então uma técnica é provocar falhas no ambiente de computação em nuvem. Uma forma de se fazer isso é utilizando o mecanismo de *Software Fault Injection* (SFI). De acordo com Natella, Cotroneo e Madeira (2016) SIF é um método utilizado com o intuito de antecipar possíveis cenários que são ocasionados por erros, por meio da injeção de falhas a nível de *software* e/ou *hardware*.

A injeção pode ser feita de duas formas (HSUEH; TSAI; IYER, 1997):

- ***Software*:** este mecanismo é simples e barato, além de permitir simular algumas falhas em *hardware* também, porém, dependendo do caso, com menos precisão quando comparado a utilização de *hardware* para injeção da falha; e
- ***Hardware*:** neste modelo, as falhas são implementadas utilizando um *hardware* adicional como apoio, sendo consequentemente um método mais caro. Esse método é utilizado para estudar características de dependabilidade que necessitem de precisão em relação ao tempo, por exemplo, latência de falhas em CPU.

Segundo Hsueh, Tsai e Iyer (1997) um SFI é composto por: alvo (sistema que será testado), injetor de falhas, biblioteca de falhas, gerador de *workload*, biblioteca de *workload*, controlador, monitor, coletor de dados e analisador de dados. Esses componentes, bem como suas interações, podem ser observados na Figura 9.

Figura 9 – Sistema de Injeção de Falhas (HSUEH; TSAI; IYER, 1997).



(Traduzido e adaptado pelo autor)

Assim, em um SFI o injetor de falhas é o responsável por injetar falhas no sistema, enquanto este executa tarefas do gerador de *workloads* (aplicações e *benchmarks*). O monitor observa toda a execução do sistema alvo e inicia o coletor de dados sempre que for necessário armazenar alguma informação. A análise de dados pode ser feita *on-line* ou *off-line* usando o analisador de dados. Todos estes elementos são gerenciados pelo controlador, que pode ser executado em qualquer computador do ambiente (HSUEH; TSAI; IYER, 1997).

Uma alternativa ao uso de um SFI é inserir falhas de maneira manual. Isso envolve ações diversas como, por exemplo, desabilitar a placa de rede de um dos nós para provocar uma falha de comunicação, entre outras.

2.9 Trabalhos relacionados

O trabalho de revisão sistemática apresentado no próximo capítulo identificou uma grande quantidade de publicações envolvendo tolerância a falhas em ambientes de computação em nuvem. Dentre esses trabalhos foi possível identificar um pequeno conjunto cujo objetivo era de fazer uma revisão de trabalhos na área. Esses trabalhos de revisão serão apresentados aqui, ficando os trabalhos que apresentam técnicas para o tratamento de tolerância a falhas para o Capítulo 4.

Começando pelo artigo escrito por Prathiba e Sowvarnica (2017), os autores abordam de forma objetiva técnicas para tolerância a falhas, tais como rejuvenescimento de *software*, migração de trabalho, e *checkpoint*. Também é feita a classificação das falhas, e menção às

métricas para tolerar falhas. Por fim o autor apresenta um resultado com o comparativo de algumas técnicas utilizadas para tolerar falhas em *clusters*.

Já Ataallah, Nassar e Hemayed (2015) trazem uma abordagem um pouco mais detalhada em relação ao estudo anterior. O artigo possui alguns dos mesmos itens abordados no anterior, porém, aqui são abordados um número menor de técnicas para tolerar falhas, mas de forma mais completa. Por fim, o autor conclui que nuvem é um campo novo em que existem inúmeras maneiras para tolerar falhas e constantemente pesquisadores estão buscando um método autônomo de tolerância a falhas.

Pode-se dizer que o artigo escrito por Cheraghlou, Khadem-Zadeh e Haghparast (2016) apresenta as vantagens de cada um dos dois artigos descritos anteriormente. Nele, além de possuir uma abordagem detalhada de cada assunto, também apresenta uma quantia significativa de métodos para tolerar falhas.

Por último, o artigo escrito por Agarwal e Sharma (2015) é mais focado a definições de conceitos como "falha" e "erro", assim como a taxonomia de cada um. A abordagem em relação a técnicas de tolerância a falhas é feita de maneira objetiva, mas aborda uma quantidade considerável de técnicas. Os autores concluem frisando a importância da tolerância a falhas, e que o proposito do artigo é discutir as falhas que podem ocorrer em um ambiente de computação em nuvem.

Diferente dos trabalhos encontrados, neste trabalho se detalha cada solução proposta com o adicional de apontar vantagens e desvantagens entre eles. São incluídas também informações relacionadas a números de artigos encontrados abordando falhas em determinada situação, técnicas mais utilizadas, técnicas utilizadas de acordo com o tipo de falha, entre outros, criando assim um documento que possa ser utilizado para recomendação em tomadas de decisões referente a mecanismos de tolerância a falhas em computação em nuvem.

2.10 Considerações Finais

Em relação aos cenários de tolerância a falhas fica claro, com a Tabela 4, que é possível alcançar o mesmo objetivo para tolerar falhas usando sistemas distintos, e que é possível utilizar mais de uma técnica em um único sistema tolerante a falhas. Visto que existem vários tipos de falhas, assim como várias formas para seu tratamento, o SFI é um mecanismo importante para avaliar e identificar se um sistema é realmente tolerante a falhas. Os próximos capítulos se concentrarão na sistematização de soluções para tolerância a falhas encontradas em bases academicamente bem reconhecidas, como IEEE, ACM e ScienceDirect.

Apesar de existirem *surveys* sobre tolerância a falhas em ambientes de nuvem, nenhum dos que foram encontrados permite a geração de um sistema de recomendação de ações para gestores de computação em nuvem. Com base nos dados obtidos na revisão apresentada neste

capítulo é possível perceber que a criação desse ambiente de recomendação é interessante e necessário.

3 Revisão Sistemática

A Revisão Sistemática da Literatura (RSL) é uma técnica usada para identificar, avaliar e interpretar evidências na literatura científica relacionadas a um campo específico. Este capítulo apresenta essa metodologia (Seção 3.1) e os primeiros resultados quantitativos obtidos com a sua aplicação para o campo de tolerância a falhas em computação em nuvem (Seções 3.2, 3.3 e 3.4).

3.1 Metodologia

Com a RSL é possível identificar, avaliar e interpretar trabalhos relevantes a uma pesquisa em questão. Segundo Keele et al. (2007), este processo deve ser realizado, na medida do possível, de forma transparente e replicável seguindo-se as premissas:

- A revisão deve ser documentada em detalhes suficientes para avaliar seu rigor;
- A pesquisa deve ser documentada conforme ocorre e mudanças anotadas e justificadas; e
- Os resultados de pesquisa não filtrados devem ser salvos e retidos para possíveis reanálises.

Ainda de acordo com Keele et al. (2007) a RSL é uma ferramenta adequada para auxiliar o estudo de um novo campo. Isso é suportado considerando que sua prática permite primeiro resumir evidências existentes a respeito de uma tecnologia. Além disso, com tais dados é possível identificar lacunas ainda existentes naquele campo, permitindo, por fim, providenciar uma base para o posicionamento adequado de novas pesquisas.

A aplicação de RSL em um dado campo envolve as seguintes etapas:

1. Especificação dos objetivos e questões da pesquisa;
2. Desenvolvimento do protocolo de revisão;
3. Identificação da pesquisa;
4. Seleção de estudos primários;
5. Avaliação da qualidade dos estudos;
6. Extração de dados;
7. Síntese dos dados; e
8. Formação do relatório (dissertação).

Sendo assim, considerando que o objetivo era localizar trabalhos na área de tolerância a falhas em computação em nuvem, passou-se a desenvolver as etapas seguintes. Para as etapas de definição das questões de pesquisa, protocolo e identificação, as decisões tomadas foram as seguintes:

1. Definiu-se as bases que serão utilizadas para pesquisa, sendo escolhidas bases do IEEE, ACM e Science Direct.
2. Definiu-se o conjunto de palavras-chave que serão utilizadas para criação das *strings* de busca.
3. Definiu-se que o espaço de buscas nas bases escolhidas incluiria como metadados o *abstract*, título, e palavras-chave de cada artigo.
4. Para a base do IEEE serão considerados apenas artigos "publicados em conferências", e de "jornais ou revistas".
5. Para a base da ScienceDirect serão considerados artigos do tipo "artigos de revisão", "artigos originais", e "outros".
6. Para o trabalho de busca considerou-se o período de 2010 a 2017.

Para criação de *strings* de busca utilizou-se as seguintes palavras chaves: *fault*, *tolerance*, *defect*, *high*, *availability*, FT, OpenNebula, CloudStack, OpenStack, Cloud, Computing.

Todo o processo da RSL foi subdividido em três etapas. Na primeira etapa os resultados obtidos com cada *string* foram brevemente avaliados para definir quais resultados seriam utilizados nas próximas etapas, além de excluir artigos duplicados. Ao final da primeira etapa, foram gerados planilhas e gráficos com as informações extraídas, como por exemplo, número de artigos relevantes, números de artigos relevantes por ano, número de artigos por ano, entre outros.

Na segunda etapa os resultados foram filtrados com o intuito de extrair apenas artigos que realmente estão relacionados a tolerância a falhas em nuvem. Esse procedimento foi executado manualmente e repetido nas demais buscas aqui descritas fazendo-se a leitura dos tópicos:

- ***Abstract*:** verificação se o que é abordado no artigo tem realmente a ver com o tema buscado, no caso, tolerância a falhas em ambientes de computação em nuvem;
- **Tópicos específicos do trabalho:** identificação se o que está sendo sugerido é válido; e
- **Resultados ou conclusões:** verificação se a metodologia utilizada realmente trouxe resultados benéficos.

Os artigos são classificados conforme o tipo de falha tratada, sendo subdivididos da seguinte forma:

- **Falhas relacionadas a processos:** são os sistemas que tratam de falhas diretamente relacionadas a processos computacionais;
- **Falhas relacionadas a dados:** engloba sistemas que tratam falhas relacionados a arquivos computacionais;
- **Falhas relacionadas a comunicação:** engloba sistemas que tratam falhas relacionadas a comunicação, isto é, falhas relacionadas a rede;
- **Falhas relacionadas a virtualização:** engloba sistemas que tratam falhas com VMs;
- **Indefinido:** o artigo está relacionado com tolerância a falhas, porém, não foi possível classifica-lo apenas com a leitura dos itens utilizados para primeira filtragem. O mesmo entra para o grupo a ser estudado de forma mais profunda nas próximas etapas, para assim classifica-lo em alguma das falhas ou desclassifica-lo; e
- **Nenhum:** caso o artigo não esteja relacionado ao que se busca, retirando-o das próximas etapas.

Vale destacar que um artigo pode tratar de uma falha que engloba um ou mais itens dos quais foram citados acima. Com isso conclui-se a segunda etapa.

Por fim, a terceira etapa é voltada para a análise detalhada de cada artigo obtido após as etapas iniciais. O objetivo desta etapa é extrair informações referentes ao sistema utilizado para tolerância a falhas, identificando suas principais caraterísticas, modo de funcionamento e implementação, assim como suas vantagens e desvantagens. Nesta etapa cada sistema proposto foi classificado de acordo com as abordagens utilizadas (replicação, auto cura, entre outras). Ao final de cada avaliação, gerou-se uma tabela que traz de forma sintetizada algumas informações sobre o sistema apresentado em cada artigo. Essa tabela contém:

- **Técnicas utilizadas**;
- **Gestor de nuvem compatível**;
- **Modelo de programação**;
- **Problema solucionado**;
- **Tipo de falha tratada**;
- **Vantagem**;
- **Desvantagem**;

Deve-se destacar que mesmo existindo uma ordem pré-estabelecida em cada etapa, o processo todo possui um aspecto iterativo. Portanto, é possível que sejam feitas alterações em relação à classificação de um dado artigo, podendo inclusive desclassificá-lo.

Para as etapas citadas anteriormente, além do trabalho manual feito em planilhas do Microsoft Office Excel e anotações no Microsoft Office Word, utilizou-se a ferramenta StArt (Estado da Arte através da Revisão Sistemática) (START, 2017), para auxiliar os inúmeros estágios e atividades previstas na RSL. O uso dessa ferramenta é conveniente pois muitas vezes cada etapa tem de ser feita mais de uma vez, o que é facilitado com o suporte de uma ferramenta computacional.

3.1.1 Bases científicas

Os dados foram extraídos de três bibliotecas digitais: IEEE Xplore, ACM Digital Library e ScienceDirect.

IEEE Xplore (IEEE, 2017) é uma biblioteca digital mantida pelo Institute of Electrical and Electronics Engineers (IEEE). Ela apresenta quase 4 milhões de itens (entre artigos, livros, etc.) publicados pelo próprio IEEE ou seus parceiros. Grande parte dos itens fornecidos podem ser vistos tanto como documentos PDF como, também, páginas *web* comuns.

ACM Digital Library (ACM, 2017) é uma biblioteca digital mantida pela Association of Computing Machinery (ACM). Ela apresenta duas bases internas, a ACM Full-Text Collection, com mais de 400 mil publicações, e a ACM Guide to Computing Literature, com mais de 2 milhões de publicações entre livros, artigos, etc. Para a pesquisa utilizou-se somente a base ACM Full-Text Collection, devido a questões de acesso.

ScienceDirect (ELSEVIER, 2017) é uma biblioteca digital mantida pela Elsevier. Ela apresenta mais de 14 milhões de publicações pela própria Elsevier ou seus parceiros. Também permite que muitas das publicações possam ser vistas como PDFs ou páginas *web*.

Inicialmente foram consideradas também as bases da Springer, Wiley e Hindawi. Entretanto, as bases da Springer e Wiley não possuem filtros específicos de busca como os utilizados em outras bases. Já a base da Hindawi forneceu resultados muito limitados. Assim, essas bases não foram utilizadas. O mesmo ocorreu com as bases da Scopus e Web of Science, embora para estas o problema foi a falta de acesso para efetuar as buscas.

Os resultados obtidos através das três etapas da RSL, nas bases digitais indicadas, são descritos nas próximas seções.

3.2 Resultados obtidos na primeira etapa da RSL

Esta subseção contém os resultados obtidos com a aplicação da primeira etapa de revisão, que consiste basicamente em:

- Analisar resultados obtidos com *strings* de busca;
- remover artigos com pouca afinidade com o assunto desejado;
- classificar resultados por ano de publicação do artigo;
- verificar número de publicações em cada ano do período pré estabelecido; e
- identificar artigos duplicados.

No total foram criadas 24 *strings* de busca, que foram subdivididas em quatro grupos, cada grupo com um ID, sendo:

- SG*n*: não utilizam como palavra chave nenhum gestor de computação em nuvem na criação das *strings* de busca;
- SS*n*: utiliza a palavra chave *OpenStack* para criar as *strings* de busca;
- SCS*n*: utiliza a palavra chave *CloudStack* para criar as *strings* de busca;
- SN*n*: utiliza a palavra chave *OpenNebula* para criar as *strings* de busca;

3.2.1 Resultados para *Cloud Computing*

O processo de revisão envolveu buscas separadas por determinadas palavras chaves, como computação em nuvem e os vários gestores examinados. Para a busca de computação em nuvem a Tabela 5 mostra as *strings* de busca utilizadas, sendo atribuído uma identidade (ID) para cada *string* para facilitar sua referência no decorrer do texto.

Tabela 5 – Strings de busca utilizadas para Cloud Computing

ID	String de Busca
SG1	“fault tolerance” AND Cloud Computing
SG2	fault tolerance AND Cloud Computing
SG3	“fault tolerance” AND “Cloud Computing”
SG4	fault tolerance AND “Cloud Computing”
SG5	(“fault tolerance” OR FT) AND “Cloud Computing”
SG6	((fault OR defect OR FT OR tolerance) OR “High Availability”) AND "Cloud Computing"

Fonte: Elaborado pelo autor.

A Tabela 6 traz os resultados obtidos com a aplicação de cada *string* mostrada na Tabela 5, sendo que o quadro 6-A mostra resultados obtidos na base do IEEE, o 6-B com resultados da base da ACM, e o quadro 6-C para a base da Science Direct.

Tabela 6 – Número de artigos obtidos em cada base de acordo com cada string de busca SG*n*

IEEE	
ID	Resultados
SG1	971
SG2	1036
SG3	859
SG4	868
SG5	18
SG6	84

A

ACM	
ID	Resultados
SG1	693
SG2	695
SG3	450
SG4	480
SG5	450
SG6	0

B

ScienceDirect	
ID	Resultados
SG1	1.436
SG2	1703
SG3	1058
SG4	1143
SG5	53
SG6	1

C

Fonte: Elaborado pelo autor.

Com base nos dados mostrados na Tabela 6 é possível observar a imensa quantidade de informações existentes referente a tolerância a falhas em ambientes de computação em nuvem. As diferentes *strings* foram elaboradas com o intuito de filtrar o maior número de artigos relevantes para este trabalho, isto é, artigos que estão diretamente relacionados com o tema do *survey*. Porém, por mais variantes que fossem adicionadas na *string*, só SG5 e SG6 trouxeram resultados em uma quantidade tratável, com exceção da base da ACM. Para todas as bases as *strings* SG1, SG2, SG3, SG4, e SG5 trouxeram um número elevado de artigos e, SG6 nenhum artigo. Sendo assim será utilizada a *string* SG5, porém, inicialmente se desconsiderando os artigos referentes a base da ACM.

A Tabela 7 traz os artigos obtidos para a *string* SG5 (bases do IEEE e Science Direct) separados por ano de publicação.

Tabela 7 – Resultados por ano utilizando SG5 (Setembro de 2017)

IEEE	
Ano	Publicações
2010	2
2011	1
2012	2
2013	2
2014	1
2015	5
2016	0
2017	5

A

ScienceDirect	
Ano	Publicações
2010	1
2011	2
2012	3
2013	9(7)
2014	7(6)
2015	15(11)
2016	9
2017	18(14)

B

Fonte: Elaborado pelo autor.

Os resultados iniciais da ScienceDirect consideram não só artigos, mas também livros e enciclopédia . Porém, como o foco é artigos e por questões de acesso, esses elementos foram desconsiderados. Sendo assim, o número de artigo considerados o entre parêntesis.

3.2.2 Resultados para Apache Openstack

A Tabela 8 mostra as *strings* de busca utilizadas para a busca relativa ao Openstack. Assim como na Tabela 5, atribuiu-se uma ID para cada *string*.

Tabela 8 – Strings de busca utilizadas para Openstack

ID	String de Busca
SS1	(fault OR defect OR FT) AND tolerance AND Openstack
SS2	"fault tolerance" AND Openstack
SS3	fault AND tolerance AND OpenStack
SS4	fault AND (tolerance OR tolerant) AND Openstack
SS5	fault AND tolerance AND Reliability AND OpenStack
SS6	((fault OR defect OR FT OR tolerance) OR "High Availability") AND Openstack

Fonte: Elaborado pelo autor.

A Tabela 9 traz os resultados obtidos com cada *string* da Tabela 8, sendo que o quadro 9-A mostra resultados obtidos na base do IEEE, o quadro 9-B com a base da ACM, e o quadro 9-C da base da Science Direct.

Tabela 9 – Número de artigos obtidos em cada base de acordo com cada string de busca SS*n* (Setembro de 2017)

IEEE	
ID	Resultados
SS1	2.518
SS2	19
SS3	19
SS4	10
SS5	6
SS6	15

A

ACM	
ID	Resultados
SS1	8
SS2	8
SS3	8
SS4	8
SS5	4
SS6	0

B

ScienceDirect	
ID	Resultados
SS1	0
SS2	116
SS3	116
SS4	44
SS5	53
SS6	0

C

Fonte: Elaborado pelo autor.

A *string* SS1 trouxe uma quantidade elevada de artigos da base do IEEE, porém, sua maioria é de assuntos minimamente relacionados ao que realmente se quer, como virtualização, *big data*, entre outros. Já as *strings* SS2 e SS3 trouxeram resultados idênticos em cada base, visto que a única variante entre elas é o acréscimo do conectivo lógico "AND". Para SS4 e SS5 utilizou-se outras palavras chave, como "*tolerance*" em SS4, e "reliability" em SS5, com o intuito de aumentar o escopo, porém, em ambos os casos houve queda no número de artigos (com exceção da base da ACM) em relação às outras *strings*. SS6 trouxe resultado satisfatório apenas na base da IEEE. A partir dessa análise escolheu-se a *string* SS3 para aprofundamento da análise.

A Tabela 10 traz os resultados obtidos utilizando a *string* SS3, separados por ano e bases, sendo IEEE o quadro 10-A, ACM o quadro 10-B, e ScienceDirect o quadro 10-C.

Tabela 10 – Resultados por ano utilizando SS3 (Setembro de 2017)

IEEE	
Ano	Publicações
2010	0
2011	0
2012	0
2013	3
2014	0
2015	3
2016	8
2017	5
Total	19

A

ACM	
Ano	Publicações
2010	0
2011	0
2012	1
2013	0
2014	2
2015	1
2016	3
2017	1
Total	8

B

ScienceDirect	
Ano	Publicações
2010	0
2011	0
2012	1 (0)
2013	11 (8)
2014	12 (10)
2015	22 (19)
2016	23 (20)
2017	42 (35)
Total	111 (93)

C

Fonte: Elaborado pelo autor.

Os resultados iniciais da ScienceDirect consideram não só artigos, mas também livros (17) e enciclopédia (1). Porém, como o foco é artigos e por questões de acesso, esses elementos foram desconsiderados. Sendo assim, no quadro 10-C os números entre parênteses são as quantidades reais de artigos obtidos por ano, sendo estes artigos que serão considerados para análise. Portanto, considerando as três bases obteve-se no total 120 artigos.

