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

Camila Baleiro Okado Tamashiro

Modelagem de falhas em nuvem

São José do Rio Preto
2020

Camila Baleiro Okado Tamashiro

Modelagem de falhas em nuvem

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: Profª. Drª. Renata Spolon Lobato

São José do Rio Preto
2020

T153m

Tamashiro, Camila Baleiro Okado
Modelagem de falhas em nuvem / Camila Baleiro Okado Tamashiro. -- São José do Rio Preto, 2020
68 p. : tabs., fotos

Dissertação (mestrado) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientadora: Renata Spolon Lobato

1. Computação em nuvem. 2. Simulação. 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.

Camila Baleiro Okado Tamashiro

Modelagem de falhas em nuvem

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.

Comissão Examinadora

Profª. Drª. Renata Spolon Lobato
UNESP – Câmpus de São José do Rio Preto
Orientadora

Prof. Dr. Rodrigo Capobianco Guido
UNESP – Câmpus de São José do Rio Preto

Prof. Dr. Henrique Dezani
FATEC - São José do Rio Preto

São José do Rio Preto
22 de outubro de 2020

A meu esposo Ricardo.

AGRADECIMENTOS

Ter cursado a pós-graduação em ciência da computação foi uma grande conquista pessoal. Tive muita satisfação a cada chegada ao campus e muita alegria em despedir-me semanalmente. Tenho orgulho de dizer que sim, eu fiz minha pós-graduação no IBILCE, em São José do Rio Preto.

Durante cinco anos, as inúmeras tentativas e apoio que obtive para esta jornada com certeza fizeram a diferença para que o dia de hoje chegasse. Agradeço ao corpo docente da pós-graduação em ciência da computação, em especial ao Prof. Aleardo por me aceitar como sua discente e também como colega de profissão.

Durante o período desses cinco anos, tive momentos de incertezas, superação, conquistas, alegrias e muito aprendizado. Agradeço a Deus pela força e perseverança para que eu continuasse nos dias mais solitários e difíceis. Em momentos que eu abri mão de escolhas profissionais e pessoais para que eu pudesse fazer essa caminhada sozinha, mas com muito apoio e orientação de pessoas que gostaria que fossem lembradas neste trabalho.

Assim, agradeço: a meus pais, Edson e Elaine, à minha irmã Jéssica, a meu esposo Ricardo. Agradeço também meus demais familiares pelo apoio nesta jornada e compreensão das minhas ausências.

Ao meu amigo Douglas que me incentivou a iniciar este desafio e a meus amigos Ricardo, Rogério, Reginaldo, Sérgio, Carlos, Tiago, Henrique e demais colegas de trabalho que sempre me apoiaram, compreenderam e me ajudaram nos momentos em que precisei.

Aos colegas do Grupo de Sistemas Paralelos e Distribuídos da UNESP, Lucas, Fernanda, Luis Vinicius, Fábio, João, Vinícius e Wellington. Agradeço especialmente à minha orientadora, Profª. Drª Renata Spolon pela compreensão e ensinamentos, e ao Prof. Dr. Aleardo Manacero pelas demais contribuições e orientações.

RESUMO

Neste trabalho é apresentada a modelagem de falhas para o simulador iSPD *(iconic Simulator of Parallel and Distributed systems),* assim como o desenvolvimento de gerador de falhas e experimentos realizado. Para isso, são apresentadas as pesquisas de revisão sistemática da literatura, evolução da computação em nuvem, e as técnicas de tratamento proativas e reativas de tratamento de falhas que podem ser aplicadas em caso de ocorrência de falhas nesta arquitetura, assim como as classes que a compõem e sua integração. Em relação aos tipos de falhas, foram estudadas as principais técnicas empregadas em sistemas tolerantes a falhas, em especial na computação em nuvem para que houvesse a sua modelagem. Para a realização dos experimentos, foi utilizado o simulador já existente, o iSPD. Neste, foi acrescido e atualizado algumas interfaces icônicas, o desenvolvimento do gerador de falhas e sua biblioteca foram integrados ao iSPD com posterior realização dos testes quanto a modelagem a falhas. Os resultados dos experimentos atenderam os resultados esperados, bem como validou a proposta da modelagem de falhas em nuvem.

Palavras–chave: Computação em nuvem. Simulação. Tolerância a falhas.

ABSTRACT

This work presents the fault modeling for the iSPD simulator (iconic Simulator of Parallel and Distributed systems), as well as the development of the failure generator and experiments carried out. For this, researches on systematic literature review, evolution of cloud computing are presented, as well as the proactive and reactive treatment techniques for failure treatment that can be applied in case of failure in this architecture, as well as the classes that make up and their integration. In relation to the types of failures, the main techniques employed in fault-tolerant systems were studied, especially in cloud computing so that they could be modeled. To carry out the experiments, the existing simulator, the iSPD, was used. In this, some iconic interfaces were added and updated, the development of the fault generator and its library were integrated to the iSPD with subsequent testing of the fault modeling. The results of the experiments met the expected results, as well as validated the proposal for cloud failure modeling.

Keywords: Cloud computing. Simulation. Fault tolerance.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

ACM	*Association of Computing Machinery*
BS	*Base Stations*
CaaS	*Container as a Service*
CDCs	*Cloud Data Centers*
CRAN	*Cloud-Assisted Radio Access Network*
CSPs	*Cloud Service Providers*
CCU	*Cloudbased Central Unit*
DdoS	*Distributed Denial of Service*
EUA	Estados Unidos da América
GCC	*Green Cloud Computing*
GSPD	Grupo de Sistemas Paralelos e Distribuídos
IaaS	Infraestrutura como Serviço
IEEE	*Institute of Electrical and Electronics Engineering*
iSPD	*iconic Simulator of Parallel and Distributed systems*
MCC	*Mobile Cloud Computing*
Mon*PaaS*	*Monitoring Platform as a Service*
MaaS	*Monitoring as a Service*
MNOs	*Mobile Network Operator*
PaaS	Plataforma como Serviço *(Platform as a Service)*
QaaS	Qualidade como Serviço
RU	*Radio Unit*
RSL	Revisão Sistemática da Literatura
SaaS	*Software* como Serviço *(Software as a Service)*
SLA	*Service Leve Agreement*
SOA	Serviços Orientados à Arquitetura
SOC	*Service Oriented Computing*
VM	*Virtual Machine*
VMM	*Virtual Machine Monitor*

Sumário

CAPÍTULO 1 INTRODUÇÃO

O impacto da Internet e o aumento do volume de dados produzidos mundialmente geram a necessidade de melhoria contínua dos serviços oferecidos no segmento da computação em nuvem: seja na infraestrutura, plataforma ou *software*, para que haja melhor interação nos serviços oferecidos sob demanda a partir do compartilhamento de seus recursos (MELL *et al.*, 2011).

Na arquitetura de computação em nuvem, é possível acompanhar o crescimento exponencial das informações, consequentemente, a necessidade de processamento, gerenciamento e disponibilização de grandes conjuntos de dados.

Em virtude deste crescimento, há a preocupação das empresas atuantes nesta área quanto à disponibilidade, confiabilidade e segurança da informação dos seus serviços entregues (BHADAURIA, *et al.*, 2011). As empresas que atuam neste segmento têm demonstrado a preocupação em prever as falhas que podem ocorrer neste ambiente para que estejam preparadas para aplicar técnicas de tolerância e de recuperação de falhas caso haja sua ocorrência, para evitar a indisponibilidade dos serviços oferecidos sob demanda (REZAEIPANAH *et al.*, 2020).

Neste trabalho é apresentada uma revisão bibliográfica sobre computação em nuvem, sua taxonomia e suas classes de serviços. Também é apresentada uma revisão bibliográfica sobre falhas nesta arquitetura, sua taxonomia, falhas em computação em nuvem, principais técnicas para tratamento e principais falhas relatadas em alguns gestores de nuvem na atualidade.

O objetivo deste trabalho foi modelar falhas e seu respectivo tratamento para falhas em computação em nuvem no simulador de eventos discretos - *iconic Simulator of Parallel and Distributed Systems - iSPD* - do Laboratório de Sistemas Paralelos e Distribuídos do Departamento de Ciências de Computação e Estatística da Unesp de São José do Rio Preto – o LSPD, que será explicado no Capítulo 4.

O trabalho está organizado da seguinte forma:

No **Capítulo 2** são apresentados conceitos da computação em nuvem e suas classes de serviço.

No **Capítulo 3** são apresentados conceitos de falhas, tolerância a falhas em sistemas e em computação em nuvem com suas respectivas técnicas de tratamento, e as técnicas de tolerância a falhas dotadas nos principais gestores de nuvem.

No **Capítulo 4** são apresentados a modelagem de falhas em nuvem e são descritos o desenvolvimento da implementação da modelagem de falhas no iSPD.

No **Capítulo 5** são apresentados os resultados.

Por fim, no **Capítulo 6** é apresentada a conclusão.

CAPÍTULO 2 – COMPUTAÇÃO EM NUVEM

Neste capítulo são apresentados os principais conceitos e classes de serviços para a computação em nuvem. Além dos conceitos apresentados, no texto são abordadas também as principais características existentes neste tipo de computação e a atuação dos gestores de nuvem para auxiliar na estabilidade dos serviços oferecidos sob demanda.

2.1. Evolução, conceitos e taxonomia

A computação em nuvem pode ser compreendida como a integração em três níveis de ofertas de Serviço denominadas: "*Software* como Serviço", do inglês *"Software as a Service"* (*SaaS*), "Plataforma como Serviço", do inglês *"Platform as a Service"* (*PaaS*) e "Infraestrutura como Serviço", do inglês *"Infrastructure as a service"* (*IaaS*). (BHARDWAJ *et al.,*2010).

Outra definição, utilizada atualmente pelo Instituto Nacional de Padrões e Tecnologia (*National Institute of Standards and Technology - NIST) "[...] a computação em nuvem é como um modelo que fornece conveniência e acesso sob-demanda para recursos compartilhados (redes, servidores e serviços) que podem ser rapidamente entregues quase sem nenhum esforço de gestão por parte dos usuários."* (MELL *et al.,* 2011).

A partir dessa definição, as nuvens são classificadas em três categorias: nuvens públicas, privadas ou híbridas.

Nas nuvens públicas, há o compartilhamento de recursos para diferentes usuários e organizações (SYED *et al.*, 2017). Neste compartilhamento, a segurança de infraestrutura física se torna essencial e tem como grande desafio a manutenção da privacidade e segurança, uma vez que, nesta classe de serviço, todas as Máquinas Virtuais compartilham as mesmas Máquinas Físicas (SANAEI *et al.,* 2013).

As nuvens privadas são nuvens de propriedade da uma organização, assim como sua infraestrutura física e virtual, com políticas de monitoramento e de níveis de acesso (SYED *et al.*, 2017).

Já as nuvens híbridas são a combinação entre as nuvens públicas e privadas em que as infraestruturas físicas ou virtuais podem se alterar em função à sua organização (SYED *et al.*, 2017).

Nestes conceitos é demonstrada que a taxonomia da computação em nuvem pode evoluir em virtude da variedade de serviços ofertados – definidos como classes de serviços.

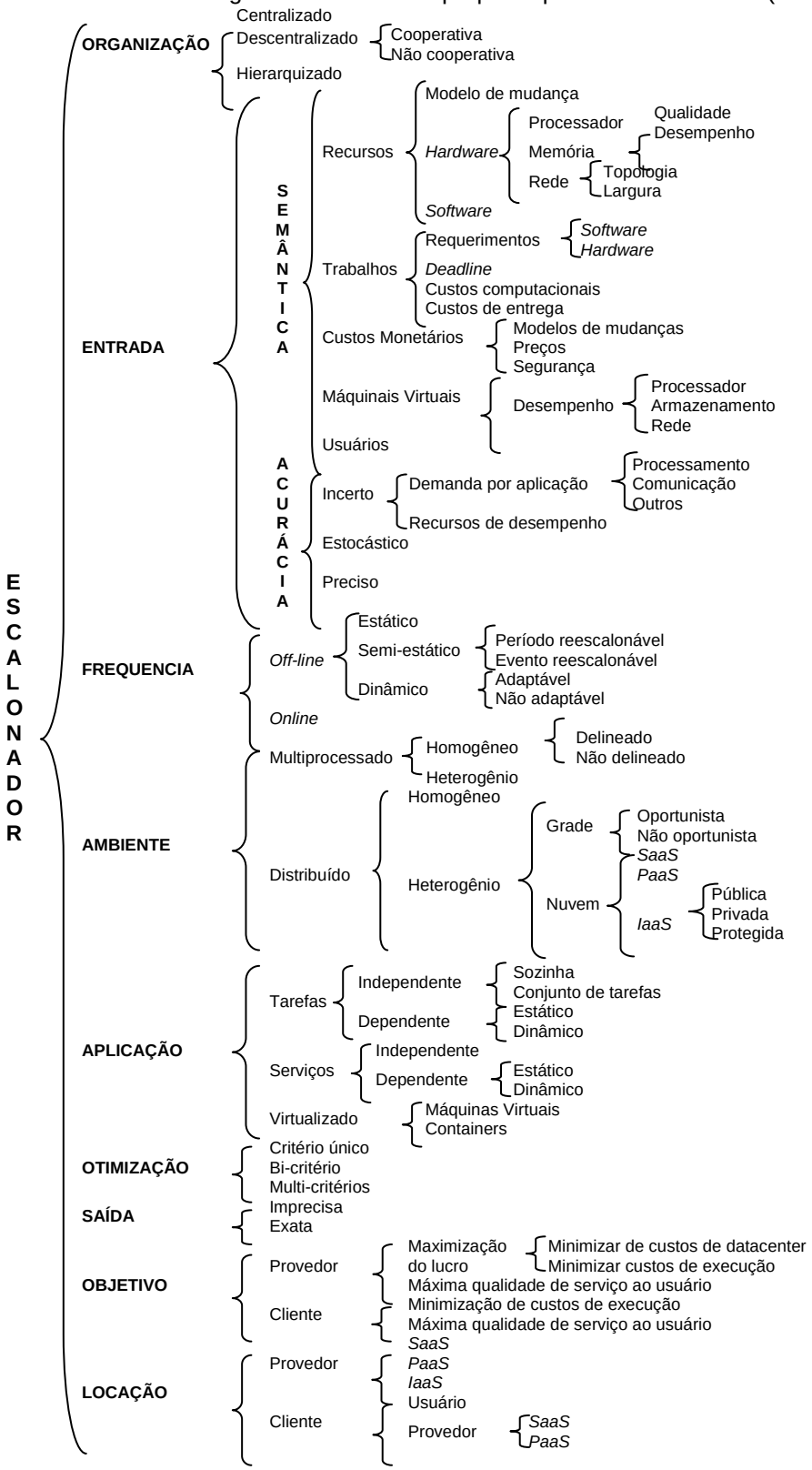
A taxonomia apresentada por BITTENCOURT *et al.* (2018) é organizada a partir dos escalonadores ou planejadores em nuvem, e, sucessivamente, a sua organização, entrada, frequência, ambiente, aplicação, otimização, saída, objetivo e locação. Desta forma, ela pode ser aplicada em ambientes de arquitetura homogêneos e heterogêneos, apresentado na Figura 1, os quais podem ser observados a partir do ambiente distribuído.