3.2.3 Resultados para Cloudstack

O mesmo procedimento apresentado para o Openstack foi executado para o Cloudstack. No caso, a Tabela 11 traz os IDs das *strings* de busca utilizadas para obter artigos com mecanismos de tolerância a falhas voltados para o gestor Cloudstack.

Tabela 11 – Strings de busca utilizadas para Cloudstack

ID	String de Busca
SCS1	(fault OR defect OR FT) AND tolerance AND CloudStack
SCS2	fault tolerance AND CloudStack
SCS3	fault AND tolerance AND CloudStack
SCS4	fault AND (tolerance OR tolerant) AND CloudStack
SCS5	fault AND tolerance AND Reliability AND CloudStack
SCS6	((fault OR defect OR FT OR tolerance) OR “High Availability”) AND CloudStack

Fonte: Elaborado pelo autor.

Os resultados da aplicação dessas *strings* são apresentados na Tabela 12. Mais uma vez, algumas das *strings* (SCS1, SCS4, e SCS5) trouxeram um número de artigos exagerado para a

base do IEEE. Várias *strings* (SCS2 e SCS3 para base do IEEE, SCS1, SCS2, SCS3 e SCS4 para base da ACM, e SCS2 e SCS3 para base da ScienceDirect) trouxeram resultados exatamente iguais. Já a *string* SCS6 trouxe bons resultados de forma geral, porém, muitos deles relacionados a outros aspectos de computação em nuvem. Sendo assim, para este caso será utilizada SCS3 para análise mais aprofundada.

Tabela 12 – Número de artigos obtidos em cada base de acordo com cada string de busca SCS*n*

IEEE	
ID	Resultados
SCS1	844
SCS2	1
SCS3	1
SCS4	15.827
SCS5	21.158
SCS6	10

A

ACM	
ID	Resultados
SCS1	1
SCS2	1
SCS3	1
SCS4	1
SCS5	0
SCS6	5

B

ScienceDirect	
ID	Resultados
SCS1	0
SCS2	8
SCS3	8
SCS4	5
SCS5	7
SCS6	0

C

Fonte: Elaborado pelo autor.

A Tabela 13 traz os resultados obtidos utilizando SCS3, agora separados por ano e bases, sendo o quadro 13-A para o IEEE, o quadro 13-B para a ACM, 13-C para a ScienceDirect.

Tabela 13 – Resultados por ano utilizando SCS3 (Setembro de 2017)

IEEE	
Ano	Publicações
2010	0
2011	0
2012	0
2013	0
2014	0
2015	1
2016	0
2017	0
Total	1

A

ACM	
Ano	Publicações
2010	0
2011	0
2012	0
2013	0
2014	1
2015	0
2016	0
2017	0
Total	1

B

ScienceDirect	
Ano	Publicações
2010	0
2011	0
2012	0
2013	1
2014	0
2015	5 (3)
2016	2
2017	2
Total	10 (8)

C

Fonte: Elaborado pelo autor.

Como já mencionado, os resultados iniciais da ScienceDirect consideram não só artigos, mas também livros e enciclopédia, os quais foram desconsiderados. Sendo assim, na Tabela 13 os números entre parênteses são as quantidades reais de artigos obtidos por ano, assim como seu total e estes são os números considerados para análise.

3.2.4 Resultados para OpenNebula

Na Tabela 14 consta as *strings* de busca utilizadas para o gestor OpenNebula.

Tabela 14 – *Strings* de busca utilizadas para OpenNebula

ID	String de Busca
SN1	(fault OR defect OR FT) AND tolerance AND OpenNebula
SN2	fault tolerance AND OpenNebula
SN3	fault AND tolerance AND OpenNebula
SN4	fault AND (tolerance OR tolerant) AND OpenNebula
SN5	fault AND tolerance AND Reliability AND OpenNebula
SN6	((fault OR defect OR FT OR tolerance) OR "High Availability") AND OpenNebula

Fonte: Elaborado pelo autor.

Os resultados obtidos para essas *strings* de busca podem ser observados na Tabela 15. Para a base da ACM não se obteve resultados, enquanto para a base do IEEE as *strings* SN2, SN3 e SN4 para base do IEEE, e SN2 e SN3 para base da ScienceDirect, trouxeram resultados exatamente iguais. Mais uma vez a base do IEEE retornou um número elevado de resultados para SN1 e SN5. SN4 trouxe bons resultados apenas para base da ScienceDirect, assim como SN6 trouxe apenas para base do IEEE. Sendo assim, será utilizado SN3 para este caso.

Tabela 15 – Número de artigos obtidos em cada base de acordo com cada string de busca SN*n* (Outubro de 2017)

IEEE	
ID	Resultados
SN1	840
SN2	1
SN3	1
SN4	1
SN5	21.164
SN6	10

A

ACM	
ID	Resultados
SN1	0
SN2	0
SN3	0
SN4	0
SN5	0
SN6	0

B

ScienceDirect	
ID	Resultados
SN1	0
SN2	54
SN3	54
SN4	23
SN5	29
SN6	1

C

Fonte: Elaborado pelo autor.

A Tabela 16 traz os resultados obtidos utilizando a *string* SN3, separando os resultados pelo ano de publicação e pela base de origem. Para os resultados com o OpenNebula o quadro 16-A mostra os resultados para o IEEE, o quadro 16-B para a ACM (apesar de serem zeros), e para a ScienceDirect se tem o quadro 16-C, mais uma vez com os valores entre parênteses sendo a quantidade de artigos identificados.

Tabela 16 – Resultados por ano utilizando SN3 (Setembro de 2017)

IEEE	
Ano	Publicações
2010	0
2011	0
2012	0
2013	0
2014	1
2015	0
2016	0
2017	0
Total	1

A

ACM	
Ano	Publicações
2010	0
2011	0
2012	0
2013	0
2014	0
2015	0
2016	0
2017	0
Total	0

B

ScienceDirect	
Ano	Publicações
2010	1
2011	0
2012	10 (8)
2013	12 (9)
2014	9 (8)
2015	11 (8)
2016	10 (9)
2017	11
Total	64 (54)

C

Fonte: Elaborado pelo autor.

Por meio das Tabelas 10, 13, 16, é possível avaliar a evolução de publicações relacionadas com cada um dos gestores de computação em nuvem avaliados. Delas é possível identificar que o OpenStack tem sido o gestor mais referenciado. Tais informações podem auxiliar para tomada de decisão na hora de escolher um gestor para trabalhar.

A Tabela 17 sintetiza as *strings* selecionadas para aplicação nas bases científicas, assim como o número total de artigos obtido com cada uma delas.

Tabela 17 – Total de artigos obtidos com cada *string* escolhida

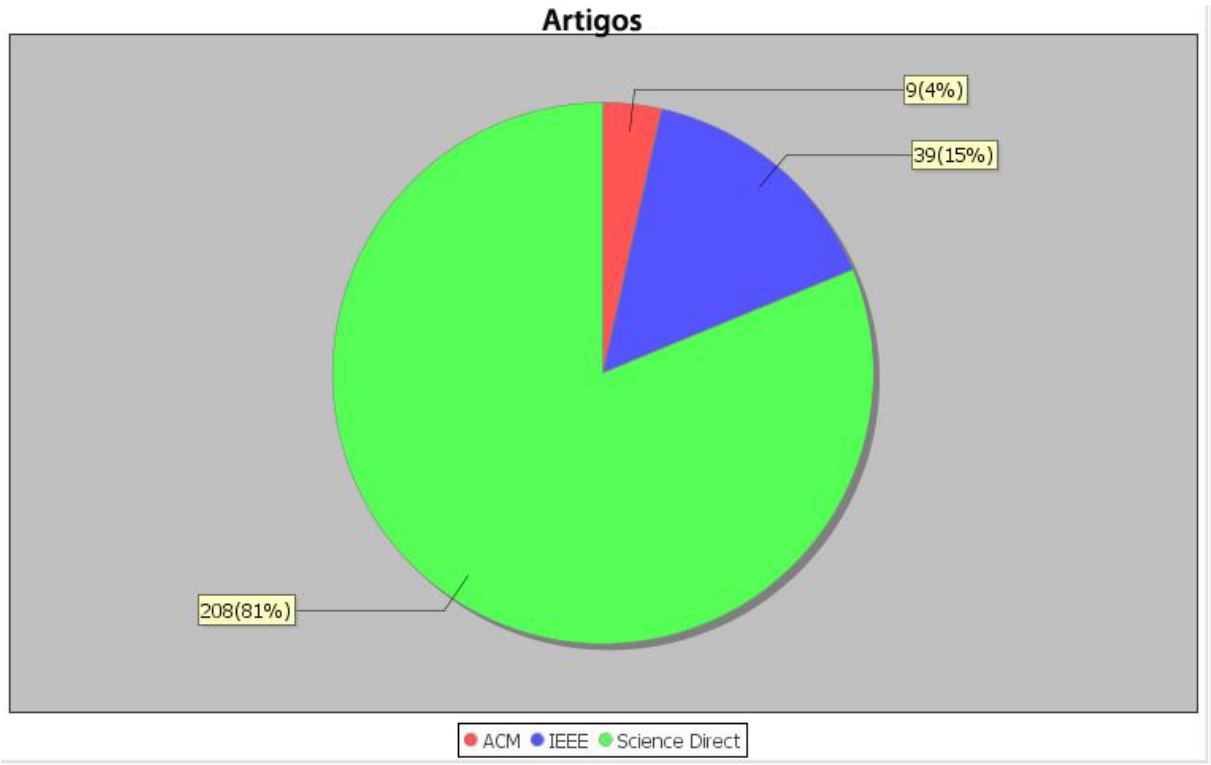
String de busca	ID	Nº de artigos obtidos
(“Fault tolerance” OR FT) AND “Cloud Computing”	SG5	71
fault AND tolerance AND OpenStack	SS3	120
fault AND tolerance AND CloudStack	SCS3	10
fault AND tolerance AND OpenNebula	SN3	55
Total	256	

Fonte: Elaborado pelo autor.

Deve ser observado que o total expresso na Tabela 17 é o valor bruto de artigos obtidos com permissão de acesso. Como mencionado anteriormente, livros e enciclopédias disponíveis na IEEE não possuem acesso liberado, sendo assim não fazem parte desse “total”. A Figura 10 traz em forma de gráfico os resultados referente ao total de artigos obtidos em cada base.

Visto que *strings* de buscas semelhantes foram utilizadas na mesma base várias vezes, é normal ao final da busca ter artigos duplicados. Sendo assim, a primeira filtragem feita nessa etapa inicial teve como objetivo remover artigos duplicados. De 256 artigos obtidos, 41 estavam duplicados (aproximadamente 16%). Portanto, a primeira etapa da RSL retornou um total de 215 artigos.

Figura 10 – Gráfico com total de artigos obtidos subi-divididos por base de dados.



(Elaborada pelo autor)

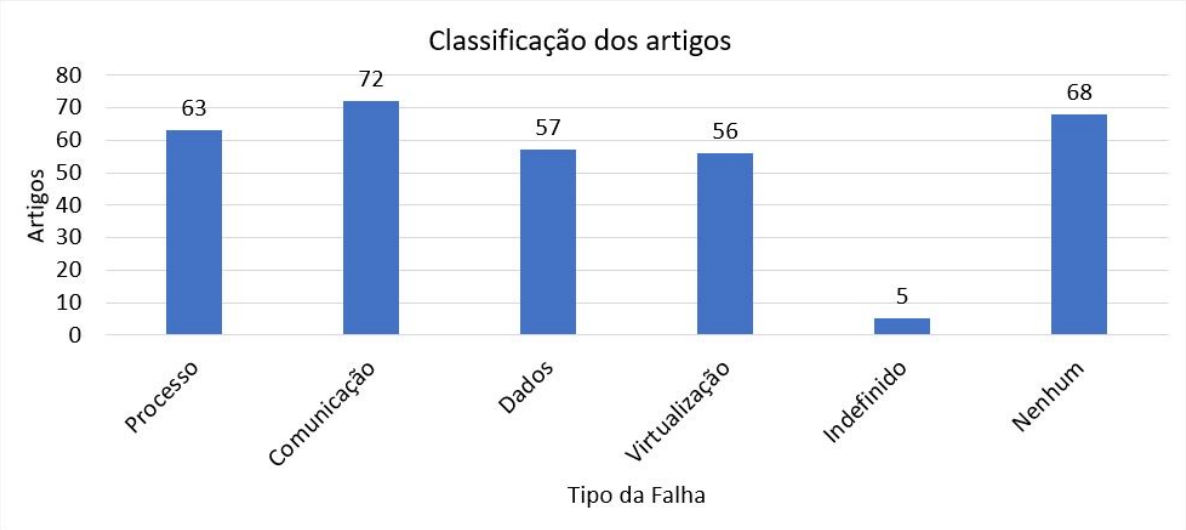
3.3 Resultados obtidos na segunda etapa da RSL

A segunda etapa da RSL envolve uma análise intermediária dos trabalhos identificados. Nessa análise se avança sobre o conteúdo dos artigos, de modo a permitir a identificação de características específicas neles tratadas. Assim, é possível identificar primeiro se existem trabalhos que, apesar de atenderem às *strings* de busca, não tratam do problema avaliado. Depois, ainda é possível categorizar artigos segundo critérios específicos, sendo que aqui o critério adotado foi o tipo de componente da nuvem que falha.

Dentre os 215 artigos selecionados na primeira etapa, 68 deles não tem relação com o tema desejado. A maioria deles trata de eficiência energética, fazendo apenas uma rápida citação ao problema de tolerância a falhas. Portanto, para a terceira etapa serão avaliados 147 artigos. Desses artigos tem-se que como alguns abordam mais de um tipo de falha, o total de casos de tipos de falha é maior do que 147.

Com base nos artigos classificados até aqui, tem-se que 63 artigos estão relacionados a falhas de processo, 72 a falhas de comunicação, 57 a falhas de dados (armazenamento), 56 a falhas de virtualização. Além disso, em 5 artigos não foi possível identificar o tipo de falha tratada, ou seja, são trabalhos que abordam tolerância a falhas de modo genérico. Na Figura 11 é apresentada a classificação descrita neste parágrafo, incluindo os 68 trabalhos não relacionados ao tema.

Figura 11 – Quantidade de artigos publicados segundo origem da falha.



(Elaborada pelo autor)

Analisando-se a Figura11 é possível concluir que para computação em nuvem as falhas relacionadas a comunicação são a maior preocupação, aparecendo em cerca de 50% dos artigos selecionados. Apesar disso, a dispersão entre os tipos de falha é relativamente pequena, com falhas de virtualização, que é a menos referenciada, sendo tratada em 38% dos trabalhos. O próximo passo foi avaliar qual técnica é mais utilizada de acordo com cada tipo de falha, demandando uma análise mais profunda dos 147 artigos selecionados.

3.4 Resultados obtidos através da terceira etapa da RSL

Esta etapa consiste na leitura integral de cada artigo, de forma a identificar a adequação real de cada trabalho ao processo de catalogação sendo feito aqui. Com esse procedimento foi possível reduzir ainda mais significativamente o total de trabalhos que compõem de fato o corpo desta análise. Na prática foram excluídos nesta etapa outros 93 artigos, sendo que o motivo para exclusão são:

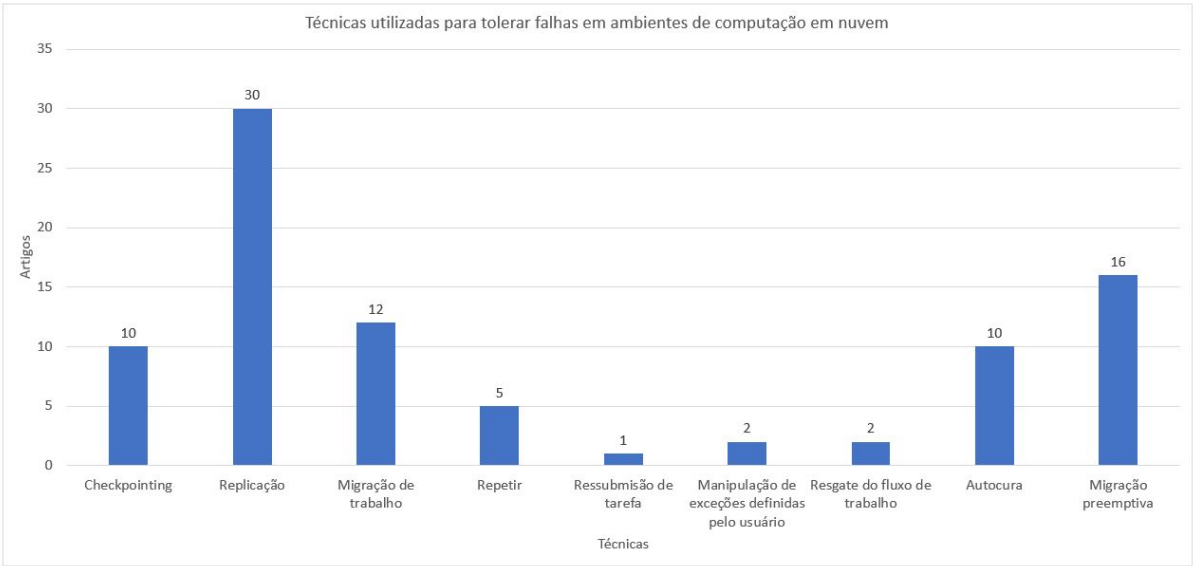
- **Artigos do tipo RSL**, que são aqueles que apenas exploram o estado da arte, sendo classificados como *surveys* e semelhantes. Visto que artigos nesta classificação não detalham os sistemas utilizadas para tolerar falhas e muitas vezes não citam algum tipo de solução, não é razoável mantê-los na anaĺise. Foram contados 35 artigos nesta categoria; e
- **Artigos sem foco no tema**, sendo que aqui os 58 artigos foram excluídos por motivos diversificados, como:
  - **Não tratavam de tolerância a falhas em nuvem:** artigos que embora tratassem de tolerância a falhas não a contextualizavam com computação em nuvem;

- **Relacionados a outros assuntos semelhantes ao desejado:** faziam menção a computação em nuvem e tolerância a falhas, porém o foco era em aspectos como balanceamento de carga, eficiência energética, entre outros;
- **Sem conclusões significativas:** elaboravam um sistema tolerante a falhas para nuvem porém não apresentavam resultados ou avaliações concretas, deixando os mesmos para trabalhos futuros; e
- **Relacionados a nuvem mas não a tolerância a falhas:** trabalhavam com computação em nuvem, porém apenas faziam menção a "tolerância a falhas", sem qualquer proposta nessa área.

Portanto, ao final da RSL obteve-se o total de 54 artigos classificados. Houve casos em que autores criaram um sistema e posteriormente os mesmos autores ou outros autores fizeram melhorias neste mesmo sistema. Nestes casos a análise ocorreu apenas uma vez, como por exemplo na subseção 4.2.4 com a proposta "SwiftER" e na subseção 4.4.1 com a proposta "Sistema de tolerância a falhas energética para computação de alto desempenho". O resultado disso é que no Capítulo 4 estão apresentadas 52 soluções propostas.

Como um subproduto da terceira etapa da RSL foram criados dois gráficos. O primeiro, apresentado na Figura 12, traz dados referentes às técnicas que foram utilizadas para tolerar falhas em cada proposta estudada, frisando que um sistema pode usar uma ou mais técnicas para tolerar falhas. Desse gráfico é possível observar que a maioria das soluções faz uso de replicação (30 propostas) ou de migração (28 propostas entre migração preemptiva e migração de trabalho), as quais devem ser vistas com mais cuidado portanto.

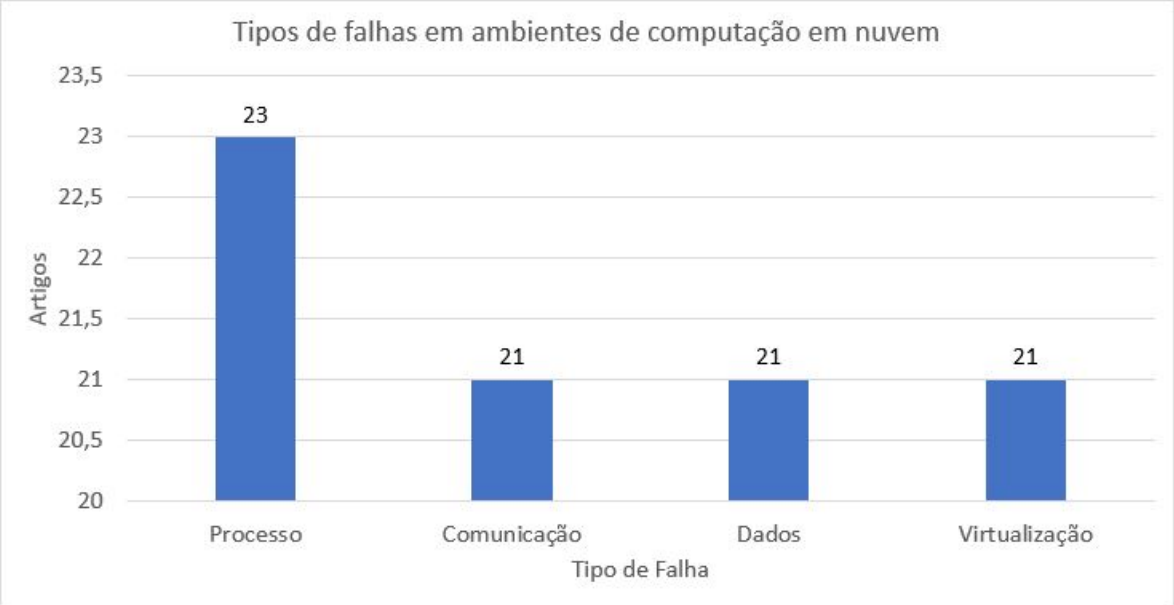
Figura 12 – Técnicas utilizadas para falhas tolerar falhas considerando todas propostas estudadas.



(Elaborada pelo autor)

O segundo gráfico gerado diz respeito a uma nova classificação quanto ao tipo de falha tratada nos artigos selecionados. Ele pode ser visto na Figura 13, que atualiza o gráfico expresso na Figura 11. Uma observação importante a fazer aqui é que agora cada tipo de falha é tratado por um total entre 37% (falhas de dados) e 43% (falhas de processo) dos artigos selecionados, diminuindo ainda mais a dispersão entre a origem das falhas.

Figura 13 – Tipos de falhas em ambientes de computação em nuvem atualizado.



(Elaborada pelo autor)

3.5 Considerações finais

Neste capítulo foi apresentada a maneira em que a RSL foi conduzida, assim como os resultados de cada etapa. Para tratar falhas em computação em nuvem a técnica de replicação é a mais utilizada, possivelmente por conta da sua simplicidade. A maioria das propostas tratam falhas relacionadas a processos, em seguida de comunicação, virtualização e dados (empatados). No próximo Capítulo será apresentada a análise detalhada dos trabalhos identificados durante a RSL.

4 Análise de sistemas para tolerância a falhas em ambientes de computação em nuvem

Neste capítulo são apresentadas as ferramentas e sistemas provenientes dos artigos selecionados para avaliação com a revisão descrita no Capítulo 3. As ferramentas foram subdividas de acordo com o tipo de falha podendo ser falhas relacionadas a: dados, processo, comunicação e virtualização. Dentro de cada item, serão subdivididas de acordo com a técnica utilizada para o tratamento de falhas, ou seja: *checkpointing*, replicação, migração de trabalho, repetir, autocura e migração preemptiva, por exemplo. Após essas seções se acrescentou uma seção para descrever o protótipo de *website* que servirá como local para a realização de buscas guiadas pela solução a um dado problema.

4.1 Metodologia de classificação

Para apresentar de forma mais clara as soluções identificadas após o trabalho de revisão sistemática, adotou-se neste capítulo uma abordagem de classificação baseada inicialmente no tipo de falha tratada. Assim, a análise está separada nas seguintes seções:

Seção 4.2 – sistemas que tratam falhas relacionadas a dados;

Seção 4.3 – sistemas que tratam falhas relacionadas a comunicação;

Seção 4.4 – sistemas que tratam falhas relacionadas a processos; e

Seção 4.5 – sistemas que tratam falhas relacionadas a virtualização.

Como em vários casos o sistema trata falhas relacionados a um ou mais dos problemas indicados, a metodologia usada faz a descrição numa seção específica e apenas uma referência em outra seção. Por exemplo, falhas em processos muitas vezes implicam em trabalhar com questões de virtualização, como acontece com a proposta “Sistema de tolerância a falhas energética para computação de alto desempenho”, que é descrita na subseção 4.4.1 e posteriormente citada em 4.5.

Para cada uma das seções principais serão criadas subseções que agruparão propostas que usam uma mesma técnica para a solução da falha. Em cada uma dessas subseções serão então apresentadas e descritas as propostas que apresentam um mesmo foco, o que pode facilitar o trabalho comparativo. Mais uma vez, como vários trabalhos formulam soluções usando técnicas diversas, o que se faz é incluir uma dada proposta dentro da seção e subseção que corresponder ao seu objetivo mais relevante, sendo a proposta apenas citada nas demais situações. Para melhor

compreensão de como fica essa classificação, o exemplo a seguir mostra como ficará a divisão das informações:

Seção 4.2 Sistemas que tratam falhas relacionadas a dados

4.2.1 Uso de *Checkpointing*

- Proposta A;
- Proposta B.

4.2.2 Uso de Migração Preemptiva

- Proposta D;
- Proposta D;
- Proposta E.

Na sequência são apresentadas as soluções propostas identificadas com o trabalho de revisão sistemática da literatura.

4.2 Falhas relacionadas a dados

Nesta seção são descritas as propostas para tolerar falhas relacionadas a dados. As propostas aqui listadas tratam essas falhas por *checkpointing*, autocura, migração preemptiva, repetir, manipulação de exceções definidas pelo usuário, resgate do fluxo de trabalho, migração de trabalho, ou replicação. Algumas dessas propostas serão apresentadas com detalhes em seções posteriores, sendo apenas nominadas aqui.

4.2.1 Uso de *Checkpointing*

O *checkpointing* é ideal para situações com grande volume de dados, visto que em sua abordagem mais simples, garante a recuperação do ambiente a partir do ponto de verificação mais recente. Nessa categoria tem-se a abordagem IGG (JAVADI; ABAWAJY; BUYYA, 2012), apresentada a seguir.

InterGrid Gateway (IGG)

Javadi, Abawajy e Buyya (2012) propuseram uma infraestrutura de nuvem híbrida escalável, assim como políticas de provisionamento de recursos para garantir QoS. O tratamento de falhas é feito utilizando *checkpointing*, que reinicia a solicitação de uma nova VM a partir do ultimo momento de disponibilidade. Modificou-se os algoritmos de agendamento de preenchimento para suportar o mecanismo de perfeito *checkpointing* e fornecer um ambiente tolerante

a falhas para atender solicitações em nuvem privada. A Tabela 18 traz de forma resumida as principais características do sistema.

Tabela 18 – Resumo do sistema IGG.

Técnicas utilizadas	*Checkpointing*
Gestor de nuvem compatível	OpenNebula e Eucalyptus
Modelo de programação	Java
Problema Solucionado	QoS; Provisionamento de Recursos
Tipo de Falha tratada	Dados
Vantagem	Suporte para arquitetura híbrida; melhoria de QoS de 32%
Desvantagem	Possui baixo QoS considerando apenas nuvens privadas

Fonte: Elaborado pelo autor.

4.2.2 Migração Preemptiva

A migração preemptiva possibilita migrar dados com baixo custo computacional. Nessa categoria tem-se a abordagem HySARC^2 (VASILE et al., 2015), apresentada a seguir.

HySARC^2

O objetivo do *HySARC*^{2} é melhorar o agendamento em um determinado ambiente em nuvem por meio de sistemas de agrupamento e rotulagem de serviços, garantindo o uso adequado de recursos e consequentemente satisfazer os requisitos dos usuários e os interesses do provedor de serviços. As novas abordagens de agendamento têm de monitorar a estrutura do sistema e ajustar o planejamento de acordo com o uso em tempo real. Isso também auxilia em um sistema mais tolerante a falhas, visto que certas mudanças nem sempre são esperadas. Para criação do sistema, utilizaram técnicas como: grafos acíclicos dirigidos e *k-means* (VASILE et al., 2015). A Tabela 19 traz de forma resumida as principais características do sistema.

Tabela 19 – Resumo do *HySARC*^{2}.

Técnicas utilizadas	Migração preemptiva
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Provisionamento de recursos
Tipo de Falha tratada	Processos e Dados
Vantagem	Adequado para computação distribuída heterogênea computação de alto desempenho;
Desvantagem	Em certos pontos de execução ocorre sobrecarga no sistema

Fonte: Elaborado pelo autor.

4.2.3 Migração de trabalho

Se por algum motivo uma determinada tarefa não pode ser completamente executada é feita a migração da tarefa para uma máquina nova. Sendo assim, é importante que o ambiente possua número adequado de máquinas e/ou VMs. Nessa categoria tem-se a abordagem SkyCDS (GONZALEZ et al., 2015), apresentada a seguir.

SkyCDS (Protótipo)

É um sistema voltado para serviço de entrega de conteúdo (CDS), baseado em sobreposição de publicação/assinatura em armazenamento em nuvem. A entrega é dividida em camadas de fluxo de metadados e armazenamento de conteúdo. O sistema é capaz de diminuir a sobrecarga da dispersão de conteúdo e recuperação de processos.

O SkyCDS possui três níveis para mascarar falhas de componentes do CDS, que podem ser escolhidos de acordo com a necessidade, sendo:

- **Primeiro nível:** mascara as interrupções de serviço de usuários finais e editores pelas técnicas de informação de dispersão aplicadas à entrega e recuperação de fluxos de trabalho;
- **Segundo nível:** mascara o desastre; e
- **Terceiro nível:** mascara os efeitos colaterais dos atrasos de diversidade geográfica de todos os usuários ao armazenar em cache o conteúdo e movê-los para perto dos usuários finais.

A Tabela 20 traz de forma resumida as principais características do sistema.

Tabela 20 – Resumo do sistema SkyCDS.

Técnicas utilizadas	Migração de trabalho
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Uma nova maneira de comparar opções de armazenamento/entrega da avaliação de riscos
Tipo de Falha tratada	Dados e Processo
Vantagem	Redução de carga; minimização de efeitos colaterais causados por interrupções
Desvantagem	Trata-se de um protótipo

Fonte: Elaborado pelo autor.

4.2.4 Uso de Autocura

A técnica de autocura possibilita a recuperação do ambiente com pouca ou nenhuma intervenção humana. Nessa categoria tem-se as abordagens DARGOS (POVEDANO-MOLINA et al., 2013) e um protótipo para tratamento automático de anomalias (GULENKO et al., 2016), apresentadas a seguir.

A técnica de autocura traz como principal característica a recuperação de falhas de forma automática. No contexto de recuperação de falhas em dados, as propostas encontradas utilizam a técnica de autocura sempre em conjunto com outra técnica. Nessa categoria tem-se as abordagens DCR2S (GILL; SINGH, 2016) e Protótipo para tratamento automático de anomalias (GULENKO et al., 2016) apresentadas a seguir.

Distributed Architecture for Resource manaGement and mOnitoring in cloudS (DARGOS)

O sistema foi projetado para satisfazer os principais requisitos de um ambiente em nuvem e, ao mesmo tempo, apresenta baixa latência e sobrecarga. Baseado no paradigma de publicação/assinatura, o ambiente permite escolher zonas a serem monitoradas e outros recursos de comunicação, com as taxas de atualização e conjunto de sensores. Esses sensores coletam estatísticas relacionadas a aplicativos, como número de solicitações por segundo e seu tempo de atividade, permitindo assim detectar situações de sobrecarga do servidor e possíveis falhas. A Tabela 21 traz de forma resumida as principais características do sistema.

Tabela 21 – Resumo sistema DRAGOS.

Técnicas utilizadas	Autocura e migração preemptiva
Gestor de nuvem compatível	OpenStack
Modelo de programação	C, JavaScript e Python
Problema Solucionado	Dados
Tipo de Falha tratada	Dados
Vantagem	Arquitetura descentralizada; ativação de decisões de agendamento imediato; suporte para múltiplos consumidores via *multicast*;suporte para filtragem de dados
Desvantagem	Tráfego elevado em comunicação *unicast*

Fonte: Elaborado pelo autor.

Protótipo para tratamento automático de anomalias

A proposta visa por meio da utilização de inteligência artificial supervisionada e não-supervisionada, criar um sistema para fazer o mascaramento automático e em tempo real de falhas, sendo o foco para ambientes de *Network Function Virtualization* (NFV). O sistema facilita a detecção de anomalias em tempo real em implementação de NFV em larga escala.

Os resultados obtidos com mineração de dados são utilizados para selecionar e executar

rotinas de restauração de acordo com a necessidade. Inicialmente, o sistema foi disponibilizado apenas para o OpenStack, podendo ser generalizado para diferentes sistemas de computação em nuvem (GULENKO et al., 2016). A Tabela 22 traz de forma resumida as principais características do sistema.

Tabela 22 – Resumo Protótipo para tratamento automático de anomalias.

Técnicas utilizadas	Replicação, Autocura e migração preemptiva
Gestor de nuvem compatível	OpenStack
Modelo de programação	Python
Problema Solucionado	Solução envolvendo inteligência para NFV principalmente
Tipo de Falha tratada	Comunicação; Processo; Dados; Virtualização
Vantagem	Mascaramento automático da falha, evitando interrupções no sistema
Desvantagem	Trata-se de um protótipo

Fonte: Elaborado pelo autor.

4.2.5 Uso de Replicação

A replicação é uma técnica simples de ser implementada e utilizada, podendo ser aplicada em casos distintos dentro do contexto de recuperação de falhas relacionadas a dados. Por exemplo, pode-se efetuar uma (ou *n*) cópia(s) idêntica(s) da base de dados, garantindo assim, mesmo que de maneira altamente custosa, um ambiente com alta disponibilidade. Nessa categoria tem-se a abordagem SprintNet (WANG et al., 2015), apresentada na seção 4.3.4, além de propostas como FIR3 (VIJAYAKUMAR; SRINIVASAGAN; SABARIMUTHUKUMAR, 2015), DCR2S (GILL; SINGH, 2016), Morpho (LU et al., 2014), Tahoe-LAFS (SELIMI et al., 2015), Algoritmo híbrido baseado em MapReduce (ZHANG et al., 2014), GFS (NAKANISHI et al., 2014), Sistema de armazenamento privado multicamadas (GONZALEZ et al., 2013) e SwiftER (KULKARNI; BHOSALE, 2016; DATTA; CHO, 2016), que são apresentados a seguir.

Tahoe-LAFS

É um sistema open-source para computação em nuvem, voltado para tolerância a falhas em nós de armazenamento. Oferece acesso por meio de várias interfaces (Web, S.O, SSH), garantindo privacidade e segurança ao fazer criptografia de dados no lado do cliente. Em experimentos feitos em diferentes condições, a aplicação conseguiu recuperar todos os diferentes tamanhos de arquivos, mesmo em uma rede comunitária (SELIMI et al., 2015).

O sistema de replicação é baseado em *erasure code*, em que todo arquivo novo é separado em *n shares* distintos, podendo ser recuperado a partir de qualquer *share*. A Tabela 23 traz de forma resumida as principais características do sistema.

Tabela 23 – Resumo Tahoe-LAFS

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Universal
Modelo de programação	Python
Problema Solucionado	Recuperação de dados (disponibilidade)
Tipo de Falha tratada	Dados
Vantagem	Fácil processo de replicação; compatível com grande quantidade de dados
Desvantagem	Ausência de cache

Fonte: Elaborado pelo autor.

SwiftER

A proposta foi dada inicialmente por Kulkarni e Bhosale (2016) e tinha como objetivo reduzir os dados armazenados com a replicação. Por utilizar a técnica de *Erasure*, o sistema ganhou o nome SwifER (Swift Erasure). Os dados duplicados pelo sistema são armazenados em forma de Raid, podendo diminuir o espaço de armazenamento em 1.2x até 3x, mantendo a confiabilidade e alta disponibilidade. Seu funcionamento ocorre na camada Swift do OpenStack.

Posteriormente, Datta e Cho (2016) propuseram a adaptação dinâmica do grau de redundância do Erasure no Swift. Os testes feitos sugerem melhoria de até 20% no desempenho, sem perda de integridade e disponibilidade. A Tabela 24 traz de forma resumida as principais características do sistema.

Tabela 24 – Resumo do sistema SwiftER.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	Python
Problema Solucionado	Melhoria em espaço de armazenamento; mais eficiente que a replicação “tradicional”
Tipo de Falha tratada	Dados e Comunicação
Vantagem	Utiliza menos espaço de armazenamento
Desvantagem	Não recomendada para grandes sistemas; Voltado para backup secundário

Fonte: Elaborado pelo autor.

Sistema de armazenamento privado multicamadas

O sistema proposto por Gonzalez et al. (2013) é baseado em uma arquitetura hierárquica multicamada, em que cada nível representa um serviço de armazenamento diferente. possibilitando a transferência de informações redundantes entre os níveis. A disponibilidade dos arquivos

é garantida por meio do sistema unificado que permite a recuperação de diferentes categorias de falhas no serviço. A Tabela 25 traz de forma resumida as principais características do sistema.

Tabela 25 – Resumo do sistema de armazenamento em multicamadas.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	Java
Problema Solucionado	Dados e Comunicação
Tipo de Falha tratada	Recuperação de arquivos em diferentes camadas
Vantagem	Não necessita da replicação da IDA, chunks e REST
Desvantagem	Pode acarretar em custos elevados de acordo com a camada utilizada para recuperação de dados

Fonte: Elaborado pelo autor.

Gluster File System (GFS)

O sistema foi projetado baseado em conceitos de computação de alto desempenho. Ao mesmo tempo conta com uma estrutura simplificada, semelhante com a do RAID10. De acordo com Nakanishi et al. (2014), em testes comparados com o Swift, o GFS mostrou E/S de dados até 4.5x mais rápido.

A replicação é feita com base em arquivos, onde uma *hash* distribuída é utilizada para alocar estaticamente espaços elementares denominados "tijolos" para o espaço inteiro dos nomes de arquivos armazenados. A Tabela 26 traz de forma resumida as principais características do sistema.

Tabela 26 – Resumo do sistema GFS.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	C/C++
Problema Solucionado	Baixo uso de CPU (inferior a 20%); E/S mais rápida
Tipo de Falha tratada	Dados
Vantagem	E/S mais rápidas; descentralizado
Desvantagem	Dados de longo prazo utilizam armazenamento de Raid "comum" (mais lento)

Fonte: Elaborado pelo autor.

Algoritmo híbrido baseado em MapReduce

Neste sistema, o algoritmo é baseado em MapReduce utilizando dois componentes: *Top Down Specialization* (TDS) e *Bottom-Up Generalizastion* (BUG). O algoritmo proposto é capaz de normalizar sub-árvores. De acordo com os testes feitos pelos autores Zhang et al. (2014), o

ambiente melhora significativamente a eficiência e escalabilidade em relação às abordagens já existentes. A Tabela 27 traz de forma resumida as principais características do sistema.

Tabela 27 – Resumo do algoritmo híbrido baseado em MapReduce.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Normalização de sub-árvores
Tipo de Falha tratada	Dados
Vantagem	Maior eficiência e escalabilidade quando comparado com outros sistemas existentes
Desvantagem	O grande conjunto de dados prejudica a escalabilidade

Fonte: Elaborado pelo autor.

Morpho

A proposta é basicamente uma modificação do *framework* Hadoop e MapReduce. O Morpho foi projetado com o objetivo de melhorar a cooperatividade das camadas de computação e armazenamento, isto é, as tarefas do *MapReduce* podem buscar informações sobre a topologia da rede das máquinas físicas e as alocações das VMs.

O sistema também utiliza estratégias complementares para alocação de dados das VMs, gerando um mapeamento melhor e redução na localidade de entrada. A Tabela 28 traz de forma resumida as principais características do sistema.

Tabela 28 – Resumo do sistema Morpho.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenNebula
Modelo de programação	Java
Problema Solucionado	Melhoria em alocação de recursos e armazenamento de dados
Tipo de Falha tratada	Dados e Virtualização
Vantagem	Redução significativa no tráfego de rede (60% a 70%)
Desvantagem	Possui baixa escalabilidade; em outros ambientes não conseguiu-se alcançar o mesmo nível de localidade de dados quando comparado as configurações de testes

Fonte: Elaborado pelo autor.

Dynamic Cost-Aware Re-Replication and Re-balancing Strategy (DCR2S)

O sistema é compatível com nuvens heterogêneas e utiliza uma estratégia de replicação dinâmica otimizada, que identifica o número mínimo de réplicas necessárias para garantir a disponibilidade desejada. O algoritmo de replicação é subdividido em quatro fases:

- **Primeira fase:** o arquivo de dados é identificado e é tomada uma decisão quanto a sua replicação. Essa decisão é feita pelo gerenciador de réplicas com a ajuda do catálogo de réplicas. O catálogo de réplicas mantém as informações de acesso de cada arquivo de dados em intervalos de tempo regulares;
- **Segunda fase:** feita a identificação, o gerenciador de réplicas determina o número apropriado de novas réplicas a serem criadas para este arquivo de dados. Essa decisão é tomada com base na probabilidade de disponibilidade do arquivo e no fator de réplica do arquivo de dados. Para determinar o número adequado, o gerenciador também utiliza a probabilidade antiga de disponibilidade do artigo;
- **Terceira fase:** nesta fase o gerenciador de réplicas define onde a nova réplica do arquivo será armazenada. Essa decisão é tomada com base nos detalhes de acesso do arquivo; e
- **Quarta fase:** é feita a otimização da replicação utilizando o algoritmo *Knapsack*.

A Tabela 29 traz de forma resumida as principais características do sistema.

Tabela 29 – Resumo do sistema DCR2S.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Universal
Modelo de programação	Não informado pelos autores
Problema Solucionado	Recuperação de dados (disponibilidade)
Tipo de Falha tratada	Dados
Vantagem	Diminuição do custo de replicação
Desvantagem	Disponibilidade diminui de acordo com a quantidade de réplicas adicionais a mais que o adequado

Fonte: Elaborado pelo autor.

Fuzzy Inference based Reliable Replica Replacement (FIR3)

A perda de dados ou interrupções de serviços podem ocorrer com frequência na computação baseada na internet, para solucionar este problema os autores Vijayakumar, Srinivasagan e Sabarimuthukumar (2015) criaram uma nova técnica de replicação de dados baseada no Sistema de Inferência Fuzzy.

A principal ideia é manter cada réplica de dados nas diferentes zonas de disponibilidade. O algoritmo usa inferência difusa auxilia na solução de problemas de inconsistência de espaço. A replicação é implantada no ambiente de pilha da nuvem, portanto, toda a tolerância a falhas do sistema será aprimorada por esta técnica de replicação. A Tabela 30 traz de forma resumida as principais características do sistema.