Quando o ambiente distribuído é organizado para permitir ubiquidade, a conveniência e o acesso sob demanda para a utilização de recursos computacionais compartilhados, e a computação em nuvem é estruturada de forma que as somas de diversos recursos computacionais resultem na eficiência do armazenamento em nuvem e também a sua praticidade em retornar os dados requisitados.

Na Figura 1, é observada a organização com subdivisões entre sistemas *online* e *off-line*, assim como os ambientes que podem ser compostos por ambientes multiprocessados (em ambiente homogêneo ou heterogênio) ou ambientes de sistemas distribuídos (em ambientes homogêneos e heterogêneos).

Quando os ambientes em nuvem são heterogêneos, é observada a heterogeneidade aplicada na classe de serviço de infraestrutura como serviço, classificando a nuvem como pública, privada e híbrida.

FIGURA 1: Visão geral da taxonomia proposta por Bittencourt *et al.* (2018)



FONTE: Bittencourt *et al.* (2018). Traduzido pela autora

2.2. Classes de serviços

A classe de serviço “Infraestrutura como Serviço”, do inglês *“Infrastructure as a service”* (*IaaS*), é responsável pela infraestrutura de computação instantânea, provisionada e gerenciada pelos serviços de Internet (SYED *et al.*, 2017). Por exemplo, a plataforma Azure da Microsoft (2020) e a IBM Rational Team Concert (2020) oferecem serviços relacionados à *IaaS*, em que é permitido seu aumento e a diminuição da *IaaS* e *PaaS* de acordo com a demanda de serviço.

Já a terminologia de “Plataforma como Serviço”, do inglês *“Platform as a Service” (PaaS)*, é a classe de serviço intermediária, entre as classes de serviço de *software* e de infraestrutura. Muitas empresas oferecem soluções de serviços para atuarem nesta camada, conhecidos como gestores de nuvem. Neles, é permitido o fornecimento de aplicativos habilitados para serem trabalhados em computação em nuvem.

A Classe de serviço “*Software* como Serviço”, do inglês *“Software as a Service”* (*SaaS*), pode ser compreendida como o *software* que atua na entrega de serviços prestados em computação na nuvem, e pode atuar diretamente nas infraestruturas subjacentes, como *middleware*, *software* de aplicativo e dados de aplicativos *(*Microsoft, 2019).

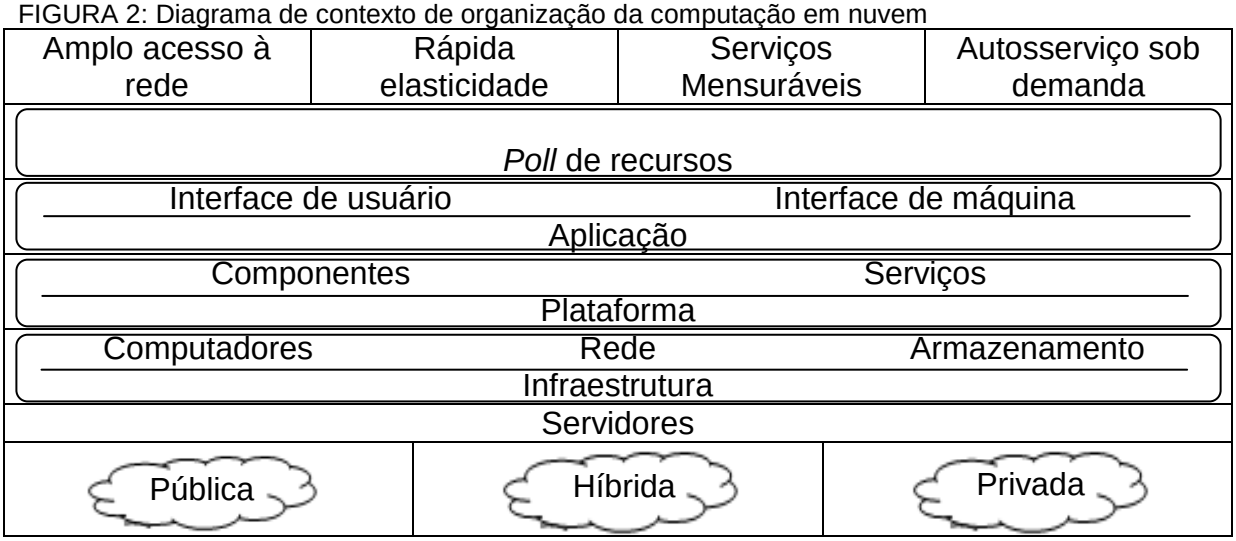
Esta classe de serviço está interligada a todos os conjuntos de recursos ofertados na computação em nuvem, uma vez que há a interação entre os serviços oferecidos e mensuráveis, o amplo acesso a rede e, principalmente à sua capacidade de rápida elasticidade para atender os serviços sob sua demanda.

2.3. Integração entre as classes de serviços

A integração entre as classes de serviço ocorre com a organização, virtualização e automação dos recursos computacionais, entre eles: servidores, armazenamento e rede*.* Na virtualização é permitida a integração e o gerenciamento de diferentes classes de serviços, nos diferentes tipos de nuvens.

Esta integração é apresentada na Figura 2, em que é exibida a integração destas classes e a disponibilização de seus recursos, adaptado de Kumari et al (2018).

FIGURA 2: Diagrama de contexto de organização da computação em nuvem

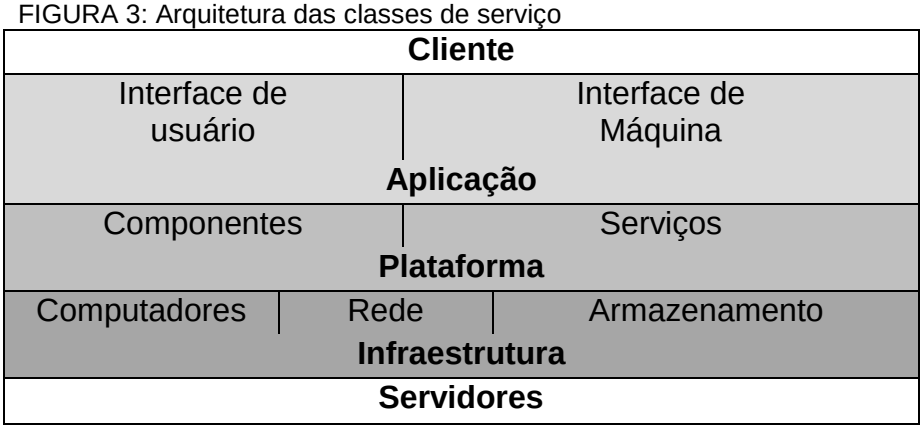


FONTE: Kumari *et al.*(2018), traduzido e adaptado pela autora

Para atender a integração das novas tecnologias, a computação em nuvem precisou ser remodelada para que seus recursos fossem configuráveis de acordo com a necessidade de seus usuários (BHARDWAJ *et al.* 2010).

Na Figura 3 é apresentado um modelo de arquitetura de classes de serviços proposta por Pijanowski (2009) *et al*. No modelo é proposto a integração entre quatro camadas, composta por: infraestrutura (composta por servidores, redes e armazenamento), plataforma (composta por componentes e serviços), aplicação (composta por interface de usuário e interface da máquina) e cliente.

FIGURA 3: Arquitetura das classes de serviço



FONTE: Pijanowski *et al*.(2009), traduzida e adaptada

Para garantir segurança, a eficiência e bom tráfego de dados entre as camadas, há empresas que oferecem ferramentas gratuitas ou proprietárias que atuam como gestores de nuvem como mencionado anteriormente. Neles, os

serviços entre as camadas podem ser gerenciados com maior garantia de disponibilidade e segurança do aplicativo e de seus dados (BUYYA *et al.*, 2014).

Existem serviços de computação em nuvem que atuam massivamente no mercado, conhecidos como gestores de nuvem, fato que tem auxiliado seu crescimento astronômico (CECCI, 2017), como por exemplo: a *AWS*, *Microsoft Windows Azure Plataform, Google App Engine*, a *IBM Cloud.*

Inseridos nos sistemas de computação em nuvem, existem os gestores de computação em nuvem. Eles atuam diretamente com a integração das diversas camadas de *software* e *hardware* (ANDRADE, 2019) necessárias para a prestação deste serviço sob demanda. Esses gestores podem atuar especificamente em determinadas classes de serviços ou oferecendo suporte como forma de integrar as classes de serviço, entre eles estão: *Eucalyptus, OpenStack, Open Nebula* e *Apache CloudStack.*

2.4 Considerações finais

Neste capítulo foram apresentados os principais conceitos da computação em nuvem, como também sua taxonomia, sua integração e os principais gestores de nuvens existentes no mercado atualmente. Esta arquitetura tem desafios internos a serem vencidos de forma que os usuários finais possam gerenciar os recursos disponibilizados em diferentes tipos de nuvens, e ofereça mais autonomia à sua utilização. No próximo capítulo serão apresentadas técnicas de tolerância a falhas existentes para a computação em nuvem, assim como sua taxonomia.

CAPÍTULO 3 TOLERÂNCIA A FALHAS

Neste capítulo são apresentados os principais conceitos de tolerância a falhas em sistemas de computação em nuvem e técnicas para evitá-las ou corrigi-las.

3.1 – Definições e Taxonomia sobre as falhas

A falha é um evento que pode ocorrer no sistema após este sofrer uma falta - que ocorre após a existência de erros (ARGAWAL, 2015). Elas podem ocorrer em ambientes de *hardware* e de *software*, desta forma, as falhas são classificadas em sete tipos.

No Quadro 1, é apresentado a Taxonomia das falhas proposto por Argawal (2015), em que foram classificadas de acordo com as suas principais características e sua ocorrência. No Quadro 1 é apresentada a classificação das falhas que podem ocorrer tanto no ambiente lógico como no ambiente físico.

QUADRO 1: Taxonomia das falhas

Falhas	**Tipos de falhas**
Falha por omissão	- *Hardware*
	- *Software*
Envelhecimento	-Negação de serviços
	-DISCO RÍGIDO (HD) Cheio
Falhas de resposta	- Valores
	- Transição de estado
Falhas de *software*	- Transiente ou intermitente
Falhas de temporização	- Falhas precoses
	- Falhas tardias
Falhas de interação	- Sobrecarga de tempo
	- Falhas tardias
Falhas diversas	- Falhas permanentes
	- Desenho incorreto

FONTE: Argawal (2015). Traduzido e adapatado pela autora

3.2. Técnicas de tolerância a falhas

Após surgir, a falha é detectada e analisada de acordo com a sua ocorrência para ser tratada (KUMARI *et al.*, 2018). No seu tratamento, pode ser que a falha receba técnicas de tratamento reativas (latentes) ou técnicas proativas (ativas).

As técnicas reativas de tratamento de falhas são aplicadas quando existe a possibilidade de prever a falha antes de sua ocorrência (KUMARI *et al.*, 2018). Ou seja, o local onde ela pode ocorrer possui técnicas de tratamento de falhas implementadas, o que garante maiores chances de tratamento caso venha ocorrer e atesta que o sistema é mais robusto e confiável.

As principais técnicas reativas de tratamento de falhas e sua respectiva descrição podem ser observadas no Quadro 2, de acordo a revisão sistemática da literatura proposta por KUMARI *et al.* (2018).

QUADRO 2: Técnicas de tolerância a falhas reativas

Técnicas	Descrição
Check-point (Ataallah *et al.* 2015; Hosseini e Arani, 2015)	Usada para salvar os estados do sistema periodicamente Em caso de alguma falha, o trabalho é reiniciado desde o começo.
Migração de trabalho (Prathiba e Sowvarnica, 2017)	Se uma tarefa não puder concluir sua execução em alguma máquina física específica devido a algum motivo e falhar, ela será migrada para outra máquina.
Replicação (Amin *et al*., 2015; Hosseini e Arani, 2015)	Usado para criar várias cópias de tarefas e armazenar réplicas em locais diferentes.
S-Guarda (Bala e Chana, 2012)	Isso depende do *roollback* e do processo de recuperação
Tente novamente (Prathiba e Sowvarnica, 2017)	Neste processo, a tarefa é executada repetidamente até atingir o sucesso. O mesmo recurso é usado repetidamente até a falha da tarefa ou seu insucesso.
Re submissão de tarefa (Amin *et al.*, 2015; Ataallah *et al.*.,2015)	Neste método, a tarefa que ocorreu a falha é submetida novamente até identificar o recurso a diversas máquinas para execução.
Resgate do *workflow* (Prathiba e Sowvarnica, 2017)	O sistema é habilitado para continuar trabalhando até depois da falha da tarefa ou trabalho até que não seja possível prosseguir sem corrigir a falha

Fonte: KUMARI *et al.* (2018), adaptado pela autora

As técnicas proativas ou ativas de tratamento de falhas podem ser compreendidas como a utilização de um conjunto de técnicas para prever a ocorrência de falhas e tratá-las antes de sua ocorrência, como por exemplo, a substituição de componentes de *hardwares* suspeitos e monitoramento da rede de comunicação.