Tabela 30 – Resumo do sistema FIR3.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Cloudstack
Modelo de programação	Java
Problema Solucionado	Substituição do arquivo é efetivamente tratada usando Fuzzy Inference System e consistência efetiva entre réplicas.
Tipo de Falha tratada	Dados
Vantagem	Sincronização entre réplicas com menor taxa de erro e taxa de dados faltantes
Desvantagem	Custo de replicação e chance de sucesso é proporcional ao tamanho do dado

Fonte: Elaborado pelo autor.

Medical Image File Accessing System (MIFAS)

O Sistema proposto é baseado no Hadoop Distributed File System (HDFS) e traz melhorias em armazenamento de imagens médicas, estabilidade durante transmissões, e confiabilidade, além de proporcionar uma interface de fácil gerenciamento.

O serviço de localização de replicação faz duplicatas automaticamente de uma nuvem para outra (mesmo se distinta) quando as imagens médicas são carregado para o MIFAS.

Os resultados dos experimentos comprovam que o sistema possui alta confiabilidade e tolerância a falhas (YANG et al., 2015). A Tabela 31 traz de forma resumida as principais características do sistema.

Tabela 31 – Resumo do sistema MIFAS.

Técnicas utilizadas	Replicação e Autocura
Gestor de nuvem compatível	OpenNebula
Modelo de programação	Não informado pelos autores
Problema Solucionado	Transferência de dados (imagens)
Tipo de Falha tratada	Dados
Vantagem	Fácil gerenciamento; baixo custo; fácil compartilhamento para nuvens privadas
Desvantagem	Em relação a outros sistemas o MIFAS possui eficiência inferior

Fonte: Elaborado pelo autor.

Sistema baseado no princípio 80/20

Mesbahi, Rahmani e Hosseinzadeh (2017) testaram um sistema baseado na regra 80/20 (80% das falhas dos clusters provêm de 20% das máquinas físicas).

A ideia é auxiliar na identificação das máquinas físicas propensas a falhas nos clusters. As máquinas são subdivididas em dois subconjuntos: confiáveis (70% a 80% das máquinas) e arriscados (20% a 30% das máquinas). O subgrupo confiável inclui uma zona altamente confiável, fornecendo alta disponibilidade para trabalhos latenciosos. A Tabela 32 traz de forma resumida as principais características do sistema.

Tabela 32 – Resumo do sistema baseado no princípio 80/20.

Técnicas utilizadas	Replicação, migração de trabalho e ressubimissão de tarefa
Gestor de nuvem compatível	OpenStack
Modelo de programação	Java
Problema Solucionado	Prevenção de falhas; melhoria de confiabilidade e disponibilidade em sistemas distribuídos de larga escala
Tipo de Falha tratada	Dados
Vantagem	Redução de manutenção; redução de custos; maior disponibilidade; redução na latência geral do sistema
Desvantagem	Recomendado apenas para grandes sistemas

Fonte: Elaborado pelo autor.

Middleware de nuvem para garantir desempenho e alta disponibilidade de aplicativos soft em tempo real

O *software* proposto por An et al. (2014) apresenta uma estrutura capaz de implementar réplicas de máquinas virtuais de acordo com algoritmos flexíveis pré-definidos pelos usuários. O sistema inclui um Gerenciador de Falhas Local (LFM) para cada *host* e um Gerenciador de Falhas Globais Replicado (GFM) para gerenciar os *clusters* de máquinas físicas. Os LFMs utilizam informações (CPU, memória, rede, armazenamento e processos) dos *hosts* físicos e de VMs que são coletadas diretamente do *hypervisor*.

O sistema implanta automaticamente réplicas de máquinas virtuais em *datacenters* de forma a otimizar os recursos, garantindo a disponibilidade e a capacidade de resposta. Além de utilizar uma solução baseada em *Continuous Checkpointing*, isto é, uma solução de alta disponibilidade na qual em intervalos frequentes de tempo a execução da VM principal é pausada para capturar seu estado. A Tabela 33 traz de forma resumida as principais características do sistema.

Tabela 33 – Resumo do Middleware de nuvem para garantir desempenho e alta disponibilidade de aplicativos soft em tempo real.

Técnicas utilizadas	Replicação, *checkpointing*, manipulação de exceções definidas pelo usuário
Gestor de nuvem compatível	OpenStack e OpenNebula
Modelo de programação	C++
Problema Solucionado	Otimização de recursos; informações de utilização de compartilhamento de recursos em tempo real
Tipo de Falha tratada	Dados; Virtualização; Processos
Vantagem	Mantém QoS elevado
Desvantagem	Não existe garantia de que opere em ambientes de produção

Fonte: Elaborado pelo autor.

mOSAIC

Para lidar com vários cenários de uso em nuvem e fornecer soluções adicionais para portabilidade, Petcu et al. (2013) projetaram o mOSAIC, cujas principais características são vistas na Tabela 34. As premissas a serem cumpridos pelo mOSAIC são aplicação, programação, monitoramento e implantação. Mais precisamente ele garante:

- Uma ferramenta de gerenciamento para implantação em diferentes Nuvens;
- Monitoramento do acordo de nível de serviço (SLA); e
- Provisionamento automático, escalabilidade ao nível dos componentes da aplicação e portabilidade de dados;
- Mesmas ferramentas para o simulador de desktop e a nuvem real;

Tabela 34 – Resumo do sistema mOSAIC.

Técnicas utilizadas	Replicação e Resgate do Fluxo de Trabalho
Gestor de nuvem compatível	OpenNebula e OpenStack
Modelo de programação	Java
Problema Solucionado	Melhor custo benefício,implementação em nuvens múltiplas, autenticação (prevenção)
Tipo de Falha tratada	Comunicação, Processo e Dados
Vantagem	Pode ser escalado para múltiplas máquinas
Desvantagem	A adoção de nuvens em grande escala é dificultada visto que os códigos legados precisam ser reescritos para tirar proveito da elasticidade

Fonte: Elaborado pelo autor.

Pilot-Data

O Pilot-Data aborda desafios fundamentais de co-colocação e agendamento de dados e computação em ambientes heterogêneos e distribuídos com interoperabilidade e extensibilidade como preocupações de primeira ordem. Pilot-Data é uma extensão da abstração Pilot-Job para suportar o gerenciamento de dados em conjunto com tarefas de computação.

O sistema da suporte às aplicações na exploração de paralelismo de dados, por exemplo, permitindo que elas particionem e/ou repliquem os dados de forma eficiente. Desta forma, permite que as aplicações otimizem processos que geram movimentação de dados, para criar uma réplica Pilot-Local do conjunto de dados, fazendo com que os dados sejam processados mais rápido. Também conta com os recursos de confiabilidade incorporados ao serviço de transferência que reinicia automaticamente as transferências com falha. . A Tabela 35 traz de forma resumida as principais características do sistema.

Tabela 35 – Resumo do sistema Pilot-Data.

Técnicas utilizadas	Replicação e Repetir
Gestor de nuvem compatível	Eucalyptus e Opensatck
Modelo de programação	Não informado pelos autores
Problema Solucionado	Alocação de recursos
Tipo de Falha tratada	Dados
Vantagem	Compatível com sistemas heterogêneos
Desvantagem	A necessidade de cada tarefa puxar dados grandes e remotamente cria um gargalo

Fonte: Elaborado pelo autor.

4.3 Falhas relacionadas a comunicação

Nesta seção serão descritas as propostas para tolerar falhas relacionadas a comunicação. As propostas aqui listadas tratam essas falhas por *checkpointing*, autocura, migração preemptiva, repetir, manipulação de exceções definidas pelo usuário, resgate do fluxo de trabalho, migração de trabalho, ou replicação. Algumas dessas propostas foram apresentadas em seções anteriores, ou serão apresentadas em seções posteriores e serão aqui apenas nominados.

4.3.1 Uso de *Checkpointing*

Com o *checkpointing* é possível tratar problemas relacionados a tráfegos de dados de forma que não haja necessidade da reinicialização integral de processos de transferência de dados, por exemplo. Nessa categoria podem ser destacadas as abordagens: Sistema para resgate de *spot* inesperados usando instâncias de *spot* heterogêneos e excesso de provisão (QU; CALHEIROS; BUYYA, 2016), e SymVirt (TAKANO et al., 2012), apresentadas a seguir.

ONHelp

O sistema apresentado por Zhao et al. (2014) auxilia o OpenNebula em questões como: segurança, monitoramento de VM, tolerância a falhas e armazenamento seguro.

Em relação a tolerância a falhas, o serviço trata de falhas de relacionadas a *software* e a VMs, sendo tratada utilizando 3 mecanismos:

- ***Lightweight VM checkpoint*:** salva e restaura o estado das VMs de maneira rápida ao mesmo tempo que otimiza o custo (tempo e espaço). Utiliza o mecanismo *"copy-while-writing"* para economizar dados da memória da VM;
- ***VM hot back-up*:** usa migração em tempo real e cria VMs "sombra" (mesmas configurações da VM original) para ficar no lugar do nó que está sendo migrado, esta fica como auxiliar durante o processo de migração; e
- ***Virtual cluster collaborative backup***: usa tecnologia do mecanismo *Lightweight VM checkpoint* para efetuar o backup de todas as VMs do *cluster*.

A Tabela 36 traz de forma resumida as principais características do sistema.

Tabela 36 – Resumo do sistema ONHelp.

Técnicas utilizadas	*Checkpointing*, migração de trabalho, manipulação de exceções definidas pelo usuário, migração preemptiva
Gestor de nuvem compatível	OpenNebula
Modelo de programação	Não informado pelos autores
Problema Solucionado	Trata de forma geral diversas falhas
Tipo de Falha tratada	Processo, Comunicação e Virtualização
Vantagem	Tratamento para diversos tipos de falhas; conta com outras ferramentas para monitoramento de rede, segurança, entre outros
Desvantagem	Trata-se de uma versão trial

Fonte: Elaborado pelo autor.

Sistema para resgate de *spot* inesperados usando instâncias de *spot* heterogêneos e excesso de provisão

O sistema possui auto escala confiável para aplicações *web* usando instâncias de pontos heterogêneos junto com instâncias sob demanda. Essa abordagem não só reduz o custo financeiro do uso de recursos da nuvem, mas também garante alta disponibilidade e o baixo tempo de resposta, mesmo quando alguns tipos de VMs manuais são encerradas inesperadamente ou consecutivamente dentro de um curto período de tempo (QU; CALHEIROS; BUYYA, 2016).

O sistema tolera falhas utilizando resgates de locais inesperados que usa instâncias de pontos heterogêneos e excesso de provisão. A Tabela 37 traz de forma resumida as principais características do sistema.

Tabela 37 – Resumo do sistema.

Técnicas utilizadas	*Checkpointing*, migração de trabalho, replicação e resgate do fluxo de trabalho
Gestor de nuvem compatível	Amazon EC2
Modelo de programação	Java
Problema Solucionado	Redução de custo financeiro de recursos em nuvem e garantia da alta disponibilidade
Tipo de Falha tratada	Comunicação, Processo
Vantagem	Lida com resgates de pontos inesperados usando instâncias de pontos heterogêneos e excesso de provisão
Desvantagem	Em algumas situações, apesar de ser algo raro, a heterogeneidade em grupos locais pode causar violação da semântica tolerante a falhas

Fonte: Elaborado pelo autor.

Symbiotic Virtualization (SymVirt)

O SymVirt permite que uma VM coopere com uma camada de transferência de mensagens no sistema operacional convidado. Ele funciona como um mediador que possibilita *hot migration* utilizando os mecanismos de tolerância a falhas SymCR e o SymPFT, que podem ou não trabalhar em conjunto conforme a necessidade (TAKANO et al., 2012).

O SymCR é utilizado para controlar/reiniciar VMs, e baseia-se nas técnicas de replicação e *checkpoint* para se recuperar de falhas. Já o SymPTF, baseia-se em técnicas proativas e permite migrar uma VM de um nó "insalubre" antes que o nó falhe. O sistema funciona em três etapas:

- **Hot-del:** um agente SymVirt remove um VMM-bypass do dispositivo de E/S da VM;
- **Checkpoint/Repetir ou Migração**; e
- **Hot-add:** atribui-se um VMM-bypass do dispositivo de E/S para VM.

A Tabela 38 traz de forma resumida as principais características do sistema.

Tabela 38 – Resumo do sistema SymVirt.

Técnicas utilizadas	Replicação, Checkpoint e Repetir
Gestor de nuvem compatível	Universal
Modelo de programação	Fortran
Problema Solucionado	Migração e comunicação entre VMM e o S.O convidado
Tipo de Falha tratada	Comunicação e Virtualização
Vantagem	Permite *hot migration* de VM; fácil implementação; API simples; suporte para *hot-add*
Desvantagem	Sobrecarga das saídas das VMs; Aumento da latência da comunicação

Fonte: Elaborado pelo autor.

4.3.2 Uso de Migração preemptiva

Graças a migração preemptiva é possível efetuar a migração antecipatória de uma tarefa, o que permite a possibilidade de tratamento de prevenção a falhas. Nessa categoria pode ser destacada a abordagem Protótipo para tratamento automático de anomalias (GULENKO et al., 2016), apresentada na seção anterior e, as abordagens Mantis (KRETSIS et al., 2015), FT-FW (AYUSO; GASCA; LEFEVRE, 2012), Sistema de controle de acesso avançado (CHUNG et al., 2018), Protótipo para Extração sistemática dos estados de rede (XU et al., 2015), e pFTree-Ext e pFTree-Wt (ZAHID et al., 2017), apresentadas a seguir.

Mantis

Apesar do sistema ser compatível com várias nuvens, os autores Kretsis et al. (2015) fizeram testes apenas no OpenStack. O Sistema em questão baseia-se na definição de uma política padrão que limita a quantidade de tarefas executadas simultaneamente em relação ao número de núcleos da máquina de hospedagem.

Sua aplicação em nuvem é baseada nos seguintes componentes: filas de solicitação e respostas, provedor de informações e despachador. O mecanismo de aplicativos em nuvem recebe os pedidos da camada de acesso e armazena-os na fila de solicitação e localmente em um arquivo de disco, fornecendo assim um mecanismo simples de tolerância a falhas. A Tabela 39 traz de forma resumida as principais características do sistema.

Tabela 39 – Resumo do sistema Mantis

Técnicas utilizadas	Migração preemptiva
Gestor de nuvem compatível	Universal
Modelo de programação	Ruby e Python
Problema Solucionado	Balanceamento de carga em ambientes com rede ópticas
Tipo de Falha tratada	Comunicação
Vantagem	Interface amigável
Desvantagem	Por tratar-se de rede ópticas, pode haver problemas quanto a definições de extensões e parâmetros (camada física)

Fonte: Elaborado pelo autor.

Fault-Tolerant stateful Firewall (FT-FW)

Ayuso, Gasca e Lefevre (2012) projetaram um sistema de *firewall* com suporte à tolerância falhas. As principais características da solução que os autores fornecem são:

- **Simplicidade:** reutiliza e amplia as soluções de alta disponibilidade existentes no *software*. O lado do cliente não requer nenhuma modificação. Assim, esta é uma solução transparente para o cliente;
- **Desempenho:** a solução garante um atraso negligenciável nas respostas dos clientes e recuperação rápida de falhas.
- **Baixo custo:** é adequada para equipamentos COTS e não requer extensões de hardware; e
- **Configurações de compartilhamento de carga de trabalho multi-primárias:** a arquitetura proposta suporta configurações avançadas onde vários *firewalls* compartilham a carga de trabalho para melhorar a escalabilidade.

O sistema utiliza um gerenciador de detecção de falhas para monitorar as falhas do conjunto de firewalls com informações de estado. Esse gerenciador é executado em cada firewall com preservação do estado para verificar a própria integridade e outros nós do *firewall* com preservação do estado que pertencem ao cluster. O firewall com monitoração do estado pode estar em dois modos de funcionamento:

- **Primário:** significa que ele está ativo e executando a filtragem com informações de estado; e
- **Backup:** o firewall aguarda a falha de uma das primárias para substituí-los. Um firewall com monitoração do estado pode estar nos modos de trabalho primário e de backup ao mesmo tempo, o que significa que está implantando a filtragem com informações do estado, mas também está agindo como backup para outro firewall com informações do estado.

Se um firewall com monitoração do estado tiver uma falha, o gerenciador de detecção de falha selecionará qual firewall com backup de estado se tornará primário.

A Tabela 40 traz de forma resumida as principais características do sistema.

Tabela 40 – Resumo do sistema FT-FW.

Técnicas utilizadas	Migração preemptiva
Gestor de nuvem compatível	Universal
Modelo de programação	Não informado pelos autores
Problema Solucionado	Tolerância a falhas para firewalls
Tipo de Falha tratada	Comunicação
Vantagem	Contém um gerenciador de detecção de falhas
Desvantagem	Todos os fluxos são recuperados com sucesso, porém, reduz drasticamente o desempenho da rede

Fonte: Elaborado pelo autor.

Sistema de controle de acesso avançado

Chung et al. (2018) propõe um novo sistema de rede para fornecer controle avançado de acesso usando SDN alavancado e autorização baseada em *token*. A arquitetura proposta garante que as reservas de fluxo de rede sejam realmente utilizadas pelo responsável da reserva seguindo as seguintes fases:

- O usuário solicita uma reserva por meio de um orquestrador;
- O orquestrador verifica se o controlador SDN possui a largura de banda solicitada entre os dois pontos finais está disponível durante o período de tempo especificado;
- No início da transferência é feita a inclusão do *token* de reserva; e
- Após o endereço IP e a porta serem conhecidos, o usuário envia uma solicitação com o *token*, o endereço IP de destino e a porta de destino para o remetente de dados.

Para garantir o processo citado anteriormente, é proposto um protocolo de confirmação de 2 fases (2PC) na comunicação entre o orquestrador e os controladores de site. O 2PC cuida da tolerância a falhas no momento da reserva, garantindo que os recursos sejam liberados se pelo menos um dos domínios envolvidos não conseguir provisionar a reserva. A Tabela 41 traz de forma resumida as principais características do sistema.

Tabela 41 – Resumo do sistema de controle de acesso avançado.

Técnicas utilizadas	Migração preemptiva
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Provisionamento de reserva antecipada
Tipo de Falha tratada	Comunicação
Vantagem	Redução no tempo de provisionamento; garante provisionamento de forma antecipada e flexível
Desvantagem	Sobrecarga por conta da latência do provisionamento

Fonte: Elaborado pelo autor.

Protótipo para Extração sistemática dos estados de rede

Apesar de se tratar de um protótipo mostrou, por experimentos e emulação, que é capaz de detectar uma ampla variedade de inconsistências de configuração de rede (XU et al., 2015). O sistema se subdivide em três componentes:

- **Subsistema de coleta de dados:** é composto por um agente de coleta residente em cada *host* e no servidor de verificação, que é responsável por receber os dados coletados dos agentes e dados de configurações do controlador;
- **Subsistema de processamento de dados:** é composto por um analisador de estado que absorve os dados coletados e insere nas três camadas de representação do estado da rede;
- **Subsistema de verificação de relatórios:** efetua a verificação de estado nas camadas L2, L3 e L4, obtidas com o uso do controlador e dos *hosts* finais, gerando alertas de inconsistências caso existam.

A Tabela 42 traz de forma resumida as principais características do sistema.

Tabela 42 – Resumo do Protótipo

Técnicas utilizadas	Migração preemptiva
Gestor de nuvem compatível	OpenStack
Modelo de programação	Python
Problema Solucionado	Detecção de inconsistências em configurações de rede
Tipo de Falha tratada	Comunicação
Vantagem	Redução de tempo de análise em L4; Grande escopo de detecção de erros
Desvantagem	Voltado para sistemas de baixa carga computacional

Fonte: Elaborado pelo autor.

pFTree-Ext e pFTree-Wt

Os autores apresentam duas extensões significativas do algoritmo de roteamento em fat-tree, denominadas pFTree-Ext e pFTree-Wt, para sistemas de HPC baseados em *InfiniBand*.

As extensões foram criadas para incorporar políticas de partição definidas pelo provedor que governa o modo de como os nós em diferentes partições distribuem recursos de rede entre si. Os autores apresentam uma versão ponderada do algoritmo, que, além das partições, também leva em conta as características do tráfego do nó para equilibrar a carga nos *links* de rede mais uniformemente. Uma avaliação abrangente que compreende experimentos e simulações no mundo real confirma a correção e a viabilidade das extensões propostas. Quando a rede não possui recursos suficientes para isolar partições unicamente no nível de ligação física, o algoritmo de roteamento pFTree-Ext, usa *Virtual Lanes* para reduzir interferências. No entanto, diferentes partições podem ter diferentes necessidades de isolamento, dependendo dos requisitos correspondentes de SLAs ou QoS (ZAHID et al., 2017).

Enquanto o pFTree-Wt é baseado na noção de pesos associados a cada nó de computação. Esses pesos são usados para conhecer as características de tráfego conhecidas ou aprendidas no cálculo de rotas. Independentemente do particionamento, o peso de um nó é dado ao calcular tabelas de roteamento e reflete o grau de prioridade dos fluxos para um nó (ZAHID et al., 2017).

A Tabela 43 traz de forma resumida as principais características do sistema.

Tabela 43 – Resumo dos sistemas pFTree-Ext e pFTree-Wt.

Técnicas utilizadas	Migração Preemptiva e Migração de Trabalho
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Isolamento pelo pFTree-Ext, distribuição de carga e distribuição de rotas da rede pelo pFTree-Wt
Tipo de Falha tratada	Comunicação e Processo
Vantagem	O pFTree-Wt satisfaz o balanceamento de carga ponderado nos links, mantendo as partições isoladas
Desvantagem	Limita o número de partições possíveis na nuvem

Fonte: Elaborado pelo autor.