No Quadro 3 são apresentadas as técnicas proativas ou ativas de tratamento de falhas e sua descrição, organizadas e descritas por KUMARI *et al.* (2018) em seu *review*.

QUADRO 3: Processamento de erros e tratamento proativo de falhas

Técnicas	Descrição
Self-Healing (Saikia and Devi, 2014)	Grandes tarefas que podem ser decompostas em múltiplos pedaços. Ela é feita para melhorar o desempenho do sistema quando uma mesma aplicação está executando em várias instâncias de Máquinas Virtuais e ocorrem falhas em uma dessas instâncias. Isso permite aos dispositivos do sistema identificá-los e reorganizá-los sem a dependência do administrador.
Rejuvenescimento de *software* (Amin *et al.*, 2015)	Quando o sistema sofre repetidas reinicializões e recomeça um novo estado de energia.
Migração preemptiva (Bala e Chana, 2012; Engelmann *et al.*, 2009, 2009)	A aplicação é observada e analisada constantemente isso depende do controle e métodos de devolutivas ao usuário
Balanceamento de carga (Nazari Cheraghlou *e. al.*,2016; Rimal *et al.,* 2009; Sing e Kinger, 2013)	É usado o balanceamento de carga de memória e de CPU.

Fonte: KUMARI *et al.* (2018), adaptado pela autora

A ocorrência da falha em qualquer classe de serviço em computação em nuvem implica na detecção da sua ocorrência e a aplicação de técnicas de tratamento eficazes de forma que haja o menor impacto possível na disponibilidade dos serviços ofertados.

Desta forma, o processo de detecção e tratamento de falhas é composto pelo diagnóstico da falha, seu tratamento e por consequência, sua recuperação (FRITZ *et al.*, 2018).

Para que a falha seja detectada de maneira eficaz, é necessário que o ambiente se encontre tolerante a falha, ou seja, tenha técnicas de tratamento de falhas implementadas para que possam ser aplicadas.

No estágio de pré-detecção, é possível que uma falha seja detectada pelo sistema tolerante a falha, mas não seja classificada, quando há casos de ambiguidade. O mesmo ocorre em casos em que há a ocorrência de falhas, mas o sistema tolerante a falhas não consegue identificá-las devido à ambiguidade existente (FRITZ *et al.*, 2018).

Em casos que não há a existência de ambiguidade, durante o processo de detecção, diferentes variáveis são avaliadas a partir de parâmetros já existentes em sistemas tolerantes a falhas, para que seja possível sua detecção, sua classificação e aplicação da melhor técnica de tratamento.

Em sistemas de computação em nuvem, quando há falhas de *hardware* ou falhas no envio e recebimento de dados (falhas de *sites, links*, mudança de configuração entre outros), outros servidores assumem as funções e há a redistribuição de tarefas (ARGAWAL *et al.*, 2015).

A utilização de técnicas de tolerância a falhas em ambientes computacionais buscam garantir a execução dos sistemas com disponibilidade, segurança e confidencialidade para manter o sistema em funcionamento após a ocorrência de uma falta, falha ou mesmo erro.

Caso alguma falha ocorra no sistema, é necessário que o sistema seja tolerante, ou seja, é necessário que ele recupere e volte a operar com o menor ou nenhum impacto nos serviços ofertados (ANDRADE, 2017).

Para que o sistema seja tolerante, várias técnicas podem ser aplicadas, para isso é necessário a sua detecção, recuperação e tratamento. No Quadro 4 são apresentadas as fases para aplicação de técnicas de tolerância a falhas.

QUADRO 4: Fases para aplicação de técnicas de tolerância a falhas

Fases	Técnicas
Detecção de erros	Replicação Testes de limites de tempo Cão de guarda Testes reversos Codificação: paridade, códigos de detecção de erros Testes 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 pela autora

3.3. Tolerância a falhas na computação em nuvem

Quando se trata de tolerância a falha em ambientes de computação em nuvem, é considerada a capacidade de elasticidade do ambiente e o atendimento dos serviços sob demanda. Em casos de instabilidades, sempre haverá a redistribuição dos serviços devido à sua capacidade de elasticidade.

Para que os ambientes em nuvem sejam os tolerantes a falhas, o uso de simuladores nestes ambientes tem se tornado uma realidade cada vez mais próxima, com o uso de diferentes estratégias para a modelagem dos possíveis cenários e modelos matemáticos utilizados para apurar o processo de detecção das falhas em casos de existência de ambiguidade (REZAEIPANAH *et al.*, 2020).

A partir da taxonomia de falhas apresentadas na seção 4.1, para falhas em ambientes de nuvem, é verificada a ocorrências de falhas em casos de *hardware* (falha por omissão), negação de serviço (falha de envelhecimento), além de falhas referentes à resposta, como as falhas de estado, e falhas de tempo de resposta, como as falhas de ocorrências tardias ou precoces que estão relacionadas à comunicação, dados e processos no ambiente de rede (ANDRADE 2019).

Há trabalhos associados à existência de falhas relacionadas à virtualização (ANDRADE, 2019), como as falhas de desenho incorreto (falha diversa), incompatibilidade de protocolos e falhas de interdependência de serviços (falhas de interação) (ARGAWAL, 2011).

Nos Quadros de 5 a 11 são apresentadas técnicas reativas e proativas que podem ser aplicadas em caso de ocorrência de falhas, de acordo com a taxonomia apresentada na seção 4.1 e as melhores técnicas reativas e proativas de tratamento de falhas (ARGAWAL, 2011), (KUMARI, 2015).

Técnica como autocura, refere-se à capacidade do sistema se reestabelecer, como ocorre em casos de falha de omissão de *software*. Já em caso de disco rígido (HD) cheio, a técnica de autocura ocorre quando o sistema verifica o espaço insuficiente no disco rígido (HD) e adverte o usuário ou, quando o armazenamento ocorre em servidores, refaz a sua rota para salvar em outro local.

Já a migração preemptiva e a migração de trabalho, consistem na capacidade do sistema verificar falhas de comunicação, ocasionadas pela negação ser serviço e assim, resubmeter as tarefas para que sejam migradas para outra estação de trabalho.

A técnica do tente novamente e resubmissão de tarefas, consistem nas tentativas de resubmeterem as tarefas, quando houve falhas de comunicação e falhas de negação de serviço.

QUADRO 5: Aplicação de técnicas de tolerância a falhas de omissão

Tipo de falha	**Ocorrência**	**Técnica**
Omissão de *software*	*IaaS*	Autocura
Omissão de *hardware*	*PaaS* e *SaaS*	Resubmissão de tarefa
		Migração

FONTE: Argawal (2011) e Kumari (2015). Traduzido e adaptado pela autora

QUADRO 6: Aplicação de técnicas de tolerância a falhas de envelhecimento

Tipo de falha	**Ocorrência**	**Técnica**
Negação de serviço	*IaaS* e *SaaS*	Migração preemptiva
		Tente novamente
		Migração de trabalho
disco rígido (HD) Cheio	*IaaS*	Autocura

FONTE: Argawal (2015) e Kumari (2015). Traduzido e adaptado pela autora

Devido a esta possibilidade, um ambiente em nuvem pode estar suscetível principalmente a falhas relacionadas à sua infraestrutura. Como por exemplo, falhas relacionadas ao processo de comunicação de pacotes além de falhas relacionadas ao ambiente de *hardware*: como instabilidade ou queda de um ou mais servidores que são as falhas relacionadas à comunicação de acordo com o Quadro 7. (GEBREYOHANNES, 2014; SHAOKA, 2013).

QUADRO 7: Aplicação de técnicas de tolerância a falhas de resposta

Tipo de falha	Ocorrência	Técnica
De valores	*PaaS* e *SaaS*	Tente novamente
De estado	*IaaS*, *PaaS*, *SaaS*	*Checkpoint*
		Replicação
		Resubmissão de tarefa
		Tente novamente
		Autocura
		Balanceamento de carga

FONTE: Argawal (2015) e Kumari (2015). Traduzido e adaptado pela autora

Quando há falhas relacionadas a *software*, é possível a aplicação de técnicas de novas tentativas de resubmissão de tarefas (tente novamente) além da aplicação da técnicas de resgate de carga de trabalho.

QUADRO 8: Aplicação de técnicas de tolerância a falhas de *software*

Tipo de falha	Ocorrência	Técnica
Transiente ou intermitente	*PaaS* e *SaaS*	Tente novamente
		Resgate de carga de trabalho

FONTE: Argawal (2015) e Kumari (2015). Traduzido e adaptado pela autora

QUADRO 9: Aplicação de técnicas de tolerância a falhas de tempo de resposta

Tipo de falha	Ocorrência	Técnica
Precoces	*IaaS*, *PaaS*, *SaaS*	Tente novamente
		Resgate da carga de trabalho
Tardia	*PaaS* e *SaaS*	Tente novamente
		Resgate de tarefas
		S Guarda

FONTE: Argawal (2015) e Kumari (2015). Traduzido e adaptado pela autora

Outras possibilidades de falhas podem estar relacionadas à comunicação, aos dados, aos processos, a ou virtualização ou a interação entre as tarefas de acordo com o Quadro 10 apresentado (ANDRADE, 2019).

QUADRO 10: Aplicação de técnicas de tolerância a falhas de interação

Tipo de falha	Ocorrência	Técnica
Sobrecarga do tempo	*PaaS* e *SaaS*	Migração de trabalho Resubmissão de tarefa Replicação Migração preemptiva Balanceamento de carga
Interdependências entre Serviços	*PaaS* e *SaaS*	Resubmissão de tarefa Migração Migração de trabalho S Guarda Resubmissão de tarefa Replicação Balanceamento de carga Autocura
Incompatibilidade de protocolos	*IaaS*, *PaaS* e *SaaS*	Replicação Migração preemptiva

FONTE: Argawal (2015). Traduzido e adaptado pela autora

No Quadro 11 são apresentadas outras falhas que podem ocorrer devido ao desenho incorreto da classe de serviço ou falhas permanentes que podem existir nas classes de Serviço *PaaS* e *SaaS*.

QUADRO 11: Aplicação de técnicas de tolerância a falhas de diversas

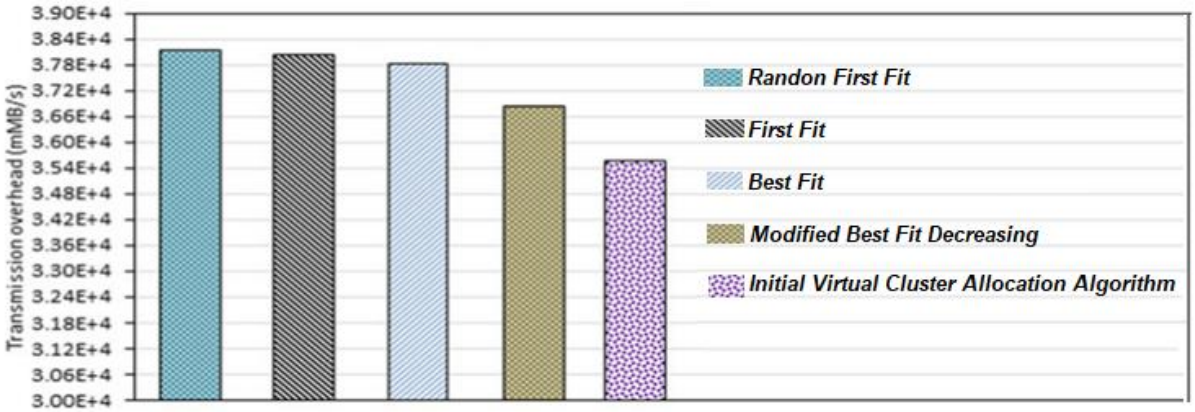
Tipo de falha	Ocorrência	Técnica
Falhas permanentes	*PaaS* e *SaaS*	Replicação *Checkpoint* Tente novamente Resubmissão de tarefa
Desenho incorreto	*IaaS*, *PaaS* e *SaaS*	Replicação *Checkpoint* Resgate de carga de trabalho Balanceamento de carga

FONTE: Argawal (2015). Traduzido e adaptado pela autora

Nos estudos realizados, foi verificado que as falhas relacionadas à comunicação e *hardware* são as que mais ocorrem, superando 50% dos casos de falhas em ambientes de nuvem. Para tanto, os sistemas de tolerância a falhas têm focado em técnicas reativas para seus tratamentos, de forma que as falhas sejam identificadas a partir de mecanismos de ponto de verificação e reinicialização.

Quando há a comparação da ocorrência de falhas relacionadas à comunicação e *hardware* em relação às demais, é observado que parte da ocorrência da falha de comunicação está relacionada à sobrecarga de trabalho, de acordo com o gráfico apresentado na Figura 4. (LIU; WANG; ZHOU; KUMAR; YANG; BUYYA , 2018).