4.3.3 Uso de Autocura

A autocura possibilita a restauração do ambiente de forma que seja necessário muito pouca ou nenhuma intervenção humana. Nessa categoria pode ser destacada a abordagem Protótipo para tratamento automático de anomalias (GULENKO et al., 2016), apresentada na seção anterior e, a abordagem Auto healing service framework (LI et al., 2017), descrita a seguir.

Auto healing service framework

O *framework* proposto por Li et al. (2017), é composto por uma estrutura de serviço melhorada com a capacidade de auto cura, adicionando um grupo de *plugins* personalizáveis e flexíveis na função de orquestração comum da Cloud. A estrutura de autocura proposta é adequada para qualquer aplicativo em nuvem, desde aplicações em nuvens nativas até aplicações de rede de virtualização de funções.

O funcionamento da auto cura envolve três perspectivas, sendo elas: detecção de falhas, notificação de falhas e recuperação de falhas. Foi proposto melhorias correspondentes ao adicionar três tipos de componentes cruciais na estrutura inicial do orquestrador (LI et al., 2017):

- **Serviço de gerenciamento de saúde:** é usado para aceitar e colocar um aplicativo em uma lista de manutenção para gerenciar seu status, desde o registro até o não registro. Este serviço detectará a falha somente após as inscrições serem registradas;
- **Serviço de monitoração de *plugins*:** usa o *software* de monitoramento de nuvem de terceiros para detectar falhas entre outros problemas que podem surgir durante o tempo de execução de um aplicativo; e
- **Serviço de recuperação:** é usado para mostrar o comportamento de cura direcionado. Os motivos por que não integrar este serviço de recuperação com gerenciamento de saúde é devido a diversidade de tomadores de ação.

A Tabela 44 traz de forma resumida as principais características do sistema.

Tabela 44 – Resumo do Auto Healing Service Framework.

Técnicas utilizadas	Autocura
Gestor de nuvem compatível	OpenStack
Modelo de programação	Python
Problema Solucionado	Uma estrutura de autocura adequada para qualquer aplicativo em nuvem com plugins personalizáveis
Tipo de Falha tratada	Processo e Comunicação
Vantagem	Alarme é disparado com base na capacidade de uso dos discos dos *hosts*
Desvantagem	Incompatibilidade com outras plataformas de nuvem

Fonte: Elaborado pelo autor.

4.3.4 Uso de Replicação

Replicação é a técnica mais simples dentre as utilizadas, com ela é possível replicar as etapas de comunicação de um determinado processo, como por exemplo, considerando o protocolo TCP/IP, é possível fazer replicas na integra ou por camadas. Nessa categoria podem

ser destacadas as abordagens: Protótipo para tratamento automático de anomalias (GULENKO et al., 2016), SwiftER (KULKARNI; BHOSALE, 2016; DATTA; CHO, 2016) e, Sistema de armazenamento privado multicamadas (GONZALEZ et al., 2013), citadas na seção anterior e, as abordagens Mayflies (AHMED; BHARGAVA, 2016), TOPSIS (IBN-KHEDHER; ABD-ELRAHMAN, 2017), Group-U (PEI et al., 2017), DORADO (NETTO et al., 2017), COSCAnet-FT (KÄCHELE; HAUCK, 2015), CIT (KIRTHICA; SRIDHAR, 2017), OpenADN (PAUL et al., 2014), e SprintNet (WANG et al., 2015), descritas a seguir.

Mayflies

O Mayflies é um sistema baseado em *Moving Target Defense* (MTD) genérica bio-inspirada para sistemas distribuídos em plataformas de nuvem virtualizadas, garantindo defesa contra ataques (AHMED; BHARGAVA, 2016). O sistema opera em três camadas lógicas diferentes da estrutura da nuvem:

- **Camada da aplicação:** calculo;
- **Camada do *hypervisor* (camada HostOS):** *Virtual Machine Introspection* (VMI); e
- **Camada de rede:** *Software Defined Networking* - SDN.

Os dois procedimentos principais do algoritmo são *introspect* e *reincarnate* para executar o Monitoramento do Nó Proativo e a Reencarnação do Nó, respectivamente (AHMED; BHARGAVA, 2016). A Tabela 45 traz de forma resumida as principais características do sistema.

Tabela 45 – Resumo do sistema Mayflies

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	Libvmi (Python e C)
Problema Solucionado	Controla as janelas de ataque de exposição dos nós por meio de monitoramento proativo e atualização da VM, referido como reencarnação de nó, em diversas plataformas em intervalos de tempo
Tipo de Falha tratada	Comunicação e Virtualização
Vantagem	Atualização da VM em menos de um minuto
Desvantagem	Limita-se a ameaças específicas

Fonte: Elaborado pelo autor.

TOPSIS

Ibn-Khedher e Abd-Elrahman (2017) propõem uma técnica de decisão multi-critérios denominada TOPSIS. A técnica é introduzida para incluir o contexto da migração *Virtual Content Delivery Network* (vCDN). O TOPSIS é usado para selecionar a camada ideal para a implantação do vCDN entre três alternativas:

- **Points of Presence (PoPs)**;
- **Home Gatway/STBs**; e
- **Dispositivos de usuário**.

A seleção da camada ideal pode reduzir o tempo de execução dos algoritmos de otimização do vCDN e sua periodicidade de execução.

O algoritmo é baseado em duas etapas principais:

- **Algoritmo TOPSIS:** os parâmetros da rede são identificados para escolher cuidadosamente os critérios de decisão e as principais alternativas; e
- **Seleção de camada e processo de migração:** o administrador de rede seleciona a camada adequada de implantação. Os nós vCDN migram dinamicamente para os servidores NFV ideais de acordo com a saída anterior da técnica TOPSIS. Isso reduz o número de parâmetros e restrições usados pelo algoritmo de otimização. Portanto, o tempo de execução e a complexidade do algoritmo podem ser reduzidos.

A Tabela 46 traz de forma resumida as principais características do sistema.

Tabela 46 – Resumo do sistema TOPSIS.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Seleção da camada ideal para a implantação do vCDN
Tipo de Falha tratada	Comunicação
Vantagem	TOPSIS seleciona a melhor camada (vCDNTTP, vCDNTTH, vCDNTTU ou vCDNATO) em que pode executar o algoritmo de otimização para a migração de nós vCDN.
Desvantagem	Com a leitura do artigo não foi possível identificar pontos que realmente fossem considerados como desvantagens

Group-U

O Group-U possui um esquema de atualização agrupado em pipeline com base em códigos de apagamento que compreende quatro recursos principais:

- Agrupa os nós de dados para completar a transmissão dos mesmos e ajusta dinamicamente o tamanho do grupo de acordo com a carga de trabalho de atualização;
- Cria a pipeline para a transmissão de dados e distribui a computação de atualização para todos os nós participantes para melhorar a eficiência da atualização;
- Adota a atualização *in-time* para os nós de dados e *lazy-update* para nós de pares para reduzir ainda mais a sobrecarga de atualização; e
- Ajusta a ocasião que desencadeia a atualização para ser compatível com a falha do nó.

A Tabela 47 traz de forma resumida as principais características do sistema.

Tabela 47 – Resumo do sistema Group-U.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenStack
Modelo de programação	Não informado pelos autores
Problema Solucionado	Redução de carga durante o processo de atualização de dados dos nós
Tipo de Falha tratada	Processo e Comunicação
Vantagem	Suporta uma única ou várias atualizações
Desvantagem	Grande consumo de tráfego de rede devido à sua abordagem líder em grupo

Fonte: Elaborado pelo autor.

Dering OveR shAreD memOry (DORADO)

Foi desenvolvido um protocolo que utiliza a memória compartilhada disponível em Kubernetes. O protocolo criado pelos autores se chama DORADO (ou Dering OveR shAreD memOry) para fazer replicação de estado em contêineres (NETTO et al., 2017).

O sistema é capaz de replicar, criar ou excluir nós, mantendo um número predefinido de réplicas em execução. O recurso de balanceamento de carga melhora a disponibilidade. A Tabela 48 traz de forma resumida as principais características do sistema.

Tabela 48 – Resumo do sistema DORADO.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Não informado pelos autores
Modelo de programação	Não informado pelos autores
Problema Solucionado	Replicação automática de estados da máquina
Tipo de Falha tratada	Processo e Comunicação
Vantagem	Pode replicar nós, criar ou excluir, mantendo um número predefinido de réplicas em execução
Desvantagem	O protocolo do sistema usa um coletor de lixo na memória compartilhada para evitar o crescimento ilimitado dos dados e um mecanismo de controle para limitar a quantidade de mensagens necessárias para restaurar o estado. No entanto, o protótipo atual não contém esses dois mecanismos.

Fonte: Elaborado pelo autor.

COSCAnet-FT

Kächele e Hauck (2015) apresentam um roteador (COSCAnet-FT) de nível de transporte que funciona como um serviço na rede que transmite de forma transparente o tráfego TCP para instâncias de serviços replicadas ativamente.

No COSCAnet-FT, cada aplicação obtém um endereço IP virtual individual. O roteador de camada de rede e transporte encaminha datagramas direcionados para um endereço IP virtual para o nó correto. O mapeamento é mantido no sistema para que as aplicações sejam independentes do endereço IP do nó e, portanto, possam migrar. Os nós e o componente da camada de rede cooperam para que os datagramas de saída carreguem o endereço do remetente correto. Este mecanismo também é usado para implementar escalabilidade e elasticidade, permitindo que múltiplos nós hospedem suas instâncias do mesmo aplicativo, cada nó com o mesmo endereço IP virtual. Portanto, o roteador funciona como um balanceador de carga de camada de transporte que encaminha corretamente as conexões TCP (KÄCHELE; HAUCK, 2015).

Utiliza-se um roteador à nível de transporte como um serviço na rede que, de maneira transparente, faz multicast do tráfego TCP para instâncias de serviço ativamente replicadas. As réplicas executam serviços independentes e não armazenam o estado em um meio compartilhado.

O roteador também amplia a capacidade de roteamento baseado em *Network Address Translation* (NAT), tendo um mapa para cada réplica. O tráfego de rede, ou seja, os datagramas de IP, para um serviço específico, são forçados em todos os seus processos de réplica pelo roteador. Essa replicação é transparente para os protocolos de transporte.

A Tabela 49 traz de forma resumida as principais características do sistema.

Tabela 49 – Resumo do sistema COSCAet-FT.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Universal
Modelo de programação	Não informado pelos autores
Problema Solucionado	Balanceamento de carga na camada de transporte para encaminhamento das conexões TCP corretamente
Tipo de Falha tratada	Comunicação
Vantagem	Suporta um serviço de valor agregado, pois aplicativos OSGi comuns podem ser reproduzidos de forma transparente sob demanda, oferecendo alta disponibilidade
Desvantagem	Suporte apenas para TCP

Fonte: Elaborado pelo autor.

Cloud Inter-operation Toolkit (CIT)

Proposto por Kirthica e Sridhar (2017), é voltado para ambientes interoperacional multi-nuvem. Quando ocorre o evento de uma nuvem desligando para manutenção ou qualquer outro motivo, este evento não é conhecido por outras nuvens na rede, neste caso, se tal nuvem tiver emprestando recursos para usuários de outra nuvem, esses recursos não estarão mais disponíveis, causando-lhe inconvenientes. O CIT propõe um sistema de empréstimo mais confiável, de forma que evite tais problemas. A Tabela 50 traz de forma resumida as principais características do sistema.

Tabela 50 – Resumo do sistema Cloud Inter-operation Toolkit.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Universal
Modelo de programação	Não descrita pelo autor
Problema Solucionado	Empréstimo de VM
Tipo de Falha tratada	Comunicação, Processo e Virtualização
Vantagem	Economia de recursos; maior confiabilidade em provisionamento de VMs
Desvantagem	Compatível apenas com nuvens externas

Fonte: Elaborado pelo autor.

OpenADN

O OpenADN é destinado à resolução de entrega de aplicativos em um ambiente multi-nuvem. O sistema permite que os fornecedores de serviços de aplicativos possam alavancar os ambientes multi-nuvem para a implantação e entrega de seus aplicativos és de uma infraestrutura de rede compartilhada, centrada em serviços e de ampla área fornecida por provedores de terceiros, incluindo provedores de serviços de Internet, provedores de serviços em nuvem e

redes de entrega de conteúdo. O sistema fornece um novo acesso a aplicativos que possui conhecimento dos requisitos de implantação centrados no serviço, incluindo particionamento de serviços, replicação de serviços, encadernação de serviços e composição de serviços (PAUL et al., 2014). A Tabela 51 traz de forma resumida as principais características do sistema.

Tabela 51 – Resumo do sistema OpenADN.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Não informado pelo autor
Modelo de programação	Não informado
Problema Solucionado	Disponibilidade
Tipo de Falha tratada	Comunicação
Vantagem	Sobreposição IP (cada entidade OpenADN: *host* final, *proxy*, *middlebox* e servidor de aplicativos recebe um identificador globalmente exclusivo e fixo que é separado do seu localizador
Desvantagem	Compatível apenas com comunicação UDP

Fonte: Elaborado pelo autor.

SprintNet

SprintNet é uma arquitetura de rede centrada no servidor para *data centers*, que atinge um alto desempenho na capacidade da rede, tolerância a falhas e latência. Por exemplo, em um dos experimentos o tamanho do *buffer* da unidade de encaminhamento é definido como 10 MTU, equivalente a 15000 bytes por padrão, verificando uma redução de 13% a 20% da latência da rede quando se aplicam unidades de encaminhamento (WANG et al., 2015). A tolerância a falhas depende principalmente da arquitetura da rede, isto é, a forma das conexões físicas redundantes e dos esquemas de roteamento tolerantes a falhas. A Tabela 52 traz de forma resumida as principais características do sistema.

Tabela 52 – Resumo do sistema SpritNet.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	OpenNebula
Modelo de programação	Não informado
Problema Solucionado	Redução de 13% - 20% na latência da rede
Tipo de Falha tratada	Comunicação
Vantagem	Baixa perda de pacotes (0,02%)
Desvantagem	Arquitetura de rede escalável de baixo diâmetro

Fonte: Elaborado pelo autor.

4.4 Falhas relacionadas a processos

Nesta seção serão descritas as propostas para tolerar falhas relacionadas a processos. As propostas aqui listadas tratam essas falhas por *checkpointing*, autocura, migração preemptiva, repetir, migração de trabalho, ou replicação. Algumas dessas propostas foram apresentadas em seções anteriores e serão aqui apenas nominados.

4.4.1 Uso de Replicação

É bastante utilizada por ser simples. No contexto de processos é possível efetuar a simples replicação de um processo em *n* VMs de forma transparente para o usuário. Nessa categoria pode-se citar as seguintes abordagens: Middleware de nuvem para garantir desempenho e alta disponibilidade (AN et al., 2014), mOSAIC (PETCU et al., 2013), CIT (KIRTHICA; SRIDHAR, 2017), Group-U (PEI et al., 2017), DORADO (NETTO et al., 2017), Sistema para resgate de *spot* inesperados (QU; CALHEIROS; BUYYA, 2016), já apresentadas e, FASA (WAHLER et al., 2015), FTM (VILLAMAYOR; REXACHS; LUQUE, 2017), FT-FC (GARRAGHAN; TOWNEND; XU, 2011), Promethee Scheduler (MOCA et al., 2016) e, Proposta de Arquitetura de *cluster* virtual tolerante a falhas (GÓMEZ et al., 2014), apresentadas a seguir.

Future Automation System Architecture (FASA)

O FASA trabalha com estruturas de execução em tempo real escalonáveis, flexíveis e independentes de plataforma, que também fornecem recursos avançados, como tolerância a falhas baseada em *software* e alto grau de isolamento e segurança (WAHLER et al., 2015).

A tolerância a falhas é implementada com uma combinação de mecanismos de redundância e detecção de erros, ou seja, replicação de aplicações críticas ou partes do sistema. Com a redundância, se uma entidade falhar, sua contraparte assumirá suas tarefas. A Tabela 53 traz de forma resumida as principais características do sistema.

Tabela 53 – Resumo do sistema FASA.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Universal
Modelo de programação	C++ e Java
Problema Solucionado	Problemas de sistemas paralelos de automação
Tipo de Falha tratada	Processo
Vantagem	Suporta a execução de aplicativos de controle em um sistema com um número arbitrário de *hosts* cujos relógios são sincronizados
Desvantagem	Devido a baixos tempo de ciclos pode haver conflitos com alguns dos recursos avançados do FASA, como a execução distribuída

Fonte: Elaborado pelo autor.

Fault Tolerance Manager (FTM)

Villamayor, Rexachs e Luque (2017) apresentam o FTM (Fault Tolerance Manager) para arquivos coordenados de *checkpointing* que fornece aos usuários recuperação automática de falhas ao perder nós de computação. Ele tira proveito do armazenamento local do nó para salvar pontos de verificação e distribuir cópias em todos os nós de computação, evitando o gargalo.

O FTM replica arquivos de pontos de verificação locais para um nó vizinho lógico. O vizinho é selecionado levando em conta que os dois vizinhos não podem compartilhar um dispositivo provável de falha comum, como: roteadores de rede, *switches* ou fonte de alimentação. A configuração é feita para evitar a perda de dois nós vizinhos, melhorando as possibilidades de reinicialização em caso de falhas. O replicador está dentro do gerenciador do FTM e executa a replicação. Este replicador é integrado como um novo componente com o controlador protetor RADICs, que é anexado a cada nó do ambiente de execução. A Tabela 54 traz de forma resumida as principais características do sistema.

Tabela 54 – Resumo do sistema FTM.

Técnicas utilizadas	*Checkpointing*, repetir, replicação e autocura
Gestor de nuvem compatível	Amazon EC2
Modelo de programação	Não informado pelos autores
Problema Solucionado	Recuperação automática em caso de falhas em nós
Tipo de Falha tratada	Processo
Vantagem	Permite a implementação de tolerância a falhas em aplicações sem esforço significativo do usuário
Desvantagem	Falhas simultâneas do nó vizinho não são suportadas

Fonte: Elaborado pelo autor.

Fault-tolerance in federated Clouds (FT-FC)

O FT-FC possibilita a criação rápida de sistemas tolerantes a falhas bizantinas baseados em diversidade e aplicá-los a nuvens federadas. A tolerância a falhas bizantina funciona com base na suposição de que existe uma forte independência de falha entre os nós dentro do sistema. O FT-FC permite a criação de diferentes formas de algoritmos redundantes e tolerantes a falhas. O sistema inclui vários recursos para facilitar e incentivar o uso da tolerância a falhas bizantina em aplicativos federados da nuvem. Esses recursos são(GARRAGHAN; TOWNEND; XU, 2011):

- Uma ferramenta de agendamento automático de tarefas que permite que as tarefas de aplicativos do Cloud sejam automaticamente enviadas para várias nuvens heterogêneas, executando hipervisors Xen ou KVM;
- Um sistema de mensagens baseado no protocolo SSH (ou seja, ssh e scp) que permite que as comunicações entre uma nuvem e o FT-FC sejam executadas de forma segura; e

- Um sistema de adjudicação tolerante a falhas que pode enviar e receber comunicações de várias nuvens, a fim de alcançar um consenso sobre os resultados retornados de nuvens que poderiam falhar.

A Tabela 55 traz de forma resumida as principais características do sistema.

Tabela 55 – Resumo do sistema FT-FC.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Não informado pelos autores
Modelo de programação	Não informado pelos autores
Problema Solucionado	Tolerar falhas bizantinas
Tipo de Falha tratada	Processo
Vantagem	O FT-FC permite a criação de muitas formas diferentes de algoritmos redundantes tolerantes a falhas; trata falhas bizantinas
Desvantagem	A configuração de falhas e propagação de falhas nos experimentos é relativamente simplista, com falhas injetadas na camada de aplicação de nuvem, bem como uma taxa fixa de propagação

Fonte: Elaborado pelo autor.

Promethee Scheduler

O sistema em questão é um planejador tolerante a falhas e confiável, que permite executar aplicativos Bag-of-Tasks em ambiente de computação distribuída elástica e híbrida, seguindo as estratégias de programação definidas pelo usuário.

A abordagem denominada Promethee Scheduler, combina um agendador baseado em *pull* com o algoritmo de decisão multicritério *Promethee*. Como o agendamento multicritério leva à multiplicação das possíveis estratégias de escalonamento, os autores propõem uma metodologia que permite encontrar as estratégias de escalonamento ótimas, considerando um conjunto de requisitos de aplicação. Em geral, o agendador é inspirado no comportamento do XtremWeb, utilizando componentes funcionais básicos, como os mecanismos de replicação baseada em *pull* e *task* (MOCA et al., 2016). A Tabela 56 traz de forma resumida as principais características do sistema.

Tabela 56 – Resumo do sistema Promethee Scheduler.

Técnicas utilizadas	Replicação
Gestor de nuvem compatível	Amazon EC2
Modelo de programação	Não informado pelos autores
Problema Solucionado	Execução de aplicativos Bag-of-Tasks em DCI
Tipo de Falha tratada	Processo
Vantagem	Maximiza a satisfação do usuário expressa de acordo com três critérios distintos: preço, tempo de conclusão esperado e confiança, enquanto maximiza o emprego útil da infraestrutura do ponto de vista do proprietário dos recursos
Desvantagem	Necessita que o tamanho da janela esteja acima de um limite (ou seja, 10%) a fim de produzir um baixo *makespan*

Fonte: Elaborado pelo autor.

Proposta de Arquitetura de *cluster* virtual tolerante a falhas

Gómez et al. (2014) apresentam uma arquitetura de *cluster* virtual tolerante a falhas que pode lida com problemas no contexto de aplicativos de *bag-of-tasks*. Testes feito pelos autores demonstram que parte do *cluster* virtual distribuído é perdida e restaurado usando mecanismos de recuperação e regras de elasticidade, sem interrupção do serviço.

Um *Elasticity Engine* foi incluído no *cluster* e executado no nó mestre. O algoritmo de elasticidade monitora o desempenho do aplicativo periodicamente e gerencia a adição ou remoção dos nós para atingir o limite de tempo desejado para um indicador chave de aplicação específico. O critério de elasticidade é definido como o número de núcleos necessários para alcançar o o limite de tempo desejado, minimizando o número de nós implantados. Quando o desempenho do *cluster* está abaixo do desempenho estimado, novos nós são implantados se o número de núcleos necessários for positivo.

O *cluster* é implantado em dois sites para ser resiliente a falhas. O mestre está em um site, a sombra em outro e os nós de computação são distribuídos entre eles. Nessa configuração, se um site falhar, a parte sobrevivente do *cluster* poderá se recuperar da situação e aplicar regras de elasticidade, ampliar-se para fornecer os resultados ao usuário final no prazo.

Tabela 57 – Resumo da proposta de arquitetura de *cluster* virtual tolerante a falhas.

Técnicas utilizadas	Replicação, repetir e migração preemptiva
Gestor de nuvem compatível	OpenNebula
Modelo de programação	Java e JavaScript
Problema Solucionado	Problemas de aplicativos de *baf of tasks*
Tipo de Falha tratada	Processo e Virtualização
Vantagem	Pode ser implantado em ambientes de nuvem de uma ou mais aplicações, com diferentes configurações;
Desvantagem	Parâmetros de configuração dependem do gestor de nuvem e de sua infraestrutura, com a necessidade de executar experimentos em cada gestor para obter informações sobre os tempos de implantação e de descarte. Essa configuração podem mudar por adição de nova infraestrutura, melhorias no *software backend* do provedor, ou alteração nas cargas

Fonte: Elaborado pelo autor.

4.4.2 Uso de *Checkpointing*

A técnica de *checkpointing* permite que o processo seja recuperado a partir de um determinado ponto, o que permite recuperações mais rápida e menos custosa. Nessa categoria podem ser destacadas as abordagens: Middleware de nuvem para garantir desempenho e alta disponibilidade de aplicativos soft em tempo real (AN et al., 2014) e ONHelp (ZHAO et al., 2014), apresentadas em seções anteriores, e Unibus (SLAWINSKA; SLAWINSKI; SUNDERAM, 2010) e CACS (CAO et al., 2015), apresentadas a seguir.

Cloud-Agnostic Checkpointing Service (CACS)

O Sistema desenvolvido por Cao et al. (2015) utiliza pacote *Distributed MultiThreaded Checkpointing* (DMTCP) a nível de processo. Este modelo foi escolhido por conta do suporte transparente para aplicações como TCP/IP e rede *InfiniBand*.

O sistema garante um verificador de mesmo nível de processo para aplicativos distribuídos executados em máquinas virtuais.Também reduz o custo da migração ao vivo, além de reduzir quase a zero a carga mínima gerada por conta do gerenciamento de *checkpointing*. A Tabela 58 traz de forma resumida as principais características do sistema.

Tabela 58 – Resumo do sistema CACS.

Técnicas utilizadas	*Checkpointing*, migração preemptiva, migração de trabalho e Autocura
Gestor de nuvem compatível	Universal
Modelo de programação	Java (Restlet)
Problema Solucionado	Migração de VMs; diminuição de custo de *checkpointg*
Tipo de Falha tratada	Processo, Comunicação e Virtualização.
Vantagem	Compatível com múltiplas nuvens heterogêneas (incluindo migração e *cloudification*); pontos de verificação distribuída
Desvantagem	Aumento de nós aumenta ligeiramente o tempo de execução (flexibilidade limitada)

Fonte: Elaborado pelo autor.

Unibus

Tem como objetivo facilitar execuções tolerantes a falhas de aplicativos MPI (Message Passing Interface) em blocos de computação em nuvem. Em geral, o sistema é focado na virtualização de acesso a recursos e no provisionamento automático e transparente de recursos que simplificam o uso de recursos heterogêneos disponíveis para os usuários (SLAWINSKA; SLAWINSKI; SUNDERAM, 2010).

Para suportar execuções tolerante a falhas de aplicativos, o sistema baseia-se na utilização da virtualização de acesso a recursos e provisionamento de recursos. Dessa maneira consegue orquestrar recursos de forma tolerante a falhas sem a necessidade de modificação do código-fonte do aplicativo. A implementação dos serviços *checkpoint* e repetir é baseada em DMTCP .

O ponto de verificação é iniciado no nó principal e procede de forma distribuída e coordenada em cada nó do *cluster*. A coordenação é possível graças ao DMTCP que intercepta os subprocessos ssh, executados pelo OpenMPI para iniciar processos MPI em nós remotos. A Tabela 59 traz de forma resumida as principais características do sistema.

Tabela 59 – Resumo sistema Unibus.

Técnicas utilizadas	*Checkpointing* e repetir
Gestor de nuvem compatível	Amazon EC2
Modelo de programação	Python
Problema Solucionado	Execução tolerante a falhas de aplicativos distribuídos e paralelos
Tipo de Falha tratada	Processo e virtualização
Vantagem	Baixo custo operacional e fácil configuração
Desvantagem	O método para tolerar falhas é basicamente o mesmo que outros sistemas já utilizam

Fonte: Elaborado pelo autor.

4.4.3 Uso de Autocura

A técnica de autocura permite a recuperação de um determinado processo de maneira automática, porém, na proposta em questão, necessita do auxílio de uma segunda técnica de tolerância a falhas.Nessa categoria pode ser destacada a abordagem Availability Knob (SHAHRAD; WENTZLAFF, 2016) descrita a seguir.

Availability Knob

Conhecido como Knob, ou AK, o sistema proposto por Shahrad e Wentzlaff (2016) oferece disponibilidade flexível definida pelo usuário, em nuvens IaaS.

O sistema rastreia falhas que ocorreram na máquina anteriormente e determina a probabilidade de ocorrência das mesmas falhas para cada nova máquina física, comparando o tempo que passou desde a última falha e compara-a com o tempo médio esperado entre falhas para esse tipo de máquina física. Para cada posicionamento de VM, o planejador calcula a probabilidade de falha, bem como o tempo de inatividade esperado por falha e considera a disponibilidade entregue/solicitada de um usuário para calcular o custo indireto esperado.

O agendador AK pode migrar periodicamente VMs super-atendidas para máquinas de baixo custo, se disponíveis. Essa capacidade, chamada *Benign VM migration* (BVM). Os provedores podem migrar deliberadamente as VMs dos usuários para reduzir suas despesas operacionais. Os candidatos a BVM são selecionados com base em quão bem eles estão sendo atendidos no período de medição atual, usando o método de atendimento de inatividade. A Tabela 60 traz de forma resumida as principais características do sistema.

Tabela 60 – Resumo do sistema Availability Knob.

Técnicas utilizadas	Migração preemptiva e autocura
Gestor de nuvem compatível	OpenStack e outras nuvens IaaS
Modelo de programação	Não foi possível identificar
Problema Solucionado	Alocação de VM
Tipo de Falha tratada	Processo e Virtualização
Vantagem	A implantação de AK leva a mais de 10% de redução de custos para provedores e melhora a satisfação do usuário
Desvantagem	A integração do AK no OpenStack requer algumas modificações no banco de dados do OpenStack Nova

Fonte: Elaborado pelo autor.

Protótipo para Sistema de Migração de VM

Wu et al. (2014) propõem uma abordagem a tolerância a falhas é definida pelo modelo que implanta automaticamente mecanismos de tolerância a falhas seguindo um padrão de alto nível, como por exemplo, replicação e migração preemptiva. Com a ajuda deste modelo, os

mecanismos de tolerância a falhas existentes serão otimizados. A Tabela 61 traz de forma resumida as principais características do sistema.

Tabela 61 – Resumo do protótipo para sistema de migração de VM.

Técnicas utilizadas	Migração preemptiva, replicação e autocura
Gestor de nuvem compatível	CloudStack
Modelo de programação	Java
Problema Solucionado	Migração de VM
Tipo de Falha tratada	Processo e Virtualização
Vantagem	O tempo médio de resposta com um mecanismo de migração é de 0,2799s, enquanto o tempo de resposta básico é de 0,2707s, portanto esta abordagem baseada em modelo não induzirá perda significativa de desempenho
Desvantagem	Não consegue lidar com um erro de falha parada

Fonte: Elaborado pelo autor.

4.4.4 Uso de Migração de Trabalho

Nessa categoria podem ser destacadas as abordagens: FTESW (DING; YAO; HAO, 2017), Sistema de detecção e mitigação de falhas (GOKHROO; GOVIL; PILLI, 2017), Algoritmo de tolerância a falhas energética para computação de alto desempenho (EGWUTUOHA et al., 2012; MOCA et al., 2016) e, Zurmo Native (TOFFETTI et al., 2017), descritas a seguir.

Fault-Tolerant Elastic Scheduling Algorithm for Workflow (FTESW)

O FTESW é um algoritmo criado para aprimorar a agilidade do sistema e a utilização de recursos, desde que seja obtida a tolerância a falhas para agendamento de fluxo de trabalho em sistemas em nuvem.

A fim de projetar uma estratégia tolerante a falhas para cada tarefa e cumprir o prazo do fluxo de trabalho, o *deadline* integral de um fluxo de trabalho foi dividido em vários intervalos para diferentes tarefas pelo analisador *workflow* no agendador. Conduziu-se para dividir o prazo de um fluxo de trabalho em sub-prazos para todas as tarefas no fluxo de trabalho e assim segmentar o prazo integral (*deadline*) em várias janelas de tempo, onde para cada janela atribuí-se a tarefa correspondente no fluxo de trabalho (DING; YAO; HAO, 2017). A Tabela 62 traz de forma resumida as principais características do sistema.

Tabela 62 – Resumo do sistema FTESW.

Técnicas utilizadas	Migração de Trabalho
Gestor de nuvem compatível	Universal
Modelo de programação	Não informado pelos autores
Problema Solucionado	Escalonamento
Tipo de Falha tratada	Processo e virtualização
Vantagem	O mecanismo elástico de provisionamento de recursos do FTESW é capaz de impulsionar a utilização de recursos de forma eficaz
Desvantagem	Funciona apenas *offline*

Fonte: Elaborado pelo autor.

Sistema de detecção e mitigação de falhas

A novidade da abordagem reside no método de detecção da falha com base no estado de execução do trabalho. O algoritmo monitora periodicamente o progresso do trabalho em máquinas virtuais e relata o trabalho parado por falha na VM para o gerenciador tolerante a falhas. Isso reduz o desperdício de recursos e garante a entrega atempada de serviços para evitar qualquer penalidade devido à violação do acordo de nível de serviço (GOKHROO; GOVIL; PILLI, 2017).

Inicialmente, o sistema utiliza um mecanismo menos carregado que calcula a carga em cada VM (número de processos em execução em cada VM) e escolhe a VM com a carga mínima como VM de destino. O algoritmo de migração determina a VM em que o processo será migrado, ele seleciona a VM de destino com base no número de tarefas atribuídas (GOKHROO; GOVIL; PILLI, 2017). A Tabela 63 traz de forma resumida as principais características do sistema.

Tabela 63 – Resumo do sistema de detecção e mitigação de falhas.

Técnicas utilizadas	Migração Preemptiva e Migração de Trabalho
Gestor de nuvem compatível	Não informado pelos autores
Modelo de programação	Não informado pelos autores
Problema Solucionado	Detecção e mitigação de falhas com base no estado de execução do trabalho utilizando monitoramento
Tipo de Falha tratada	Processo e Virtualização
Vantagem	Garante que as sobrecargas mínimas do sistema sejam adicionadas e que o QoS afetado seja o mínimo
Desvantagem	Mesmo que haja uma ou mais falhas de VM, todos os trabalhos que foram inicialmente submetidos à VM ainda são executados com 100% de sucesso em VMs nova, mas com algum tempo de atraso

Fonte: Elaborado pelo autor.

Algoritmo de tolerância a falhas energética para computação de alto desempenho

Egwutuoha et al. (2012) apresentam tolerância a falhas eficiente de energia para HPC (computação de alta performance) em nuvem. Foi desenvolvido um algoritmo de tolerância a falha genérico para sistemas HPC na nuvem. O algoritmo usa uma abordagem de migração em nível de processo, no entanto, não depende de um nó sobressalente ou de uma computação redundante antes da previsão de uma falha. A tolerância a falhas eficiente em energia para computação de alto desempenho (HPC) na nuvem requer quatro tipos de módulos:

- Módulo de monitoramento de nó com um sensor;
- Um preditor de falha baseado em regras;
- Módulo de política de migração; e
- O módulo do controlador.

O algoritmo utiliza sensores para determinar o estado do sistema, podendo prever futuras falhas e tomar as ações necessárias para reduzir seu impacto nos aplicativos. As ações incluem o aluguel de um nó adicional do provedor de serviços, migração em nível de processo dos aplicativos e a publicação do estado de aviso no *host* principal, além de liberar o nó não íntegro e instala um FTDaemon no *host* recém-alugado. Este sistema complementa a solução *checkpointing* e repetir. A frequência de verificação dos aplicativos pode ser reduzida em até 50% com o FTDaemon. Assim, essa abordagem pode ajudar a reduzir o consumo de energia em 30% porque o *host* reserva não é provisionado. A Tabela 64 traz de forma resumida as principais características do sistema.

Tabela 64 – Resumo do sistema de tolerância a falhas energética para computação de alto desempenho.

Técnicas utilizadas	Migração de trabalho e preemptiva, *checkpointing* e autocura
Gestor de nuvem compatível	Universal
Modelo de programação	Não informado pelos autores
Problema Solucionado	Escalonamento e prevenção a falhas
Tipo de Falha tratada	Processo e virtualização
Vantagem	É capaz de impulsionar a utilização de recursos de forma eficaz
Desvantagem	Funciona apenas *offline*

Fonte: Elaborado pelo autor.

Posteriormente, Egwutuoha et al. (2013) melhoram o algoritmo de tolerância a falha, permitindo o fornecimento de tolerância a falha para HPC em nuvem em nível de hardware com

custo reduzido, enquanto executa no espaço do usuário (sob o controle dos usuários). E não depende da existência de nós sobressalentes pré-configurados.

Zurmo Native

Combinada com o posicionamento entre domínios de falha, a arquitetura proposta permite autogerenciamento hierárquico distribuído, semelhante a um organismo (ou seja, o serviço composto) que é capaz de recriar suas células para manter sua morfologia enquanto cada célula (ou seja, cada micro serviços) é um elemento de autogerenciamento. Mesmo que uma VM ou o contêiner em que a lógica de dimensionamento automático esteja em execução falhe, um novo componente poderá ser iniciado para assumir a lógica de dimensionamento automático e a base de conhecimento de onde foi deixada (TOFFETTI et al., 2017). A Tabela 65 traz de forma resumida as principais características do sistema.

Tabela 65 – Resumo do sistema Zurmo Native.

Técnicas utilizadas	Migração de trabalho e autocura
Gestor de nuvem compatível	OpenStack e AWS
Modelo de programação	Python
Problema Solucionado	Autogerenciamento escalável e resiliente
Tipo de Falha tratada	Processo e Virtualização
Vantagem	Permite a resiliência de ambas as funcionalidades sejam sem estado, de modo que, em caso de falha, elas possam ser reiniciadas em qualquer nó que esteja coletando informações de estado compartilhado por meio do sistema de gerenciamento de configuração
Desvantagem	Para de responder a requisições caso o disco da máquina esteja cheio

Fonte: Elaborado pelo autor.

4.5 Falhas relacionadas a virtualização

Nesta seção serão descritos os sistemas para tolerar falhas relacionadas a virtualização. Os sistemas aqui listados tratam essas falhas por *checkpointing*, autocura, migração preemptiva ou replicação. Alguns desses sistemas já foram apresentados e serão aqui apenas nominados.

4.5.1 Uso de *Checkpointing*

O uso de *checkpointing* para tratar falhas de virtualização se baseia no fato de que falhas desse tipo são tratadas com a restauração da máquina virtual antes da continuidade da execução. Assim, manter imagens de *checkpointing* permite a recuperação do estado do sistema sem maiores dificuldades. Nessa categoria podem ser destacadas as abordagens: ONHelp (ZHAO et al., 2014), SymVirt (TAKANO et al., 2012) , CACS (CAO et al., 2015), Middleware de nuvem

para garantir desempenho e alta disponibilidade de aplicativos soft em tempo real (AN et al., 2014), e Unibus (SLAWINSKA; SLAWINSKI; SUNDERAM, 2010), apresentadas nas seções anteriores e, em particular, a Proactive Fault Tolerance Mechanism for Data Center Network (PFT-CCKP), apresentada por Liu et al. (2015), descrita a seguir.

Proactive Fault Tolerance Mechanism for Data Center Network (PFT-CCKP)

Com o objetivo de realocar a VM em um *host* ideal e, consequentemente recuperar o sistema, a proposta baseia-se na utilização de *checkpoint* coordenado e *full complete checkpoint*. A arquitetura da proposta inclui três módulos:

- **Preditor de VM:** o módulo adota uma versão adaptada de Modelo métrico estatístico (SMM) para prever um VM deteriorada executando em um host na nuvem;
- **Ponto de controle coordenado:** quando há uma deterioração VM em um *cluster*, o módulo automaticamente faz *checkpoint* com o *cluster* virtual para garantir global consistência; e
- **Seleção ótima de *host* alvo:** quando o módulo seleciona o *host* alvo ideal para a VM deteriorada, é para ser modelado como um problema de otimização que considera três restrições, incluindo a potência da CPU, memória capacidade e capacidade do disco.

A Tabela 66 traz de forma resumida as principais características do sistema.

Tabela 66 – Resumo do sistema PFT-CCKP.

Técnicas utilizadas	*Checkpointing*
Gestor de nuvem compatível	Não informado pelos autores
Modelo de programação	Não informado pelos autores
Problema Solucionado	Realocação de VM
Tipo de Falha tratada	Virtualização
Vantagem	Migração automática de hosts; prever VMs deterioradas
Desvantagem	Precisa de ajustes para reduzir consumo de rede e de armazenamento (sugerido como trabalho futuro pelos autores)

Fonte: Elaborado pelo autor.

4.5.2 Uso de Replicação

A replicação é uma técnica de fácil uso que aplica-se a diversas situações dentro do contexto de virtualização, trazendo como opção ser feita de forma integral ou parcial, por exemplo. Nessa categoria podem ser destacadas as abordagens: Protótipo para tratamento automático de anomalias (GULENKO et al., 2016), Middleware de nuvem para garantir desempenho e alta disponibilidade de aplicativos soft em tempo real (AN et al., 2014), Morpho (LU et al.,

2014), CIT (KIRTHICA; SRIDHAR, 2017), Mayflies (AHMED; BHARGAVA, 2016), SymVirt (TAKANO et al., 2012) (TAKANO et al., 2012), Proposta de Arquitetura de *cluster* virtual tolerante a falhas (GÓMEZ et al., 2014), Protótipo para Sistema de migração de VM (WU et al., 2014), apresentadas nas seções anteriores e, em particular, a Replicação de VM para melhorar a capacidade de sobrevivência de aplicativos de missão crítica apresentada por Zhao et al. (2015), descrita a seguir.

Replicação de VM para melhorar a capacidade de sobrevivência de aplicativos de missão crítica (Protótipo)

Este sistema visa melhorar a capacidade de sobrevivência de aplicativos de missão crítica com o uso da replicação de VM. Esta nova abordagem apresenta um protótipo que trabalha com a replicação de VM em vários níveis e que usa diferentes tipos de clones de VM para fornecer uma variedade de proteções à aplicativos de missão crítica e melhorar a capacidade de sobrevivência desses aplicativos sob falhas acidentais e ataques mal-intencionados. Nessa abordagem, os clones de VM completos são empregados para fornecer tolerância a ataques, clones de chamariz são criados para desviar ataques e clones de *honeypot* são usados para analisar ataques (ZHAO et al., 2015).

Quando uma instância com falha é detectada ela é destruída e em seguida é gerada uma nova. O sistema utiliza o método de replicação ao vivo, que primeiro suspende todas as instâncias restantes para pausar temporariamente a computação e sincronizar seu estado na memória com o armazenamento do *host*. Em seguida, replica um dos clones completos copiando seu estado de memória e estado de disco armazenados em seu *host*. Também provisiona a réplica com recursos de CPU (Unidade Central de Processamento), memória e E/S idênticos à instância de destino que está sendo replicada. Antes de retomar a nova réplica, é necessário fazer a escrituração necessária. Após, atribui-se uma nova ID à nova réplica para identificá-la no OpenStack. Visto que o OpenStack salva o arquivo de configuração de uma instância como parte de seu estado de memória ao suspender a instância, esse arquivo de configuração precisa ser extraído do estado da memória, modificado para refletir a nova identificação da instância e injetado novamente no estado da memória, antes da nova instância pode ser retomada e reconhecida pelo OpenStack. Após a conclusão do processo, é necessário alterar a configuração de rede da nova réplica para evitar conflitos com a existente. Para isso, atribui-se à nova réplica um endereço MAC (*Media Access Control*) exclusivo em sua configuração e registra-se na instância para modificar o MAC e aplicar a alteração reiniciando a rede. Após a contabilidade concluída, todas as outras instâncias de réplica são retomadas e a computação continua normalmente (ZHAO et al., 2015). A Tabela 67 traz de forma resumida as principais características do sistema.