FIGURA 4: Gráfico de comparação dos algoritmos relacionados à sobrecarga de trabalho

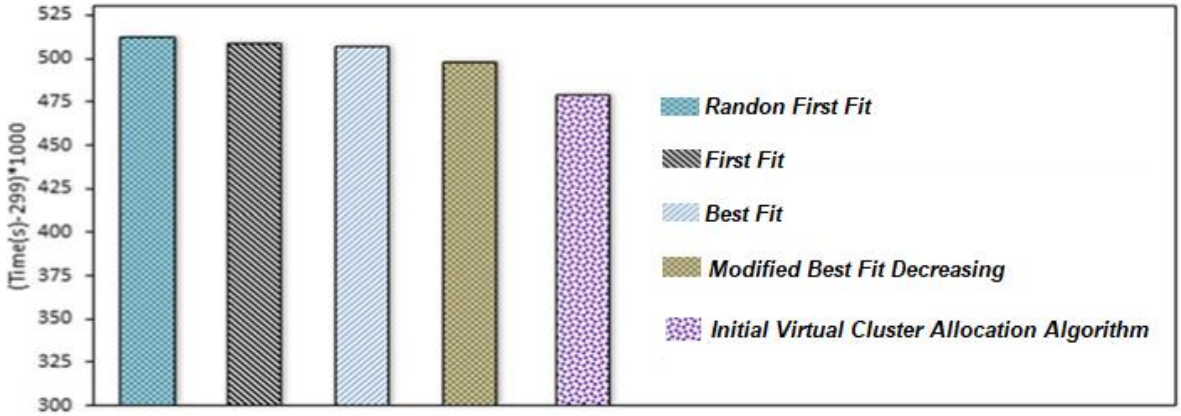


FONTE: LIU; WANG; ZHOU; KUMAR; YANG; BUYYA , 2018

Os estudos relataram que, se os aplicativos em nuvem são dinâmicos, e, devido a sua elasticidade e seu grande volume de fluxo de dados e atualizações, como em casos de redes sociais, as técnicas reativas não refletem um bom desempenho (LIU; WANG; ZHOU; KUMAR; YANG; BUYYA , 2018).

Em casos específicos, as técnicas proativas de tratamento de falhas tem se tornado mais eficientes, pois visam a antecipação das falhas, como por exemplo, a deterioração do estado de saúde do *software*, o monitoramento da velocidade dos ventiladores dos servidores, a temperatura da CPU, entre outros, que a partir do monitoramento constante, pôde ser observado os melhores resultados, conforme apresentado na Figura 5, em que no gráfico é apresentado a diminuição da ocorrência de algns tipos de falhas, a partir da aplicação de técnicas. (LIU; WANG; ZHOU; KUMAR; YANG; BUYYA , 2018).

FIGURA 5: Gráfico de comparação dos algoritmos relacionados à eficiência do tempo



FONTE: LIU; WANG; ZHOU; KUMAR; YANG; BUYYA , 2018

Para que o ambiente de computação em nuvem possa ter tolerante a falha, ele pode conter técnicas tolerantes a falhas de acordo com sua capacidade e sua demanda. Inicialmente, é essencial a detecção de erros, avaliação dos danos, recuperação do erro, para posterior tratamento e continuidade do serviço (ANDERSON *et al.*, 1981).

Além do ambiente ser tolerante a falha, ele também poderá ser estruturado de forma que seja robusto, confiável, adaptável, disponível, possuir uma boa taxa de transferência e de tempo de resposta.

Uma das técnicas mais utilizadas para a detecção de falhas em ambientes de computação em nuvem se refere a criação de pontos de verificação, ou seja, a aplicação da técnica de *checkpoint,* em que são executados de forma paralela, como forma de complementar a técnica de tolerância a falha reativa. (RAMAMOHANA-RAO *et al.,* 2014).

O planejamento da inserção de pontos de verificação auxilia no processo de mapeamento do fluxo de trabalho das tarefas e dependência entre elas. A utilização desta técnica tem se mostrado menos custosa, com economia de até 70% em condições relaxadas e bem planejadas. (RAMAMOHANARAO *et al.,* 2014).

Para os casos de planejamento de inserção de pontos de verificação, a economia do custo computacional chega a 14% dos casos, de acordo com o gráfico apresentado na Figura 6 (RAMAMOHANARAO *et al.,* 2014).

Na Figura 6 é apresentado um gráfico elaborado pelos autores Ramaohanarao *et al.* (2014), em que foram realizadas análises comparativas referentes ao uso da técnica de tratamento de falhas, de inserção de pontos de verificação, ou *checkpoints*.

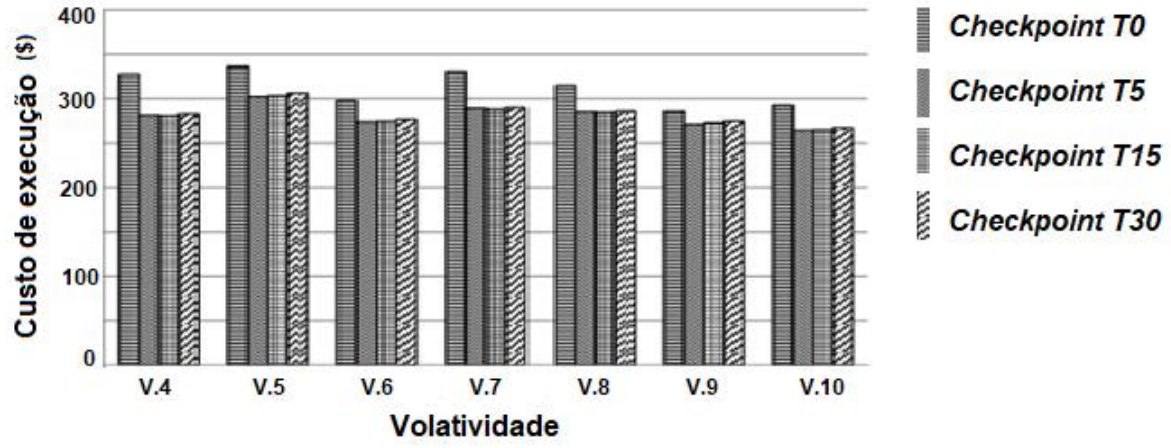
Na oportunidade, os autores inseriram pontos de verificação em diferentes períodos e estão presentes no gráfico apresentado à Figura 5, em que há a comparação da eficiência do tempo nos algoritmos relacionados e seu respectivo custo computacional.

O CHKPT5 inseriria pontos de verificação a cada cinco segundos, o que resultou nos sete grupos de testes, um menor custo computacional, mas com melhor resultado apresentado no 7º grupo de testes.

O CHKTP15 inseriria pontos de verificação no intervalo de 15 segundos e o CHKTP30 no intervalo de 30 segundos. Ambos apresentaram um custo computacional maior do que o ponto de verificação CHKTP5, em virtude da possibilidade de haver um número maior de informações a serem gravadas neste ponto de verificação.

Na Figura 6 é possível observar os grupos de testes e os comparativos representados por cada um dos pontos de verificação inseridos nos grupos.

FIGURA 6: Economia de recursos computacionais com o tratamento de falhas



FONTE:Adaptado de RAMAMOHANARAO; BUYYA, 2014

Outras técnicas empregadas para ambientes de computação em nuvem são técnicas de migração de trabalho, migração preemptiva, replicação - principalmente para ambientes que possuem um grande volume de dados. Porém, em ambientes com pouca ou nenhuma interferência humana, é recomendada técnica de autocura. (KUMARI, 2018), pois, de acordo com os trabalhos apresentados, o sistema busca tratar a suas falhas automaticamente.

As técnicas apresentadas para o tratamento da ocorrência de falhas são baseadas nos parâmetros de tratamento de falhas proposto por Kumari *et al*. (2018), apresentado no Quadro 12, de forma que há a interação entre as fases de detecção,

avaliação e contingência da ocorrência dessas falhas, e, caso venham a ocorrer simular formas de minimizar os prejuízos sofridos pelos sistemas.

Em sistemas tolerantes a falhas é considerado o seu ciclo de detecção, classificação ou avaliação e tratamento ou contingência da ocorrência da falha, de forma que posso minimizar o prejuízo das falhas sofridas.

Para compreender se o sistema que está em um ambiente em nuvem é tolerante a falha, é realizado um levantamento de atendimento de conformidade de parâmetros, em que é verificado se o sistema é adaptável, se possui uma boa taxa de desempenho de resposta, sua taxa de transferência, sua disponibilidade, sua confiabilidade, sua usabilidade e sua sobrecarga associada (KUMARI, *et al.*, 2018).

No Quadro 12 é apresentada a descrição de cada parâmetro de tolerância a falhas em ambientes de computação em nuvens KUMARI *et. al* (2018).

QUADRO 12: Parâmetros para tolerância a falhas e sua descrição em computação em nuvem

Parâmetros	Descrição
Adaptável	Todo o processo é automaticamente executado de acordo com suas condições
Desempenho	Usada para garantir a eficiência do sistema
Tempo de resposta	Tempo total usado para responder a um algoritmo específico
Taxa de transferência	O número de tarefas computacionais implementadas e completadas com sucesso
Confiabilidade	Fornecer resultados precisos ou aceitáveis em um determinado período de tempo
Disponibilidade	A probabilidade de o sistema estar funcionando corretamente após ser solicitado / destinado a uso
Usabilidade	Capacidade de o sistema estar tolerante a falhas, e, ao mesmo tempo ser fácil de usar e intuitivo ao usuário
Sobrecarga associada	Determina o total de sobrecarga envolvida enquanto executa o algoritmo de tolerância a falhas

Fonte: Kumari *et al*. (2018), traduzido e adaptado pela autora

Quanto à métrica empregada em tolerância a falhas em computação em nuvem, é realizada a avaliação de parâmetros (PATRA; SINGH; SINGH, 2013) tais como a taxa de transferência (ou *throughput),* tempo de resposta, escalabilidade, disponibilidade, usabilidade e sobrecarga associada.

3.4. Ocorrência de falhas em gestores de nuvem

Nos gestores de nuvem são permitidas ferramentas que auxiliam no seu gerenciamento. Estes, não tornam o ambiente protegido e ileso de qualquer tipo de ocorrência de falha.

Em relação às principais ocorrências de falhas em gestores de nuvem, foram pesquisados trabalhos relacionados a ocorrência de falhas relacionadas a dados, comunicação, processos e virtualização dos gestores de nuvem *Eucalyptus, OpenStack, Open Nebula* e *CloudStack.*

Dos gestores de nuvem relacionados, apenas o *CloudStack* possui tratamento para os recursos de rede. Os demais, possuem pelo menos um recurso de tratamento de falha na classe de serviço *IaaS* (VOGEL; GRIEBLER; MARON; SCHEPKE; FERNANDES, 2016)

O gestor de nuvem *Eucalyptus* possui código aberto para a criação de nuvens privadas e híbridas compatíveis com os serviços de computação em nuvem oferecidos pela Amazon EC2, a AWS. Nele, é possibilitado que seus usuários provisionem seus recursos de computação e armazenamento sob demanda. Com o foco no armazenamento, ele não possui a técnica de autocura implementada em caso de ocorrência de falhas e nem técnicas tolerância falhas relacionadas à comunicação, dados e virtualização quando se trata do monitoramento de Máquinas Virtuais (SOMASHEKHAR; VIKAS MAHESHWARI; SINGH, 2019).

Em outros trabalhos foram relatados que, quando o gestor de nuvem *Eucalyptus* é exposto em testes que medem a disponibilidade, elasticidade e sistemas redundantes, foi apresentado que uma simples replicação no componente crítico pode aumentar a disponibilidade do sistema, reduzindo o tempo de inatividade, que, em proporção significativa (mais de 99% de redução), mesmo com a adição de clusters, não houve significativa melhoria. (DANTAS; ARAÚJO; MATOS; MACIEL, 2015). Devido a apenas este ponto de falha, é sugerido pelos pesquisadores o uso de técnicas de redundância de dados - ou replicação - por se tratar de grandes volumes de dados.

Na literatura apresentada para o gestor de nuvem *OpenStack* foi verificado foco para a gestão da classe de *IaaS*, porém com ocorrência de falhas relacionadas a dados e com a possível incompatibilidade de protocolos, como a *“perda de pacotes”* (VOGEL *et al.* , 2016).

Quanto a avaliação de implantação e desempenho de redes virtuais baseadas em no *OpenStack*, há possíveis falhas documentadas em relação à comunicação, processos e virtualização. Com ocorrência, em casos de implantação de *host* único, ou seja, um único nó de rede e vários nós de computação implementados, além de ocorrências quando houver alto tráfego de rede (SHAOKA, 2013).

No caso do *OpenStack*, as técnicas recomendadas para a tolerância de falhas referem-se à inserção de técnicas *checkpoint* e replicação (KULKARNI; BHOSALE, 2016) para casos de aumento de tráfego. Quanto às falhas relacionadas à comunicação, as técnicas de migração de trabalho e migração preemptiva podem ser adequadas (KULKARNI; BHOSALE, 2016).

No gestor de nuvem *Open Nebula* são oferecidas soluções com foco em recursos e de alta flexibilidade, para o gerenciamento de *data centers* virtualizados para permitir sua aplicação em nuvens privadas, públicas e híbridas (OPENNEBULA, 2017).

Para o *Open Nebula,* as principais sugestões de tolerância a falhas são recomendadas a aplicação de técnicas de rejuvenescimento de *software* para casos em que há evidências de seu envelhecimento, sendo possível identificá-lo caso haja métodos de geram a sobrecarga de trabalho (TORQUATO; MACIEL; ARAÚJO; UMESH, 2017).

Outra técnica implementada neste gestor refere-se a migração de trabalho, que ocorre antes que haja o rejuvenescimento de *software* como uma de suas estratégias (TORQUATO; MACIEL; ARAÚJO; UMESH, 2017).