A técnica de autocura é utilizada ao final do processo de recuperação da falha, isto é, após todo o processo de replicação ao vivo. Por tratar-se de um protótipo, o sistema possui limitações que não existem em outros sistemas como, por exemplo, só conseguir recuperar um

clone perdido iniciando uma nova instância do zero, e não a partir de instâncias já existentes.

Tabela 67 – Resumo do protótipo de replicação de VM para melhorar capacidade de sobrevivência de aplicativos de missão crítica.

Técnicas utilizadas	Autocura e replicação
Gestor de nuvem compatível	OpenStack, Amazon-EC2
Modelo de programação	Python
Problema Solucionado	Replicação de VM em vários níveis com diferentes tipos de clones
Tipo de Falha tratada	Virtualização
Vantagem	Pode desviar ataques usando clones chamariz
Desvantagem	Trata-se de um protótipo; não se pode alterar sua interface para implementar a replicação em tempo real, portanto, uma instância de clone completo perdido só pode ser recuperada iniciando uma nova instância do zero

Fonte: Elaborado pelo autor.

4.5.3 Uso de Autocura

O uso de autocura para tratar falhas relacionadas a virtualização permite que tarefas relativamente mais simples, como por exemplo, migração de VM, sejam feitas de forma automatizada sempre que necessário, com pouca ou nenhuma intervenção humana, graças a rotinas pré-definidas. Nessa categoria podem ser destacadas as abordagens: CACS (CAO et al., 2015), Protótipo para tratamento automático de anomalias (GULENKO et al., 2016), Availability Knob (SHAHRAD; WENTZLAFF, 2016), Protótipo para Sistema de Migração de VM (WU et al., 2014), Zurmo Native (TOFFETTI et al., 2017), apresentadas nas seções anteriores e, em particular, a HAStack, apresentada por Moghaddam, Gherbi e Lemieux (2016), descrita a seguir.

HAStack

O HAStack é um *plug-in* personalizado do OpenStack Heat na forma de um sistema multiagente que visa fornecer alta disponibilidade a nível de aplicativo e VM (Virtual Machine).

Os agentes de HA (High Availability) monitoram as instâncias e os nós de cálculo do OpenStack em busca de falhas e caso ocorreram falhas o sistema tentará recuperar as instâncias afetadas. Os agentes se comunicam e monitoram uns aos outros por meio da fila de mensagens do OpenStack criando novas tarefas e as adotam na fila de mensagens. O mecanismo de manuseio de tarefas garantirá que dois agentes não estejam executando a mesma tarefa ao mesmo tempo. Nenhuma tarefa pode ser perdida se houver pelo menos um agente HA ativo no sistema. Se uma tarefa (monitoramento ou recuperação de instância) não estiver verificada como "concluída" por outros agentes em um período de tempo razoável, ela será automaticamente adotada por outro agente . A Tabela 68 traz de forma resumida as principais características do sistema.

Tabela 68 – Resumo do sistema HAStack.

Técnicas utilizadas	Autocura
Gestor de nuvem compatível	OpenStack
Modelo de programação	Python
Problema Solucionado	Suporte para *failover* e outros modelos de redundância
Tipo de Falha tratada	Virtualização
Vantagem	Compatibilidade e escalabilidade total com o OpenStack
Desvantagem	A utilização em nível da instância e em nível de pilha pode causar alguns conflitos com os componentes de Operação do OpenStack

Fonte: Elaborado pelo autor.

4.5.4 Uso de Migração preemptiva

A utilização de migração preemptiva para tratar falhas relacionadas a virtualização traz como principal beneficio o custo operacional reduzido quando comparada aos processos de migração "tradicional". Nessa categoria podem ser destacadas as abordagens: Protótipo para tratamento automático de anomalias (GULENKO et al., 2016), Protótipo para Extração sistemática dos estados de rede (XU et al., 2015), ONHelp, Sistema de detecção e mitigação de falhas (GOKHROO; GOVIL; PILLI, 2017), CACS (CAO et al., 2015), Availability Knob (SHAHRAD; WENTZLAFF, 2016), Proposta de Arquitetura de *cluster* virtual tolerante a falhas (GÓMEZ et al., 2014), Protótipo para Sistema de migração de VM (WU et al., 2014), apresentadas nas seções anteriores e, em particular, o FTStack, apresentado por Kanso et al. (2017), descrita a seguir.

FTStack

Baseado na técnica de migração preemptiva, o FTStack monitora o estado de cada VEEs (*Virtual Execution Environments*) até que obtenha-se sucesso na solicitação de provisionamento/remoção de um VEE. Se ocorrer uma falha durante a criação, o FTStack aguardará até que o tempo da espera (um atributo de tempo configurável do FTStack) expire e, em seguida, exclua a instância em questão e emita outra solicitação para criar uma nova VEE com a mesma configuração da excluída. O FTStack armazena a configuração do VEE até que a nova seja criada com sucesso. A Tabela 69 traz de forma resumida as principais características do sistema.

Tabela 69 – Resumo do sistema FTStack.

Técnicas utilizadas	Migração Preemptiva
Gestor de nuvem compatível	OpenStack
Modelo de programação	Python
Problema Solucionado	Provisionamento de VEE
Tipo de Falha tratada	Virtualização
Vantagem	Diminui a latência de sobrecarga para solicitações “normais”
Desvantagem	Não suporta todos os serviços do OpenStack

Fonte: Elaborado pelo autor.

4.6 Web site para consulta guiada

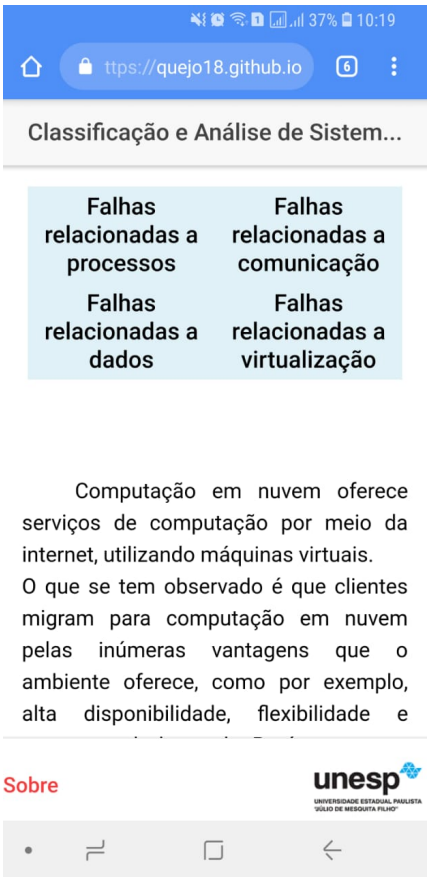
Para consultar de forma sintetizada e prática as informações relacionadas aos sistemas de tolerância a falhas obtidos com a aplicação da RSL e informações gerais sobre o projeto, criou-se um *web site* que foi hospedado no GitHub (https://quejo18.github.io./) onde será possível efetuar tais consultas de forma guiada. O *web site* foi criado utilizando o *framework* Ionic. Baseado em Angular, CSS e HTML, esse *framework* possibilita a criação de um *web site* funcional e com *layout* agradável e responsivo.

O *web site* é composto por seis páginas, contendo as seguintes informações:

- **Página Inicial:** informações gerais sobre o projeto;
- **Página “Falhas relacionadas a dados”:** tabelas resumo sobre sistemas avaliados para o tratamento de falhas relacionadas a dados;
- **Página “Falhas relacionadas a processos”:** tabelas resumo sobre sistemas avaliados para o tratamento de falhas relacionadas a processo;
- **Página “Falhas relacionadas a virtualização”:** tabelas resumo sobre sistemas avaliados para o tratamento de falhas relacionadas a virtualização;
- **Página “Falhas relacionadas a comunicação”:** tabelas resumo sobre sistemas avaliados para o tratamento de falhas relacionadas a comunicação; e
- **Página Sobre:** lista de artigos na RSL utilizados.

A Figura 14 mostra um *printscreen* de um acesso realizado com um celular modelo Samsung Galaxy S8.

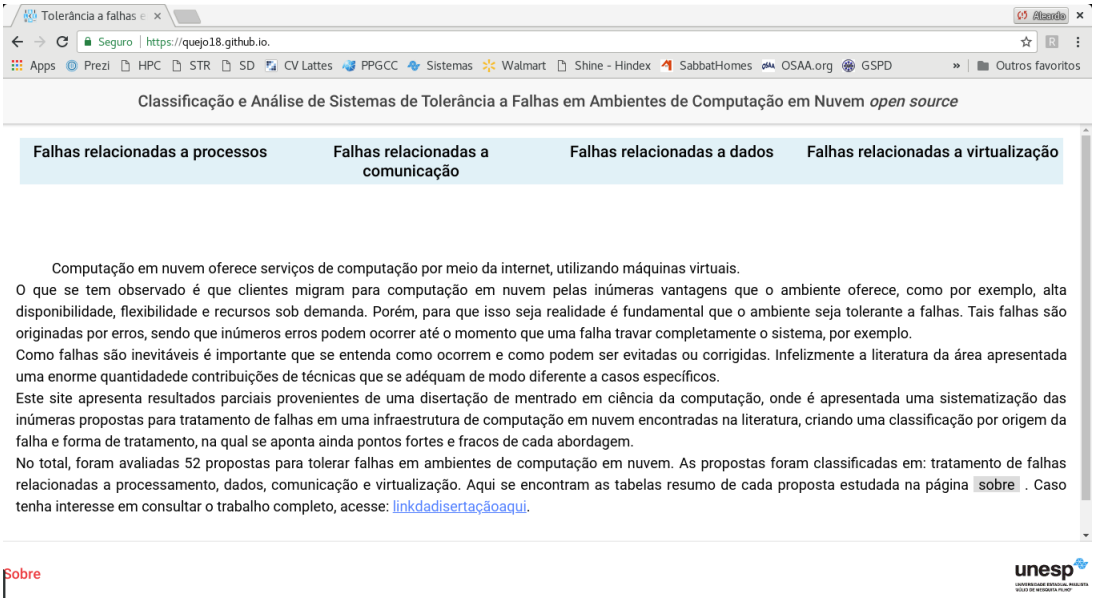
Figura 14 – Página inicial do *site* acessado usando um celular.



(Elaborada pelo autor)

Para mostrar a diferença nas formas de visualização do conteúdo, a mesma página é apresentada na Figura 15, porém agora com acesso feito por meio de um computador *desktop*.

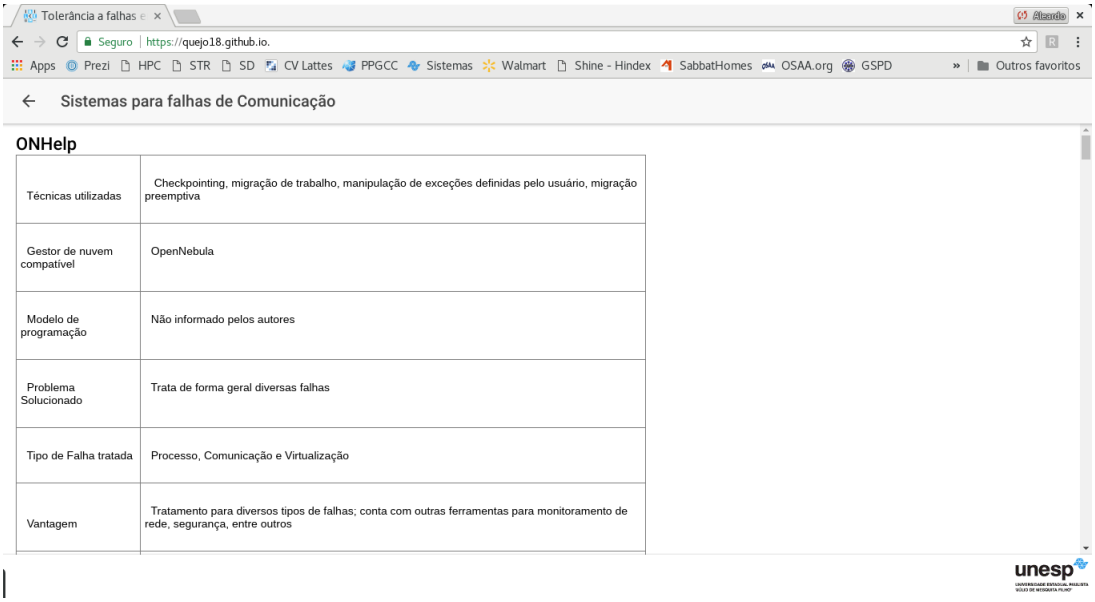
Figura 15 – Página inicial do *site* acessado usando um *desktop*.



(Elaborada pelo autor)

Ao acessar um dos *links* o usuário é levado ao conjunto de tabelas referente às soluções daquele tipo de falha, como pode ser visto na Figura 16, com um exemplo de acesso para falhas de comunicação feito a partir de um computador *desktop*. Uma nova versão desse protótipo deve permitir um acesso mais orientado com divisões para técnica de solução utilizada.

Figura 16 – Conteúdo da página "Falhas relacionadas a comunicação" do *site* acessado usando um *desktop*.



(Elaborada pelo autor)

4.7 Considerações finais

Neste capítulo foram apresentados os resultados obtidos por meio da análise dos artigos selecionados na RSL. Sempre que possível, buscou-se também identificar se a proposta era orientada para algum gestor específico. Embora essa informação nem sempre estivesse disponível, foi possível identificar que a maioria das propostas apresenta soluções compatíveis com o gestor OpenStack.

5 Conclusões e trabalhos futuros

Neste trabalho apresentou-se uma classificação de sistemas utilizados para tolerar falhas em ambientes de computação em nuvem. A classificação foi feita com base em uma revisão sistemática da literatura, que é subdividida em várias etapas, sendo que em cada etapa tem-se como objetivo extrair informações específicas dos artigos científicos. Como resultado dessa revisão foi possível gerar um conjunto de estatísticas sobre a evolução da pesquisa na área e impacto de determinadas ferramentas nesse campo, em específico no período de 2010 a 2017.

5.1 Conclusões

A classificação foi feita com o objetivo de criar um documento que possa auxiliar pessoas que tenham que decidir sobre qual solução para tratamento de falhas em um sistema de computação em nuvem é mais adequada ou eficiente para o problema enfrentado. Para cada solução descrita se incluiu uma breve introdução, que tem como propósito explanar qual o propósito da criação da solução, e uma tabela resumo que possui como intuito mostrar de maneira objetiva as principais informações dela para uma rápida consulta.

O trabalho também proporciona informações em relação ao estado da arte (de forma geral) em relação a tolerância a falhas em ambientes de computação em nuvem. Com a Seção 3.2 é possível obter informações que auxiliam em questões como: qual nuvem esta sendo mais utilizada atualmente; qual nuvem foi mais utilizada em um determinado período (2010 a 2017). Por meio das sessões 3.3 e 3.4 é possível verificar qual tipo de falha é mais comum, além de qual técnica para tolerar falhas foi mais utilizada para tratar tal falha. Tais informações são úteis para auxiliar na tomada de decisões em projetos que se encontrem em fase inicial, com questões relacionadas à qual nuvem utilizar, ou qual tipo de falha tratar, por exemplo.

Com relação aos objetivos propostos neste trabalho, pode-se considerar que foram alcançados, visto que se apresentou a descrição objetiva de cada solução, além da criação da tabela resumo com informações que buscam auxiliar na escolha de um sistema tolerante a falhas, concluindo com a criação de um *web site* para consultar as informações das tabelas resumo de maneira mais prática e acessível. Além disso, o trabalho feito por meio da RSL para a classificação dos sistemas, pode facilmente ser continuado, levando em considerações novos parâmetros (período de busca, *strings* de busca, etc), visto que a metodologia de classificação está bem descrita, assim como cada fase da RSL está bem detalhada.

O *web site* desenvolvido é multiplataforma e totalmente responsivo, como foi mostrado no final do Capítulo anterior, permitindo maior facilidade para acessar os dados obtidos por meio deste estudo.

5.2 Trabalhos Futuros

Do desenvolvimento deste trabalho foi possível identificar trabalhos que podem dar continuidade ou partir de seus resultados. De início, é possível continuar a RSL, considerando outros parâmetros como: novas *strings* de busca, outras bases acadêmicas, artigos do ano de 2018, etc.

Apesar de ser um trabalho de grande monta, também é interessante validar as propostas encontradas em diferentes nuvens, considerando diferentes parâmetros e *benchmarking*. Isso possibilitaria uma recomendação mais precisa sobre cada proposta.

Além disso, o *web site* criado é apenas um protótipo para um sistema de recomendação. Desse modo, é desejável melhorar esse protótipo com a criação de uma *Application Programming Interface* (API) que forneça algum grau de inteligência ao processo de recomendação. Com isso seria possível buscar os dados no sistema por meio de uma *string* de busca, trazendo como resultado sugestões específicas ou mesmo comparações entre técnicas, por exemplo. Um outro desenvolvimento possível, visto que o *web site* foi criado em Ionic, é possível facilmente gerar o *app* para plataformas Android e iOS, o que permitiria coloca-lo nas respectivas *Stores*, bastando ter o ambiente configurado com as ferramentas de desenvolvimento e, no caso do iOS, também será necessário um dispositivo com MacOS.

Referências

ACM. Association for Computing Machinery. <https://dl.acm.org/>. 2017. Citado na página 49.

AGARWAL, H.; SHARMA, A. A comprehensive survey of fault tolerance techniques in cloud computing. In: IEEE. *Computing and Network Communications (CoCoNet), 2015 International Conference on*. [S.l.], 2015. p. 408–413. Citado na página 44.

AHMED, N. O.; BHARGAVA, B. Mayflies: A moving target defense framework for distributed systems. In: ACM. *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. [S.l.], 2016. p. 59–64. Citado 2 vezes nas páginas 83 e 101.

AMAZON. Amazon. <https://aws.amazon.com/pt/>. 2017. Citado na página 28.

AMIN, Z.; SINGH, H.; SETHI, N. Review on fault tolerance techniques in cloud computing. *International Journal of Computer Applications*, Foundation of Computer Science, v. 116, n. 18, 2015. Citado na página 38.

AN, K. et al. A cloud middleware for assuring performance and high availability of soft real-time applications. *Journal of Systems Architecture*, Elsevier, v. 60, n. 9, p. 757–769, 2014. Citado 4 vezes nas páginas 72, 89, 93 e 100.

ANDERSON, T.; LEE, P. A. *Fault tolerance -principles and practice*. [S.l.]: Prentice-Hall, 1981. Citado na página 38.

ATAALLAH, S. M.; NASSAR, S. M.; HEMAYED, E. E. Fault tolerance in cloud computing-survey. In: IEEE. *Computer Engineering Conference (ICENCO), 2015 11th International*. [S.l.], 2015. p. 241–245. Citado na página 44.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004. Citado 2 vezes nas páginas 9 e 36.

AYUSO, P. N.; GASCA, R. M.; LEFEVRE, L. Ft-fw: A cluster-based fault-tolerant architecture for stateful firewalls. *computers & security*, Elsevier, v. 31, n. 4, p. 524–539, 2012. Citado 2 vezes nas páginas 77 e 78.

BALA, A.; CHANA, I. Fault tolerance-challenges, techniques and implementation in cloud computing. *IJCSI International Journal of Computer Science Issues*, v. 9, n. 1, p. 1694–0814, 2012. Citado 3 vezes nas páginas 39, 40 e 41.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. *Cloud Computing: Principles and Paradigms*. [S.l.]: Wiley, 2011. Citado 3 vezes nas páginas 9, 24 e 33.

BUYYA, R. et al. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, Elsevier, v. 25, n. 6, p. 599–616, 2009. Citado na página 23.

CAO, J. et al. Checkpointing as a service in heterogeneous cloud environments. In: IEEE. *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*. [S.l.], 2015. p. 61–70. Citado 4 vezes nas páginas 93, 99, 102 e 103.

CAPES. Plataforma Sucupira. <https://sucupira.capes.gov.br/sucupira/public/consultas/coleta/veiculoPublicacaoQualis/listaConsultaGeralPeriodicos.jsf>. 2018. Citado na página 116.

CARUTASU, G. et al. Cloud computing and windows azure. In: IEEE. *Electronics, Computers and Artificial Intelligence (ECAI), 2016 8th International Conference on*. [S.l.], 2016. p. 1–6. Citado na página 28.

CHERAGHLOU, M. N.; KHADEM-ZADEH, A.; HAGHPARAST, M. A survey of fault tolerance architecture in cloud computing. *Journal of Network and Computer Applications*, Elsevier, v. 61, p. 81–92, 2016. Citado 2 vezes nas páginas 41 e 44.

CHUNG, J. et al. Advance reservation access control using software-defined networking and tokens. *Future Generation Computer Systems*, Elsevier, v. 79, p. 225–234, 2018. Citado 2 vezes nas páginas 77 e 79.

CLOUDSTACK. Cloudstack. <https://cloudstack.apache.org/>. 2017. Citado na página 32.

DATTA, A.; CHO, W. H. Swifter: Elastic erasure coded storage system. In: IEEE. *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*. [S.l.], 2016. p. 229–238. Citado 3 vezes nas páginas 66, 67 e 83.

DING, Y.; YAO, G.; HAO, K. Fault-tolerant elastic scheduling algorithm for workflow in cloud systems. *Information Sciences*, Elsevier, v. 393, p. 47–65, 2017. Citado na página 96.

EGWUTUOHA, I. P. et al. A proactive fault tolerance approach to high performance computing (hpc) in the cloud. In: IEEE. *Cloud and Green Computing (CGC), 2012 Second International Conference on*. [S.l.], 2012. p. 268–273. Citado 2 vezes nas páginas 96 e 98.

EGWUTUOHA, I. P. et al. Energy efficient fault tolerance for high performance computing (hpc) in the cloud. In: IEEE. *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. [S.l.], 2013. p. 762–769. Citado na página 98.

ELSEVIER. Sciencedirect. <https://www.elsevier.com/solutions/sciencedirect>. 2017. Citado na página 49.

EUCALYPTUS SYSTEMS, INC. Official documentation for eucalyptus cloud. <https://docs.eucalyptus.cloud/eucalyptus/4.4.4/index.html>. 2017. Citado na página 29.

GARRAGHAN, P.; TOWNEND, P.; XU, J. Byzantine fault-tolerance in federated cloud computing. In: IEEE. *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*. [S.l.], 2011. p. 280–285. Citado 2 vezes nas páginas 89 e 90.

GILDER, G. The information factories. *Wired magazine*, v. 14, n. 10, p. 178–202, 2006. Citado na página 22.

GILL, N. K.; SINGH, S. A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers. *Future Generation Computer Systems*, Elsevier, v. 65, p. 10–32, 2016. Citado 2 vezes nas páginas 65 e 66.

GOKHROO, M. K.; GOVIL, M. C.; PILLI, E. S. Detecting and mitigating faults in cloud computing environment. In: IEEE. *Computational Intelligence & Communication Technology (CICT), 2017 3rd International Conference on*. [S.l.], 2017. p. 1–9. Citado 3 vezes nas páginas 96, 97 e 103.

GÓMEZ, A. et al. Fault-tolerant virtual cluster experiments on federated sites using bonfire. *Future Generation Computer Systems*, Elsevier, v. 34, p. 17–25, 2014. Citado 4 vezes nas páginas 89, 92, 101 e 103.

GONZALEZ, J. L. et al. An approach for constructing private storage services as a unified fault-tolerant system. *Journal of Systems and Software*, Elsevier, v. 86, n. 7, p. 1907–1922, 2013. Citado 3 vezes nas páginas 66, 67 e 83.

GONZALEZ, J. L. et al. Skycds: A resilient content delivery service based on diversified cloud storage. *Simulation Modelling Practice and Theory*, Elsevier, v. 54, p. 64–85, 2015. Citado na página 64.

GOOGLE, INC. Google App Engine. <https://cloud.google.com/appengine/>. 2017. Citado na página 29.

GOOGLE, INC. Google Scholar. <https://scholar.google.com.br/>. 2018. Citado na página 116.

GULENKO, A. et al. A system architecture for real-time anomaly detection in large-scale nfv systems. *Procedia Computer Science*, Elsevier, v. 94, p. 491–496, 2016. Citado 8 vezes nas páginas 65, 66, 77, 81, 83, 100, 102 e 103.

GUPTA, P.; GUPTA, S. Mobile cloud computing: the future of cloud. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, v. 1, n. 3, p. 134–145, 2012. Citado 3 vezes nas páginas 9, 22 e 26.

HARMS, R.; YAMARTINO, M. The economics of the cloud. *Microsoft whitepaper, Microsoft Corporation*, 2010. Citado na página 23.

HSUEH, M.-C.; TSAI, T. K.; IYER, R. K. Fault injection techniques and tools. *Computer*, IEEE, v. 30, n. 4, p. 75–82, 1997. Citado 3 vezes nas páginas 9, 42 e 43.

IBM. Ibm. <https://www.ibm.com/cloud-computing/br/pt>. 2017. Citado na página 29.

IBN-KHEDHER, H.; ABD-ELRAHMAN, E. Cdnaas framework: Topsis as multi-criteria decision making for vcdn migration. *Procedia Computer Science*, Elsevier, v. 110, p. 274–281, 2017. Citado 2 vezes nas páginas 83 e 84.

IEEE. Institute of Electrical and Electronics Engineering. <http://ieeexplore.ieee.org/xpl/aboutUs.jsp>. 2017. Citado na página 49.

JAVADI, B.; ABAWAJY, J.; BUYYA, R. Failure-aware resource provisioning for hybrid cloud infrastructure. *Journal of parallel and distributed computing*, Elsevier, v. 72, n. 10, p. 1318–1331, 2012. Citado na página 62.

KÄCHELE, S.; HAUCK, F. J. Coscanet-ft: Transparent network support for highly available cloud services. In: IEEE. *Networked Systems (NetSys), 2015 International Conference and Workshops on*. [S.l.], 2015. p. 1–5. Citado 2 vezes nas páginas 83 e 86.

KANSO, A. et al. Enhancing openstack fault tolerance for provisioning computing environments. In: IEEE. *High Assurance Systems Engineering (HASE), 2017 IEEE 18th International Symposium on*. [S.l.], 2017. p. 77–83. Citado na página 103.

KEELE, S. et al. Guidelines for performing systematic literature reviews in software engineering. In: *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. [S.l.]: sn, 2007. Citado na página 46.

KIRTHICA, S.; SRIDHAR, R. Cit: A cloud inter-operation toolkit to enhance elasticity and tolerate shut down of external clouds. *Journal of Network and Computer Applications*, Elsevier, v. 85, p. 32–46, 2017. Citado 4 vezes nas páginas 83, 87, 89 e 101.

KOREN, I.; KRISHNA, C. M. *Fault Tolerant Systems*. [S.l.]: Elsevier, 2007. Citado na página 37.

KRETSIS, A. et al. Mantis: Cloud-based optical network planning and operation tool. *Computer Networks*, Elsevier, v. 77, p. 153–168, 2015. Citado na página 77.

KULKARNI, B.; BHOSALE, V. Efficient storage utilization using erasure codes in openstack cloud. In: IEEE. *Inventive Computation Technologies (ICICT), International Conference on*. [S.l.], 2016. v. 3, p. 1–5. Citado 3 vezes nas páginas 66, 67 e 83.

KVM. Kernel-based Virtual Machine. <https://www.linux-kvm.org/page/Main_Page>. 2017. Citado na página 34.

LI, X. et al. An orchestration based cloud auto-healing service framework. In: IEEE. *Edge Computing (EDGE), 2017 IEEE International Conference on*. [S.l.], 2017. p. 190–193. Citado 2 vezes nas páginas 81 e 82.

LIU, J. et al. Pft-cckp: A proactive fault tolerance mechanism for data center network. In: IEEE. *Quality of Service (IWQoS), 2015 IEEE 23rd International Symposium on*. [S.l.], 2015. p. 79–80. Citado na página 100.

LU, L. et al. Morpho: A decoupled mapreduce framework for elastic cloud computing. *Future Generation Computer Systems*, Elsevier, v. 36, p. 80–90, 2014. Citado 2 vezes nas páginas 66 e 101.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011. Citado 3 vezes nas páginas 19, 23 e 25.

MESBAHI, M. R.; RAHMANI, A. M.; HOSSEINZADEH, M. Highly reliable architecture using the 80/20 rule in cloud computing datacenters. *Future Generation Computer Systems*, Elsevier, v. 77, p. 77–86, 2017. Citado na página 71.

MICROSOFT AZURE. Microsoft azure. <https://azure.microsoft.com/pt-br/>. 2017. Citado na página 28.

MICROSOFT MSDN. Microsoft msdn. <https://msdn.microsoft.com/pt-br/library/hh831531(v=ws.11).aspx>. 2017. Citado na página 34.

MOCA, M. et al. Multi-criteria and satisfaction oriented scheduling for hybrid distributed computing infrastructures. *Future Generation Computer Systems*, Elsevier, v. 55, p. 428–443, 2016. Citado 3 vezes nas páginas 89, 91 e 96.

MOGHADDAM, F. F.; GHERBI, A.; LEMIEUX, Y. Self-healing redundancy for openstack applications through fault-tolerant multi-agent task scheduling. In: IEEE. *Cloud Computing Technology and Science (CloudCom), 2016 IEEE International Conference on*. [S.l.], 2016. p. 572–577. Citado na página 102.

NAKANISHI, H. et al. Revised cloud storage structure for light-weight data archiving in lhd. *Fusion Engineering and Design*, Elsevier, v. 89, n. 5, p. 707–711, 2014. Citado 2 vezes nas páginas 66 e 68.

NATELLA, R.; COTRONEO, D.; MADEIRA, H. S. Assessing dependability with software fault injection: A survey. *ACM Computing Surveys (CSUR)*, ACM, v. 48, n. 3, p. 44, 2016. Citado na página 42.

NETFLIX. Netflix. <https://media.netflix.com/en/>. 2017. Citado na página 19.

NETTO, H. V. et al. State machine replication in containers managed by kubernetes. *Journal of Systems Architecture*, Elsevier, v. 73, p. 53–59, 2017. Citado 3 vezes nas páginas 83, 85 e 89.

OPENNEBULA. Opennebula. <https://opennebula.org/>. 2017. Citado 2 vezes nas páginas 9 e 32.

OPENSTACK. Openstack. <https://openstack.org/>. 2017. Citado na página 30.

PATRA, P. K.; SINGH, H.; SINGH, G. Fault tolerance techniques and comparative implementation in cloud computing. *International Journal of Computer Applications*, Foundation of Computer Science, v. 64, n. 14, 2013. Citado na página 38.

PAUL, S. et al. Application delivery in multi-cloud environments using software defined networking. *Computer Networks*, Elsevier, v. 68, p. 166–186, 2014. Citado 2 vezes nas páginas 83 e 88.

PEI, X. et al. Efficient in-place update with grouped and pipelined data transmission in erasure-coded storage systems. *Future Generation Computer Systems*, Elsevier, v. 69, p. 24–40, 2017. Citado 2 vezes nas páginas 83 e 89.

PETCU, D. et al. Portable cloud applications—from theory to practice. *Future Generation Computer Systems*, Elsevier, v. 29, n. 6, p. 1417–1430, 2013. Citado 2 vezes nas páginas 73 e 89.

POVEDANO-MOLINA, J. et al. Dargos: A highly adaptable and scalable monitoring architecture for multi-tenant clouds. *Future Generation Computer Systems*, Elsevier, v. 29, n. 8, p. 2041–2056, 2013. Citado na página 65.

PRATHIBA, S.; SOWVARNICA, S. Survey of failures and fault tolerance in cloud. In: IEEE. *Computing and Communications Technologies (ICCCT), 2017 2nd International Conference on*. [S.l.], 2017. p. 169–172. Citado na página 43.

QU, C.; CALHEIROS, R. N.; BUYYA, R. A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances. *Journal of Network and Computer Applications*, Elsevier, v. 65, p. 167–180, 2016. Citado 3 vezes nas páginas 74, 75 e 89.

REDHAT. Redhat. <https://access.redhat.com/documentation/en/>. 2017. Citado 3 vezes nas páginas 9, 30 e 31.

SCHARF, M. et al. Network-aware instance scheduling in openstack. In: IEEE. *Computer Communication and Networks (ICCCN), 2015 24th International Conference on*. [S.l.], 2015. p. 1–6. Citado na página 30.

SELIMI, M. et al. Cloud services in the guifi. net community network. *Computer Networks*, Elsevier, v. 93, p. 373–388, 2015. Citado na página 66.

SHAHRAD, M.; WENTZLAFF, D. Availability knob: Flexible user-defined availability in the cloud. In: ACM. *Proceedings of the Seventh ACM Symposium on Cloud Computing*. [S.l.], 2016. p. 42–56. Citado 3 vezes nas páginas 95, 102 e 103.

SHOOMAN, M. L. *Reliability of computer systems and networks: Fault tolerance*. [S.l.: s.n.], 2001. Citado na página 36.

SLAWINSKA, M.; SLAWINSKI, J.; SUNDERAM, V. Unibus: Aspects of heterogeneity and fault tolerance in cloud computing. In: IEEE. *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*. [S.l.], 2010. p. 1–10. Citado 3 vezes nas páginas 93, 94 e 100.

START. Start. <http://lapes.dc.ufscar.br/tools/start_tool>. 2017. Citado na página 49.

TAKANO, R. et al. Cooperative vm migration for a virtualized hpc cluster with vmm-bypass i/o devices. In: IEEE. *E-Science (e-Science), 2012 IEEE 8th International Conference on*. [S.l.], 2012. p. 1–8. Citado 4 vezes nas páginas 74, 76, 99 e 101.

TANENBAUM, A. S.; STEEN, M. V. *Distributed systems: principles and paradigms*. [S.l.]: Prentice-Hall, 2007. Citado 2 vezes nas páginas 36 e 37.

THONGTANUNAM, P. et al. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In: *Proceedings of the 38th International Conference on Software Engineering*. New York, NY, USA: ACM, 2016. (ICSE '16), p. 1039–1050. ISBN 978-1-4503-3900-1. Disponível em: <http://doi.acm.org/10.1145/2884781.2884852>. Citado na página 30.

TOFFETTI, G. et al. Self-managing cloud-native applications: Design, implementation, and experience. *Future Generation Computer Systems*, Elsevier, v. 72, p. 165–179, 2017. Citado 3 vezes nas páginas 96, 99 e 102.

VASILE, M.-A. et al. Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing. *Future Generation Computer Systems*, Elsevier, v. 51, p. 61–71, 2015. Citado na página 63.

VERAS, M. *Cloud Computing: nova arquitetura da TI*. [S.l.]: Brasport, 2015. Citado 4 vezes nas páginas 9, 23, 24 e 28.

VIJAYAKUMAR, D.; SRINIVASAGAN, K.; SABARIMUTHUKUMAR, R. Fir3: A fuzzy inference based reliable replica replacement strategy for cloud data centre. In: IEEE. *Computing and Network Communications (CoCoNet), 2015 International Conference on*. [S.l.], 2015. p. 473–479. Citado 2 vezes nas páginas 66 e 70.

VILLAMAYOR, J.; REXACHS, D.; LUQUE, E. A fault tolerance manager with distributed coordinated checkpoints for automatic recovery. In: IEEE. *High Performance Computing & Simulation (HPCS), 2017 International Conference on*. [S.l.], 2017. p. 452–459. Citado 2 vezes nas páginas 89 e 90.

VMWARE. Vmware. <https://www.vmware.com/>. 2017. Citado na página 33.

VOGEL, A. et al. Private iaas clouds: a comparative analysis of opennebula, cloudstack and openstack. In: IEEE. *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*. [S.l.], 2016. p. 672–679. Citado na página 32.

WAHLER, M. et al. Fasa: A software architecture and runtime framework for flexible distributed automation systems. *Journal of Systems Architecture*, Elsevier, v. 61, n. 2, p. 82–111, 2015. Citado na página 89.

WANG, T. et al. Designing efficient high performance server-centric data center network architecture. *Computer Networks*, Elsevier, v. 79, p. 283–296, 2015. Citado 3 vezes nas páginas 66, 83 e 88.

WU, Y. et al. Model defined fault tolerance in cloud. In: ACM. *Proceedings of the 6th Asia-Pacific Symposium on Internetware on Internetware*. [S.l.], 2014. p. 116–119. Citado 4 vezes nas páginas 95, 101, 102 e 103.

XENSERVER. Zenserver. <https://xenserver.org/>. 2017. Citado na página 34.

XU, Y. et al. Identifying sdn state inconsistency in openstack. In: ACM. *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. [S.l.], 2015. p. 11. Citado 3 vezes nas páginas 77, 80 e 103.

YANG, C.-T. et al. Accessing medical image file with co-allocation hdfs in cloud. *Future Generation Computer Systems*, Elsevier, v. 43, p. 61–73, 2015. Citado na página 71.

ZAHID, F. et al. Efficient network isolation and load balancing in multi-tenant hpc clusters. *Future Generation Computer Systems*, Elsevier, v. 72, p. 145–162, 2017. Citado 2 vezes nas páginas 77 e 81.

ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, Springer, v. 1, n. 1, p. 7–18, 2010. Citado 3 vezes nas páginas 9, 26 e 27.

ZHANG, X. et al. A hybrid approach for scalable sub-tree anonymization over big data using mapreduce on cloud. *Journal of Computer and System Sciences*, Elsevier, v. 80, n. 5, p. 1008–1020, 2014. Citado 2 vezes nas páginas 66 e 68.

ZHAO, K. et al. Onhelp: Components in building secure cloud based on opennebula. In: IEEE. *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*. [S.l.], 2014. p. 320–327. Citado 3 vezes nas páginas 75, 93 e 99.

ZHAO, M. et al. Multi-level vm replication based survivability for mission-critical cloud computing. In: IEEE. *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. [S.l.], 2015. p. 1351–1356. Citado na página 101.

APÊNDICE A – Qualis e número de citações dos artigos utilizados na etapa final da RSL

A Tabela 70 traz o título, qualis, ano e citações dos artigos utilizados na seção 4. Para o número de citações utilizou-se o Google Scholar (GOOGLE, INC, 2018) e para o levantamento de qualis a Plataforma Sucupira (CAPES, 2018), considerando o quadriênio de 2013 a 2016

Tabela 70 – Qualis e Número de citações dos artigos utilizados na última etapa da RSL.

Título	Qualis	Ano	Citações (22/04/2018)
Mayflies: A Moving Target Defense Framework for Distributed Systems	-	2016	3
Identifying SDN State Inconsistency in OpenStack	-	2015	6
Availability Knob: Flexible User-Defined Availability in the Cloud	-	2016	8
Enhancing OpenStack Fault Tolerance for Provisioning Computing Environments	-	2017	1
Efficient storage utilization using erasure codes in OpenStack cloud	-	2016	0
SwiftER: Elastic Erasure Coded Storage System	-	2016	0
Multi-level VM replication based survivability for mission-critical cloud computing	-	2015	2
An Orchestration Based Cloud Auto-Healing Service Framework	-	2017	1
Self-Healing Redundancy for OpenStack Applications through Fault-Tolerant Multi-Agent Task Scheduling	-	2016	0
Checkpointing as a Service in Heterogeneous Cloud Environments	-	2015	8
A System Architecture for Real-time Anomaly Detection in Large-scale NFV Systems	C	2016	4
An approach for constructing private storage services as a unified fault-tolerant system	A2	2013	19
CIT: A Cloud Inter-operation Toolkit to enhance elasticity and tolerate shut down of external clouds	A2	2017	3
Revised cloud storage structure for light-weight data archiving in LHD	A2	2014	5
Highly reliable architecture using the 80/20 rule in cloud computing datacenters	A2	2017	2
DARGOS: A highly adaptable and scalable monitoring architecture for multi-tenant Clouds	A2	2013	75
Mantis: Cloud-based optical network planning and operation tool	A1	2015	5
A cloud middleware for assuring performance and high availability of soft real-time applications	B1	2014	34
Self-managing cloud-native applications: Design, implementation, and experience	A2	2017	12
Resource-aware hybrid scheduling algorithm in heterogeneous distributed computing	A2	2015	97
Advance reservation access control using software-defined networking and tokens	A2	2017	5
CDNaaS Framework: TOPSIS as Multi-Criteria Decision Making for vCDN Migration	C	2017	1
Efficient network isolation and load balancing in multi-tenant HPC clusters	A2	2017	5
Portable Cloud applications - From theory to practice	A2	2013	171
Pilot-Data: An abstraction for distributed data	A2	2015	23
A hybrid approach for scalable sub-tree anonymization over big data using MapReduce on cloud	A2	2014	72

(conclusão)

Título	Qualis	Ano	Citações (22/04/2018)
Cloud services in the Guifi.net community network	A1	2016	3
SkyCDS: A resilient content delivery service based on diversified cloud storage	B2	2015	6
Efficient in-place update with grouped and pipelined data transmission in erasure-coded storage systems	A2	2016	8
Fault-tolerant virtual cluster experiments on federated sites using BonFIRE	A2	2017	1
Accessing medical image file with co-allocation HDFS in cloud	A2	2016	0
Morpho: A decoupled MapReduce framework for elastic cloud computing	A2	2016	0
Multi-criteria and satisfaction oriented scheduling for hybrid distributed computing infrastructures	A2	2015	2
Failure-aware resource provisioning for hybrid Cloud infrastructure	A2	2017	1
FT-FW: A cluster-based fault-tolerant architecture for stateful firewalls	A2	2016	0
Fault-tolerant elastic scheduling algorithm for workflow in Cloud systems	A1	2015	8
Designing efficient high performance server-centric data center network architecture	A1	2016	4
FASA: A software architecture and runtime framework for flexible distributed automation systems	B1	2013	19
Application delivery in multi-cloud environments using software defined networking	A1	2017	3
A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers	A2	2014	5
State machine replication in containers managed by Kubernetes	B1	2017	2
A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances	A2	2013	75
A Fault Tolerance Manager with Distributed Coordinated Checkpoints for Automatic Recovery	-	2015	5
Energy Efficient Fault Tolerance for High Performance Computing (HPC) in the Cloud	-	2014	34
A Proactive Fault Tolerance Approach to High Performance Computing (HPC) in the Cloud	-	2017	12
Cooperative VM migration for a virtualized HPC cluster with VMM-bypass I/O devices	-	2015	97
Byzantine fault-tolerance in federated cloud computing	-	2017	5
Detecting and mitigating faults in cloud computing environment	-	2017	1
PFT-CCKP: A proactive fault tolerance mechanism for data center network	-	2013	171
COSCAnet-FT: Transparent network support for highly available cloud services	-	2015	23
Unibus: Aspects of heterogeneity and fault tolerance in cloud computing	-	2014	72
ONHelp: Components in Building Secure Cloud Based on OpenNebula	-	2014	0
FIR3: A fuzzy inference based reliable replica replacement strategy for cloud Data Centre	-	2015	0
Model defined fault tolerance in cloud	-	2014	0

Fonte: Elaborado pelo autor.