Nos gestores de nuvens pesquisados, foi verificado que há maior incidência de ocorrência de falhas relacionados a Omissão de *Hardware* e *Software*, falhas de dados, entre elas: incompatibilidade de protocolos, provisionamento de recursos, perda de pacotes, perda de comunicação e balanceamento da rede. São relatadas outras ocorrências de falhas relacionadas ao poder computacional, Disco Rígido cheio e perdas de armazenamento.

No Quadro 13 são apresentadas as técnicas de tratamento de falhas aplicadas em casos de ocorrência de falhas de processo.

QUADRO 13: Cenários de ocorrência de falhas de processos e técnicas mais eficazes

Cenário da ocorrência da falha	**Técnica eficaz**
Falha de dados com provisionamento de recursos	*Checkpoint* Migração de trabalho
Falha de processo e dados com provisionamento de recursos	Autocura
Falha em dados	Autocura
Foco na disponibilidade de dados	*Checkpoint*
Perda de pacotes	*Checkpoint* Migração de trabalho
Perda de armazenamento	Migraçao de trabalho
Perda de comunicação e balanceamento da rede	Replicação *Checkpoint* Migração de trabalho Autocura
Melhoria de espaço de armazenamento	Replicação
Congestionamento de rede	Replicação
E/S mais rápidas / descentralizado	Replicação
Falhas relacionadas ao sistema operacional	Replicação
Falhas relacionadas às Máquinas Virtuais	*Checkpoint* Migração de trabalho Autocura Replicação

FONTE: Da própria autora (2020)

No Quadro 14 são apresentadas as técnicas de tratamento de falhas aplicadas em casos de ocorrência de falhas de comunicação nos gestores de nuvens pesquisados. Nele, 14 são visualizados alguns cenários em que há possibilidade de ocorrência de falhas, e, caso ocorram as técnicas de tratamento de falhas mais eficazes para serem aplicados. Para a seleção das possíveis técnicas a serem aplicadas em cada cenário foi levada em consideração a possibilidade do universo de dados ser grande, além das chances de existência de falhas relacionadas ao sistema operacional, Máquinas Virtuais e físicas existentes, bem como os demais serviços de rede e deperiféricos relacionados.

QUADRO 14: Cenários de ocorrência de falhas de comunicação e técnicas mais eficazes

Cenário da ocorrência da falha	**Técnica eficaz**
Falha de dados com provisionamento de recursos	*Checkpoint*
	Autocura
	Replicação
Falha de processo e dados com provisionamento de recursos	Replicação Autocura
Falha em dados	Replicação Autocura
Foco na disponibilidade de dados	Replicação *Checkpoint*
Perda de pacotes	*Checkpoint* Migração preemptiva Autocura
Perda de armazenamento	Replicação
Perda de comunicação e balanceamento da rede	Replicação *Checkpoint* Migração preemptiva Autocura
Melhoria de espaço de armazenamento	Replicação
Perda de armazenamento	Replicação
Pequena Interrupção na rede	*Checkpoint* Autocura
E/S mais rápidas ou descentralizado	Autocura
Falhas relacionadas a periféricos da rede	*Checkpoint*
Falhas relacionadas serviços da rede	*Checkpoint* Autocura
Falhas relacionadas ao sistema operacional	*Checkpoint*
Falhas relacionadas às Máquinas Virtuais	Replicação Autocura

FONTE: Da própria autora (2020)

No Quadro 15 são apresentadas as possíveis técnicas de tratamento de falhas aplicadas em casos de ocorrência de falhas de dados nos gestores de nuvens pesquisados, em que foi considerado a abrangência dos gestores de nuvem, assim como a sua capacidade em gerir nuvens privadas, públicas e híbridas.

QUADRO 15: Cenários de ocorrência de falhas de dados e técnicas mais eficazes

Cenário da ocorrência da falha	**Técnica eficaz**
Interrupção do funcionamento de uma máquina virtual	Migração de trabalho
Falha de dados com provisionamento de recursos	*Checkpoint*
Falha de processo e dados com provisionamento de recursos	Migração preemptiva
	Replicação
Falha em dados	Autocura
	Replicação
Falha de dados e processo no comparativo de opções de armazenamento/entrega de avaliação de riscos	Migração de trabalho
Foco na Qualidade sobre Serviços (QoS)	*Checkpoint*
Foco na disponibilidade de dados	Replicação
Perda de armazenamento	Replicação
Melhoria de espaço de armazenamento	Replicação
Congestionamento na rede	Autocura
Evitar interrupções no sistema	Replicação
	Migração preemptiva
	Autocura
Pequena Interrupção na rede	Autocura
E/S mais rápidas ou descentralizado	Replicação
Falhas relacionadas a periféricos da rede	Replicação
Falhas relacionadas à queda de energia	Replicação
Falhas relacionadas a armazenamento	Replicação
Falhas relacionadas à memória	Replicação
Falhas relacionadas serviços da rede	Autocura
Falhas relacionadas ao sistema operacional	Replicação
Falhas relacionadas às Máquinas Virtuais	Replicação

FONTE: Da própria autora (2020)

Assim como no Quadro 15, no Quadro 16 são apresentadas as técnicas de tratamento de falhas aplicadas em casos de ocorrência de falhas de virtuailização nos gestores de nuvens pesquisados, em que foi considerado a possibilidade de falhas relacionadas a *hardware* e a *software,* como falhas relacionadas ao sistema operacional e às Máquinas Virtuais.

QUADRO 16: Cenários de ocorrência de falhas de virtualização e técnicas mais eficazes

Cenário da ocorrência da falha	**Técnica eficaz**
Falha de dados com provisionamento de recursos	Replicação Autocura *Checkpoint*
Falhas em dados	Autocura
Perda de armazenamento	Replicação
Melhoria de espaço de armazenamento	Replicação
Congestionamento na rede	Autocura Migração preemptiva
Evitar interrupções no sistema	Replicação Autocura Migração preemptiva
Pequena Interrupção na rede	Replicação Autocura Migração preemptiva
E/S mais rápidas / descentralizado	Replicação Autocura Migração preemptiva
Falhas relacionadas a periféricos da rede	Replicação Autocura Migração preemptiva
Falhas relacionadas à queda de energia	Replicação Migração preemptiva
Falhas relacionadas a armazenamento	Replicação Autocura
Falhas relacionadas à memória	Replicação
Falhas relacionadas serviços da rede	Replicação
Falhas relacionadas ao sistema operacional	Replicação *Checkpoint* Migração preemptiva
Falhas relacionadas às Máquinas Virtuais	Replicação Autocura *Checkpoint*

FONTE: Da própria autora (2020)

3.5. Tolerância a falhas em simuladores em computação em nuvem

O uso de simuladores para ambientes de computação em nuvem tem colaborado na previsibilidade da ocorrência de possíveis falhas (AZIZ *et al.*, 2018).

Nos ambientes em que são utilizados simuladores têm gerado a economia de recursos financeiros das organizações, como também maiores acertos na hora da

execução de projetos, contribuindo significativamente para o sucesso das organizações.

O simulador torna-se ainda mais robusto se possui além da simulação do ambiente, a simulação de falhas e seu tratamento, para que os usuários possam identificar como a arquitetura se comportaria em caso de ocorrência de falhas e a aplicação das possíveis técnicas para seu tratamento.

Desta forma, o simulador legado iSPD, foi utilizado como objeto do estudo, pois permite a realização de simulação de ambientes computacionais nas arquiteturas de grade e de nuvem, neste, para os serviços *IaaS* e *PaaS,* sendo um simulador de eventos discretos voltado para a simulação de sistemas (MANACERO *et al.*, 2012).

Desenvolvido em Linguagem Java JSE, o iSPD oferece interface icônica para facilitar o uso da ferramenta e aumento da usabilidade da mesma. Nele, pode ser configurada a quantidade de máquinas físicas, Máquinas Virtuais, cargas de trabalho e a configuração do ambiente de comunicação e de rede para que seja o mais próximo possível de um ambiente em nuvem real.

3.6. Considerações finais

Neste capítulo foi apresentada a taxonomia de falhas, principais técnicas empregadas em caso de ocorrência de falhas e, em específico, falhas em computação em nuvens.

No próximo capítulo serão apresentadas as etapas percorridas para o desenvolvimento de o gerador de falhas, as atualizações implementadas no ISPD e sua atualização de versão.

CAPÍTULO 4 DESENVOLVIMENTO DE UM GERADOR DE FALHAS

Neste capítulo são apresentados o simulador existente, o iSPD, a modelagem de falhas em nuvem, os processos desenvolvidos na implementação do sistema injetor de falhas, a biblioteca de falhas e os respectivos testes realizados.

O método utilizado para o levantamento das informações foi a partir de revisão sistemática da literatura com a finalidade de levantamento das técnicas reativas e proativas, empregadas no tratamento de falhas que ocorrem ambiente de computação em nuvem de *IaaS*. (KANSO; DEIXIONNE; GHERBI; MOGHADDAM, 2017).

Neste trabalho, a técnica de tratamento de falha utilizada para o tratamento de falhas de omissão de *hardware* foi a técnica de tratamento proativo de autocura, (ARGAWAL *et al.*, 2015). Na aplicação desta técnica, é realizado na simulação o reescalonamento das máquinas físicas, representadas pela sigla *PM* (Physical Machine).

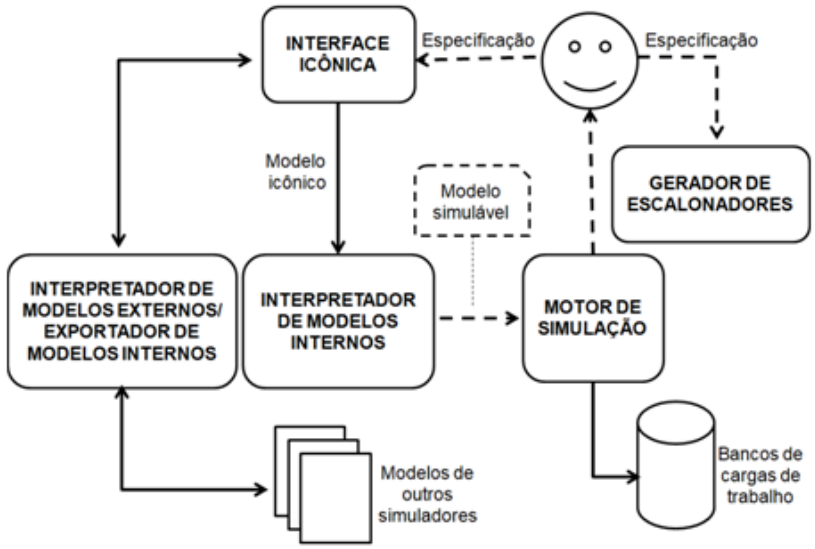
4.1. Apresentação do iSPD: visão geral e interface icônica

Para que a simulação ocorra, é necessário que o usuário represente iconicamente o ambiente de simulação e determine os algoritmos de escalonamento e a configuração de cada máquina virtual alocada.

Na Figura 7 é apresentado o Diagrama Conceitual do iSPD existente antes do desenvolvimento do gerador de falhas. Nele, está representado a integração entre os módulos que são utilizados para a realização de simulações, tais como o interpretador de modelos internos, o interpretador de modelos externos e exportador de modelos internos e o motor de simulação (SILVA, 2015).

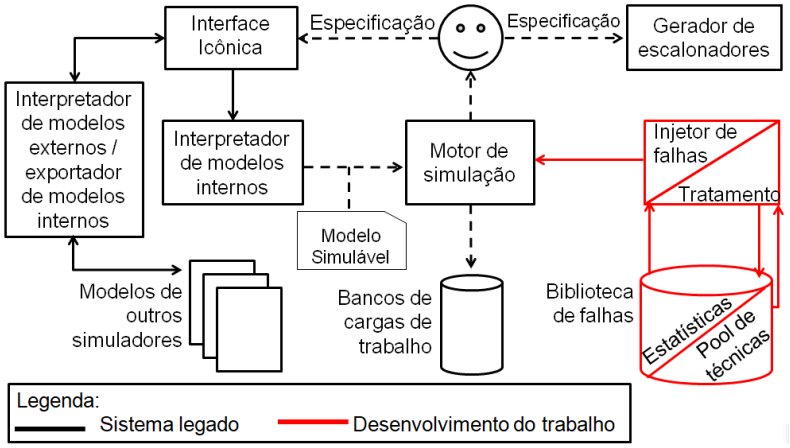
Na Figura 8 é apresentado o Diagrama Conceitual do iSPD após as alterações com o gerador de falhas e a biblioteca de tratamento de falhas.

FIGURA 7: Diagrama conceitual do iSPD



FONTE: Diagrama Conceitual do iSPD, adaptado de Silva (2015)

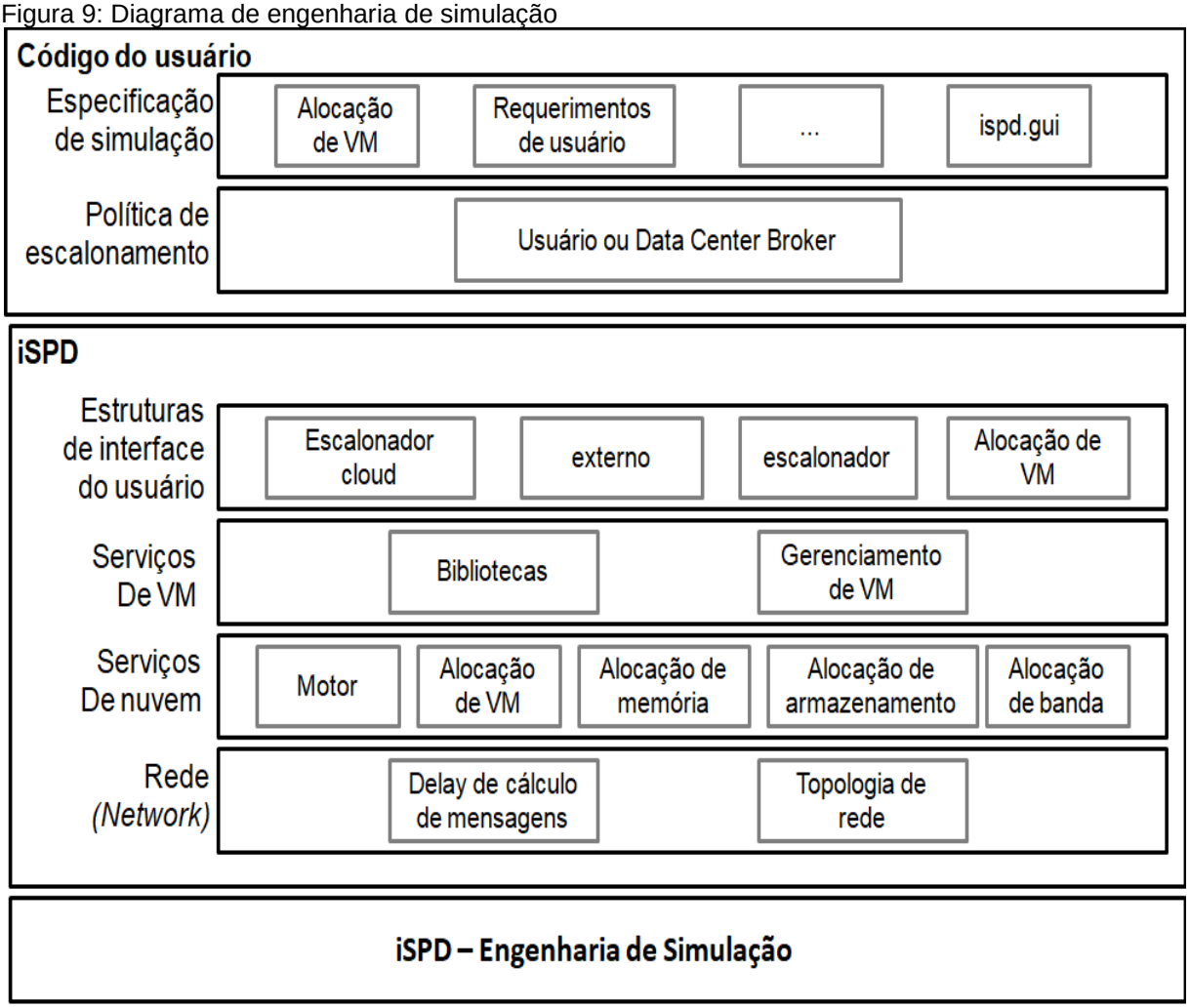
FIGURA 8: Diagrama conceitual do iSPD com alterações



FONTE: TAMASHIRO et al. 2020

Por ser um sistema legado, foi necessário rever a engenharia de simulação existente (SILVA; LOBATO; MANACERO, 2015) e a elaboração do Diagrama da engenharia de simulação apresentado na Figura 9. O simulador é composto pela interação e integração entre as estruturas de interface de usuário, serviços de Máquinas Virtuais, serviços de nuvens e de rede, assim como estes mecanismos são utilizados em outros simuladores. A interação com o usuário ocorre no evento de criação do ambiente de simulação, com a especificação da simulação, configuração das cargas de trabalho e a seleção do algoritmo de escalonamento.

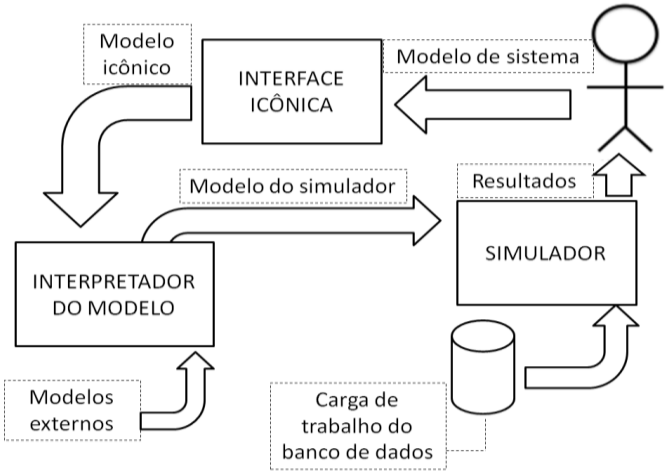
Figura 9: Diagrama de engenharia de simulação



Fonte: CALHEIROS *et al.* (2009), adaptado pela autora

A estrutura do iSPD é baseada na integração do conjunto de três módulos: o modelo icônico ou interface icônica, a linguagem interpretada e o motor de simulação, apresentado na Figura 10.

FIGURA 10: Estrutura básica de funcionamento do iSPD



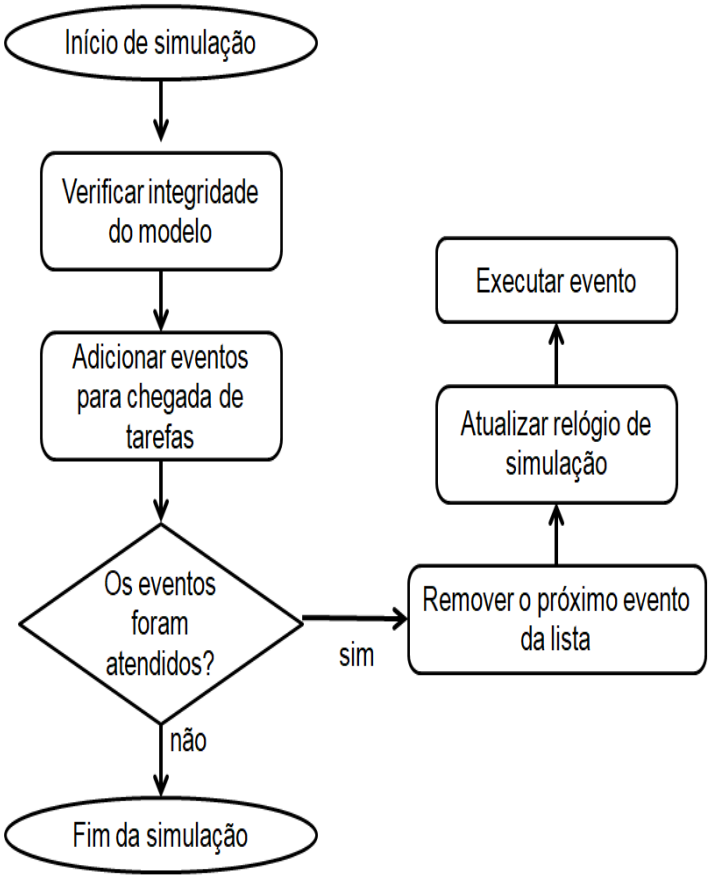
FONTE: MANACERO, A. *et al*.. Adaptado (2012)

Para o funcionamento do simulador é necessário que o usuário escolha o tipo de simulação de computação em nuvem que deseja fazer: para *IaaS* e para *PaaS*. Com a definição de simulação feita, é possível, graficamente, adicionar a quantidade de máquinas que estão sendo usadas, e definir, qual será a “máquina mestre” e quais serão as “máquinas escravas” – todas contendo nós individuais e *links* de comunicação ponto-a-ponto para simular uma *Local Area Network* (LAN) e pode também simular conexão com a Internet.

Para iniciar a simulação é necessária a configuração da rede e a definição dos parâmetros que serão simulados, tais como: largura de banda, latência, tipo de algoritmo de escalonamento entre outros. O usuário, ao projetar o modelo icônico, configura a interpretação do modelo proposto para que seja simulado.

Na Figura 11 é apresentado o fluxograma do processo de simulação, em que é exibido o processo de validação da simulação, a ocorrência da simulação, e a realização de adição de tarefas à simulação, representado pela palavra evento.

FIGURA 11: Fluxograma do processo de simulação

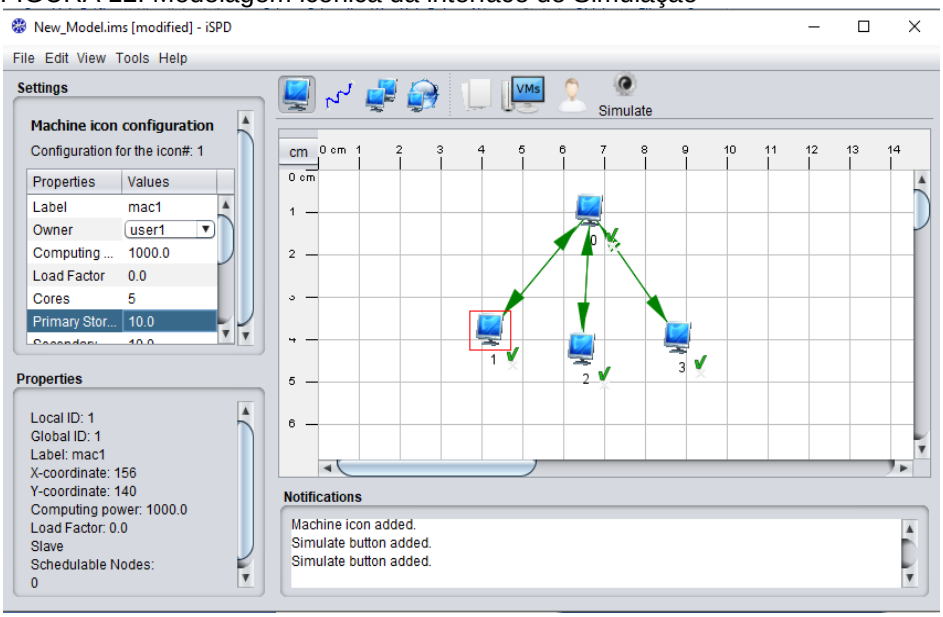


FONTE: MANACERO, A.; *et al.*. Adaptado (2012)

É considerado que, após a simulação ser finalizada, os resultados gerados são exibidos em um relatório para que o usuário verifique o tempo de simulação, a distribuição das tarefas nas Máquinas Virtuais alocadas e também a eficácia do algoritmo de escalonamento selecionado.

Na Figura 12 é apresentada a interface icônica antes da integração com a injeção de falhas e o gerador de falhas. Nela, é possível verificar o desenho de uma simulação e a configuração do poder computacional envolvido.

FIGURA 12: Modelagem icônica da interface de Simulação



FONTE: Modelagem icônica de um modelo de simulação simples do iSPD (2018)

Os resultados apresentados pelo relatório do iSPD são relativos às métricas de desempenho, as quais são fornecidas pelo iSPD, e são elas: os tempos médios de espera e de resposta, eficiência do sistema, e por último a satisfação do cliente (usabilidade e experiência de usuário) MANACERO *et al.* (2012).

4.2. Algoritmos de escalonamento existentes no iSPD

Para o ambiente de simulação *IaaS*, o simulador possui diversos algoritmos implementados, mas, somente para os algoritmos de escalonamento *First Fit e First Fit Decreasing* estão implementadas técnicas de tolerância a falhas, com verificação de configuração do ambiente de simulação e verificação se alguma máquina física ou virtual sofreu desligamento durante o processo de simulação de forma involuntária.

No Quadro 17 são apresentados os demais algoritmos implementados até o momento e sua respectiva localização no iSPD. Neste levantamento estão representados as classes identificadas e uso de técnicas de engenharia reversa para a sua manutenção.

QUADRO 17 – Algoritmos implementados no iSPD

Algoritmo	Localização
Round Robin	gspd.ispd.externo.cloudAlloc
First Fit	gspd.ispd.externo.cloudAlloc
First Fit Decreasing	gspd.ispd.externo.cloudAlloc
Volume	gspd.ispd.externo.cloudAlloc
DynamicFPLTF	gspd.ispd.externo
EHOSEP	gspd.ispd.externo
HOSEP	gspd.ispd.externo
M_OSEP	gspd.ispd.externo
RoundRobin	gspd.ispd.externo
WQR	gspd.ispd.externo
Workqueue	gspd.ispd.externo

FONTE: Da própria autora (2020)

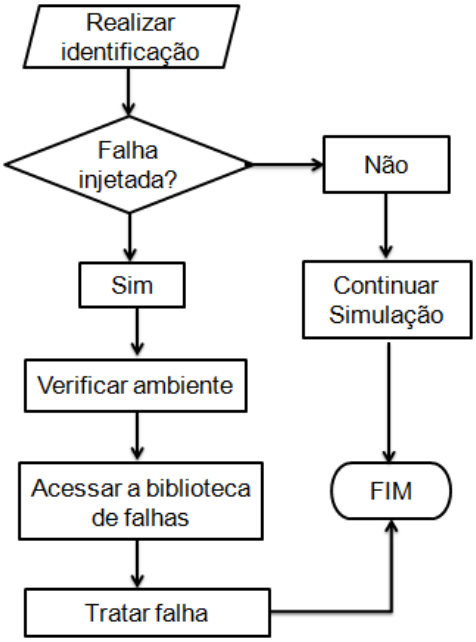
4.3. Modelagem de falhas no iSPD

O desenvolvimento de interface icônica aumenta a usabilidade e a experiência do usuário no simulador, sendo possibilitado a ele a seleção das falhas para serem injetadas no ambiente de simulação a partir da interação com a interface icônica JSelecionarFalhas.java.

Durante o processo de simulação, as falhas são inseridas pelo sistema injetor de falhas, detectadas pela biblioteca de falhas. Quando a falha é detectada, é empregado técnicas de tratameto proativas ou reativas. A técnica empregada varia de acordo com o ambiente em que a falha ocorreu.

Na Figura 13 é apresentado o fluxograma do processo de identificação de ocorrência de falha. Caso não seja identificado nenhum tipo de injeção de falha, há a continuidade da simulação.

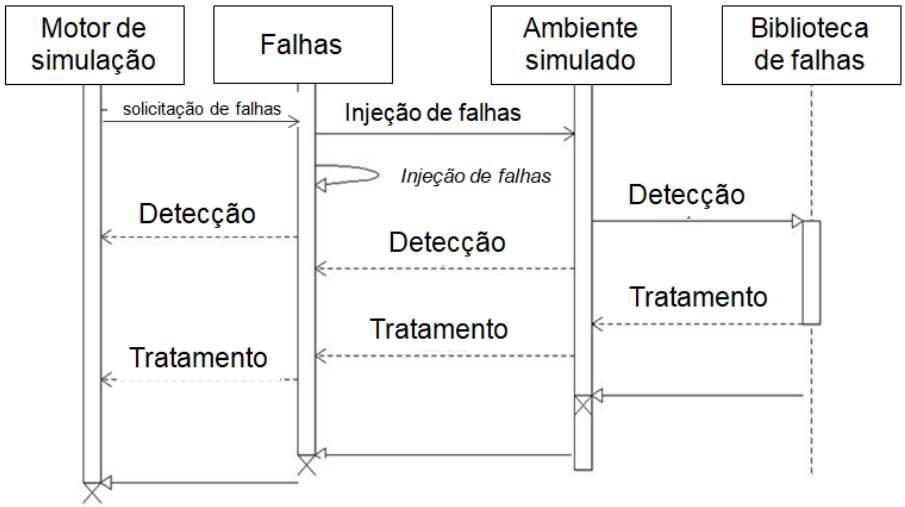
FIGURA 13: Fluxograma quanto à identificação de falhas



FONTE: Da própria autora (2020)

No Diagrama de sequência apresentado na Figura 14, é ilustrado o processo de simulação com a injeção de falhas, sua detecção e a aplicação de seu tratamento.

FIGURA 14: Diagrama de sequência da detecção e tratamento das falhas



FONTE: TAMASHIRO *et al.* (2020)

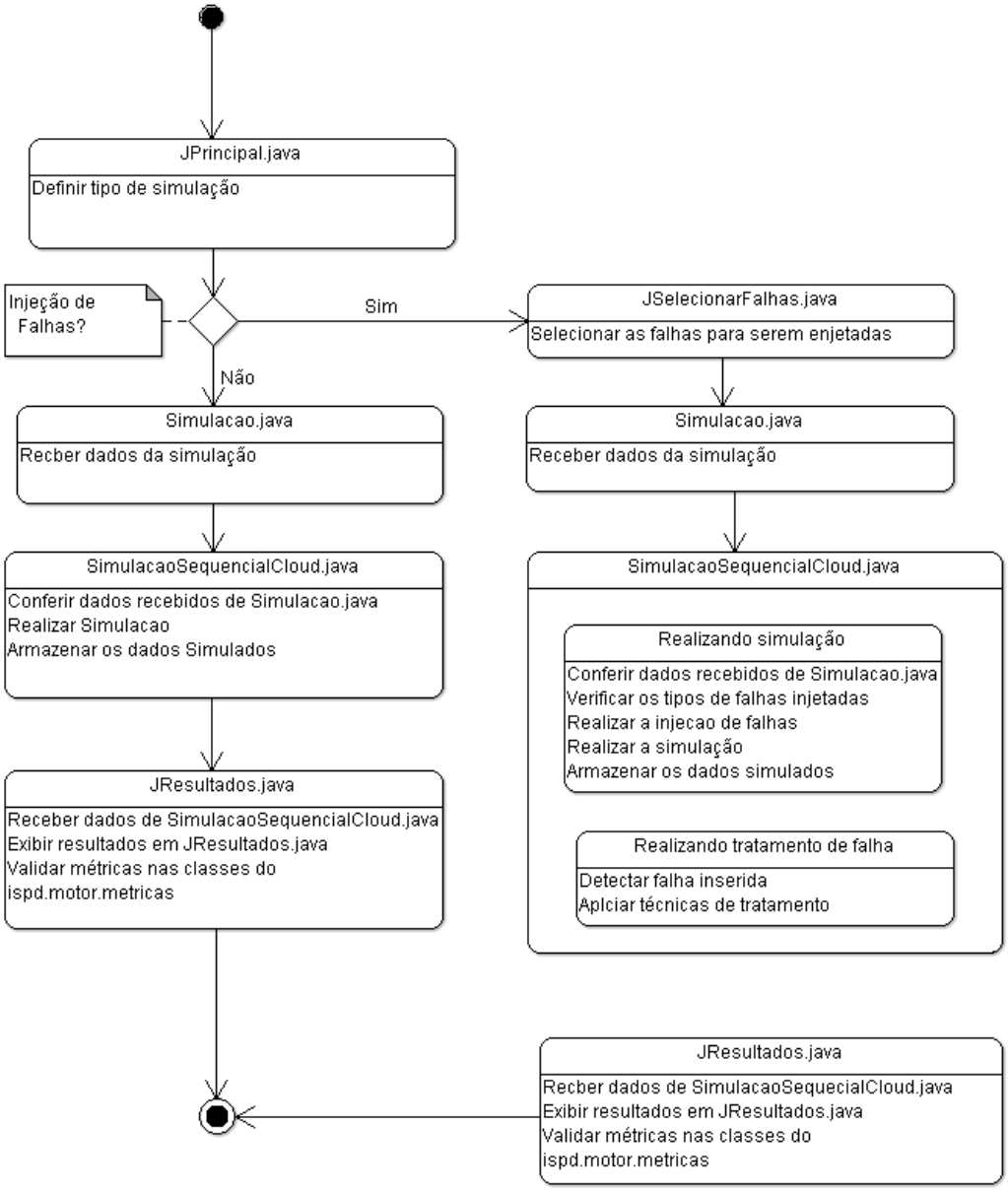
4.4. Implementação – Discussão dos resultados

No processo de desenvolvimento da modelagem de falhas foram considerados os pacotes e classes já existentes no iSPD, com destaque para o

pacote motor, que contém as classes relacionadas aos processos de modelagens de simulação.

No processo do planejamento da injeção de falhas para que houvesse a detecção e o tratamento das falhas no ambiente simulado, houve a necessidade dos eventos inseridos no sistema, de forma que o processo de injeção de falha gerasse um estímulo para que pudesse ser detectado, de acordo com o Diagrama de Máquina de Estados apresentado na Figura 15.

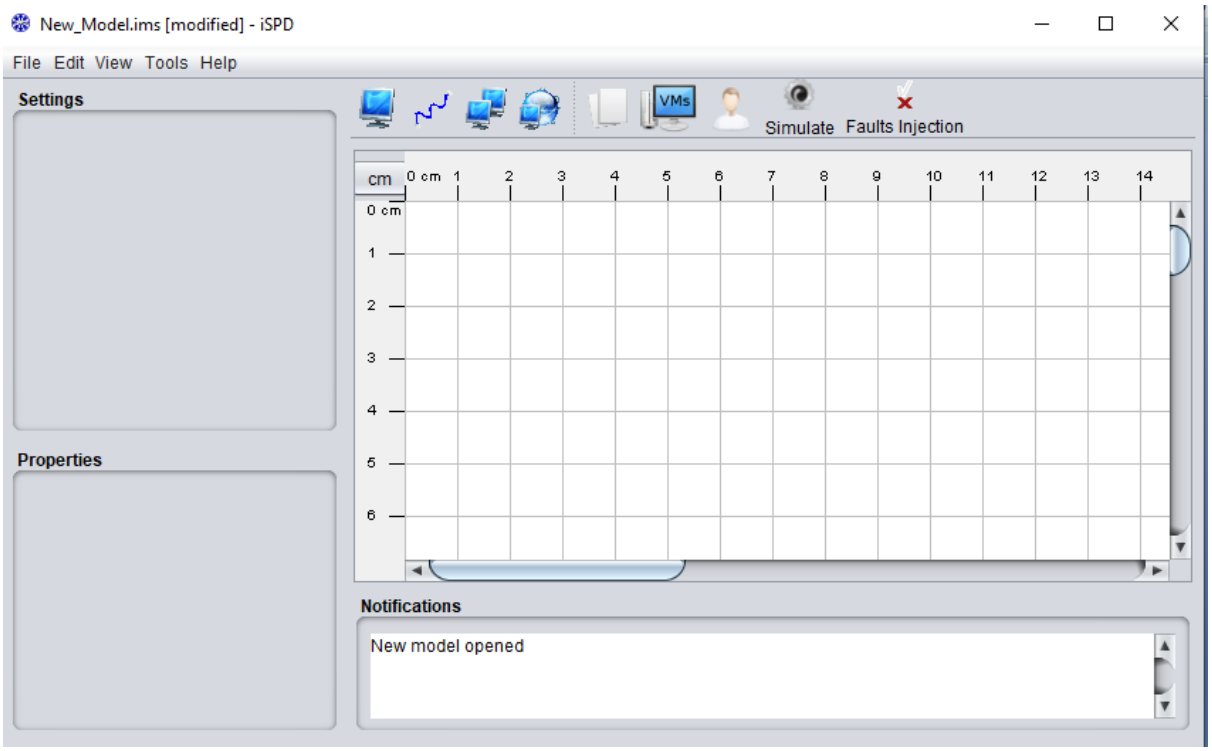
FIGURA 15: Diagrama de Máquina de Estados da modelagem das falhas



FONTE: Da própria autora (2020)

Na Figura 16 é apresentada a atualização na interface icônica principal (JPrincipal.java). Como parte do desenvolvimento do trabalho, nesta interface icônica, foi inserido um botão de injeção de falhas que conecta o usuário à interface icônica implementada no iSPD para a seleção das falhas para serem injetadas, nomeada de JSelecionarFalhas.java.

FIGURA 16: interface icônica Principal.java



FONTE: Print screen da interface icônica JPrincipal.java no iSPD

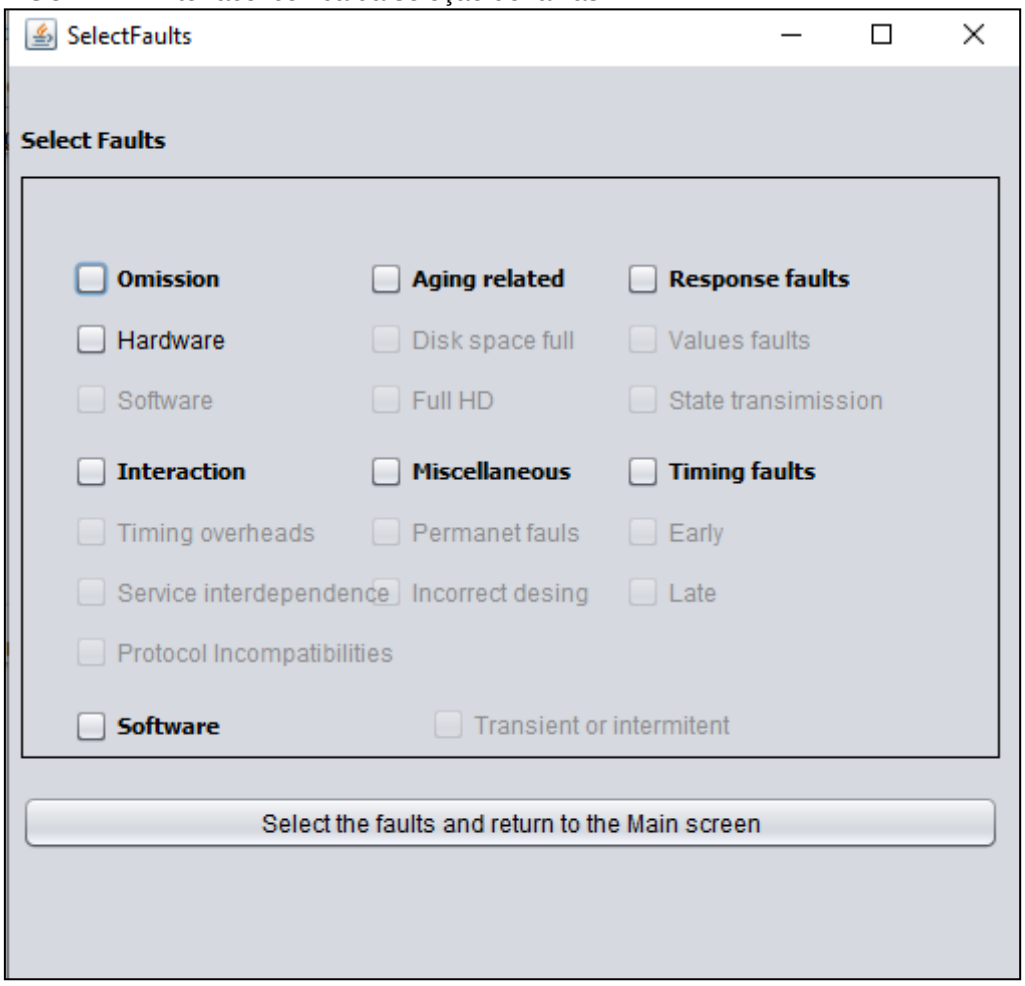
Na Figura 17 é apresentada a interface icônica JSelecionarFalhas.java, que foi desenvolvida neste trabalho, com a finalidade de possibilitar ao usuário a seleção das falhas que poderão ser injetadas no ambiente de simulação.

Na interface icônica apresentada na Figura 16, estão listadas as principais falhas e sua respectiva classificação de acordo com Argawal *et al.* (2018). Nesta interface, é permitido que o usuário selecione o tipo de falha, que pode ser: de omissão, de envelhecimento, de resposta, falhas de *software*s, de temporização, de interação ou falhas diversas.

Ao selecionar o tipo de falha de acordo com a classificação de Argawal *et al.* (2018), são habilitados as demais caixas de checagem (*JCheckBox),* de forma que permita ao usuário selecionar o tipo de falha que deseja inserir.

Caso algum tipo de falha tenha sido selecionada, ela retornará uma mensagem para o usuário nesta mesma janela. Este retorno e interação promovem a comunicação entre as partes.

FIGURA 17: Interface icônica da seleção de falhas



FONTE: Print screen da interface icônica JSelecionarFalhas.java no iSPD

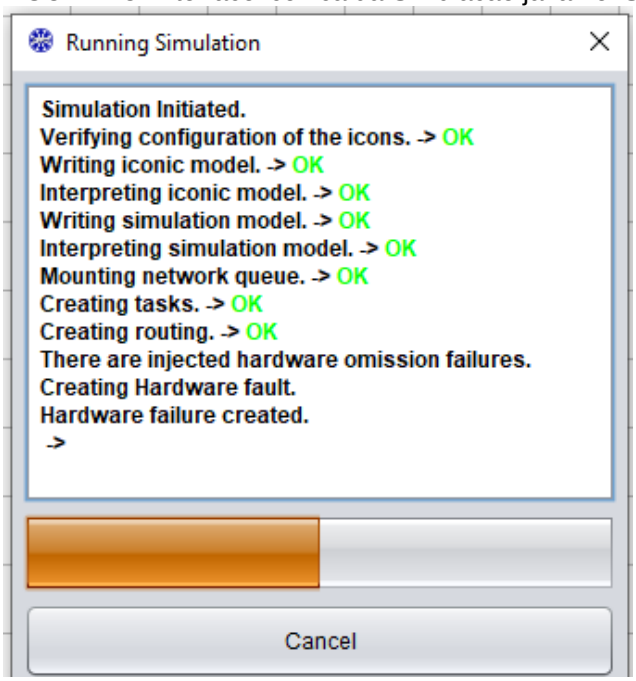
O desenvolvimento do gerador de falha incluiu alterações com finalidade de atualização da interface icônica Simulacao.java. Nela, foram adicionadas mais funcionalidades, para que retorne para o usuário informações referentes à injeção, detecção e tratamento das falhas são exibidas caso sejam verdadeiras. Caso seja detectada na simulação a seleção de falhas, é exibido uma mensagem ao usuário que há falhas detectadas para serem inseridas na simulação.

Após o seu tratamento, é exibido ao usuário uma mensagem na interface icônica Simulacao.java que a falha foi tratatada. E, em caso de existência de mais

falhas, o mesmo processo é realizado: a injeção, a detecção e o tratamento de acordo com o Diagrama de sequência apresentado na seção 4.3.

Na Figura 18 é apresentada a imagem da interface icônica Simulacao.java quando há a injeção de falha.

FIGURA 18: Interface icônica da Simulacao.java no iSPD



FONTE: Print screen da interface icônica Simulacao.java no iSPD

Após o usuário configurar os demais itens de simulação necessários, a simulação é iniciada. Neste momento, a classe SimulacaoSequencialCloud.java faz a verificação de todos os itens necessários para que a simulação seja realizada e retorne para o usuário em sua janela de checagem, representada pela classe JSimulacao.java, na Figura 17.

Em virtude deste processo de checagem e injeção de falhas, as classes SimulacaoSequencialCloud.java e JSimulacao.java sofreram alterações de implementação para o processo de injeção de falhas.

O desenvolvimento do sistema injetor de falhas contemplou a sua modalagem e o desenvolvimento de algumas classes. Cada classe do sistema injetor de falhas é representada por um tipo de falha, as quais são injetadas por chamadas de métodos que permitem a sua inserção no ambiente simulado.

Após as falhas selecionadas terem sido injetadas, a biblioteca de falhas – nomeada de FaultLib neste trabalho - é acionada para a aplicação de técnicas de tratamento de falhas de acordo com a falha injetada.

A modelagem de falhas retratada neste trabalho foi validada por meio da criação do sistema injetor de falhas, representado pela interface incônica JPrincipal.java, JSelecionarFalhas.java e as chamadas dos métodos para a injeção das falhas selecionadas nas classes Principal.java, SimulacaoSequencialCloud.java e Simulacao.java.

Após a ocorrência da simulação, o iSPD retorna seus resultados ao usuário em uma interface icônica já existente, em que são exibidas informações referentes ao tempo de simulação, se a esta ocorreu sem interrupções, ou seja, se houve satisfação de 100%.

Além dessas informações, são exibidos dados referente ao número de máquinas simuladas e o tempo de simulação estimado para o ambiente simulado. Neste, é verificado a eficiência do simulador no que diz respeito ao tempo para produção de resultado. (MENEZES, 2012)

Se o tempo estiver dentro do intervalo de tempo prevista sua taxa de eficiência é classificada como "boa", senão como "ruim", a partir das métricas implementadas em trabalhos anteriores neste simulador em que foram adotadas critérios relacionados do ponto de vista de velocidade, em decorrência à execução do processo de execução adotado a partir da seleção dos algoritmos de escalonamento implementados. (MENEZES, 2012)

Os experimentos relacionados à validação da modelagem proposta e do desenvolvimento do gerador de falhas foi a injeção de falhas de omissão de *hadware* e de *software* a partir dos levantamentos de revisão de trabalhos anteriores realizados, apresentados no Capítulo 5 deste trabalho.

Para o tratamento das falhas descritas, foi criada a biblioteca de tratamento de falhas, nomeada de ‘FaultLib’ que possui uma classe para cada tipo de falha, de forma que, em cada classe são apresentados as melhores técnicas de tratamento de falha para cada um das falhas.

O tratamento das falhas descritas se mostrou eficaz, uma vez que nele é verificado se há Máquinas Físicas disponíveis para a continuidade da simulação, e para as máquinas que foram desligadas, se foram aplicadas as técnicas de tratamento na classe *FIHardware*.java.

Neste simulador, para a ocorrência de falha de omissão de *software*, foram desenvolvidas técnicas reativas como a ressubmissão de tarefas, de acordo com Argawal *et al.* (2015), em que a Biblioteca de falhas, representada por FaultLib,

atuou na reorganização das Máquinas Virtuais disponíveis, representada pela sigla VM (*Virtual Mahcine),*em que o seu tratamento ocorreu na classe FH*Software*.java.

Caso a aplicação da técnica reativa de ressubmissão de tarefas não apresentar um tratamento eficaz, é aplicado a técnica proativa de migração preemptiva, em que ocorre novamente a divisão das tarefas solicitadas na simulação somente para as Máquinas Virtuais disponíveis (ligadas) no momento.

Neste caso, o processo de tratamento de falhas é mais longo, em virtude da necessidade de aplicação de mais de uma técnica para o tratamento da falha existente, fato que incide no aumento do custo de simulação, o qual é apresentado na classe MetricasCusto.java, alocado no pacote ispd.motor.metricas.

CAPÍTULO 5 RESULTADOS

Foram realizados experimentos para a validação da funcionalidade do sistema injetor de falhas e da biblioteca de falhas para as modelagens de falhas implementadas descritas na seção 4.4.

Para o evento da injeção de falhas ocorrer, faz-se necessário selecionar as falhas que deverão ser injetadas na simulação.

A injeção de falhas é realizada a partir das falhas selecionadas e sua injeção ocorre durante o processo de simulação.

A fase de detecção e tratamento da falha consiste na capacidade do sistema reconhecer que houve uma falha, nos casos estudados, de falha de omissão de *hardware* ou de *software*, o que acarretaria que o sistema deveria se reorganizar automaticamente de forma a promover o reescalonamento das máqiunas disponíveis para que a simulação dê continuidade.

Para validar os processos de injeção de falhas, detecção e tratamento da falha injetada por meio da interação da biblioteca de falha, foram realizados diferentes experimentos, organizados por grupos.

Houve a organização de cinco grupos, executados com o número constante de 32 Máquinas Físicas e variando o número de Máquinas Virtuais ou *Virtual Machines* (*VMs*), com 16, 32, 64, 128 e 156 de acordo com o Quadro 18.

Para os casos de testes apresentados, os ambientes de simulação criados foram de acordo com os estudos apresentados por Rodrigues *et al.* (2019) em que os recursos de cada uma das Máquinas Virtuais são baseadas no modelo A5 C5.9xlarge, com processadores de 36 núcleos e 72 GB de memória.

QUADRO 18: Quadro de organização dos casos de testes

Nome do Grupo	Nº de PM	Nº de VM
G1	32	16
G2	32	32
G3	32	64
G4	32	128
G5	32	256

FONTE Da própria autora (2020)

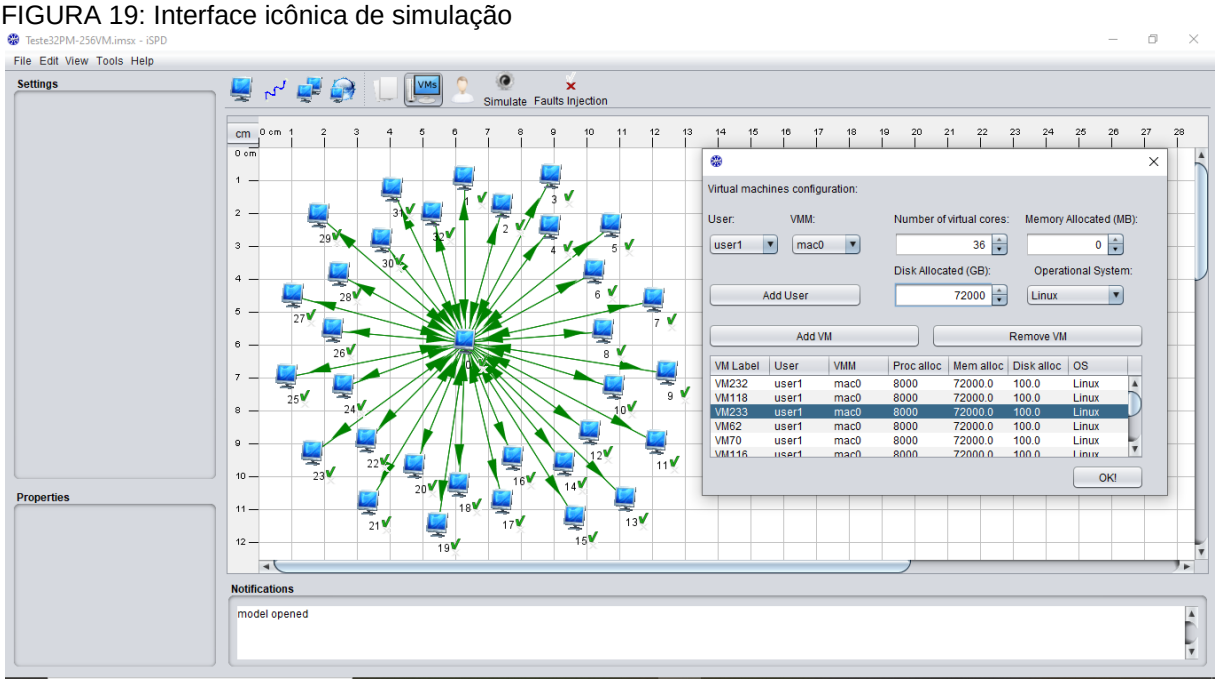
No grupo do experimento G1, houve a organização de uma modelagem icônica de 32 Máquinas Físicas e 16 Máquinas Virtuais. No grupo do experimento G2 houve a organização de uma modelagem icônica de 32 Máquinas Físicas e 32 Máquinas Virtuais. No grupo do experimento G3 houve a organização de uma modelagem icônica de 32 Máquinas Físicas e 64 Máquinas Virtuais.

No grupo do experimento G4 houve a organização de uma modelagem icônica de 32 Máquinas Físicas e 128 Máquinas Virtuais. E por fim, no grupo de experimento G5, houve a organização de uma modelagem icônica de 32 Máquinas Físicas e 256 máquivas virtuais.

Na Figura 18 é apresentada a interface icônica do iSPD. Nela, é possível configurar a inserção de Máquinas Virtuais, e suas respectivas configurações, como o sistema operacional, o seu número de processadores, espaço e armazenamento em memória.

Para a reaização dos testes, houve a configuração do ambiente de simulação no iSPD. Nele, as Máquinas Físicas e virtuais foram configuradas de acordo com os estudos apresentados por Rodrigues *et al.* (2019), em que são inseridas Máquinas Virtuais de 32 processadores, com sistema operacional Linux, de acordo com a Figura 19 apresentada.

FIGURA 19: Interface icônica de simulação



FONTE: Print screen da interface icônica JSimulacaoSequencialCloud.java no iSPD

5.1. Resultados de simulação e análises

Os resultados dos experimentos mostraram-se promissores para a modelagem de falhas no simulador iSPD na classe de serviço *IaaS*, devido a integração que esta classe possui com as demais. Uma vez injetada uma falha na simulação nesta classe de serviço, as demais classes poderão sofrer outras falhas, devido a seu efeito cascata.

A proposta da modelagem de falhas, gerador e bibliteca de falhas, foram validados a partir do experimentos realizados.

Foram realizados três experimentos para cada grupos descrito no Quadro 18. O primeiro experimento foi a injeção da falhas de omissão de *hardware*. O segundo foi a injeção de falhas de *software* e o terceiro foi a injeção da falha de *hardware* e de *sofware* para do grupo apresentado.

Foi utilizada a técnica da experimentação, pois, cada experimento realizado foi tratado como um processo de formulação ou verificação da modelagem de falhas descritas na seção 4.4 e da teoria de tolerância falhas, descrita no capítulo 3.

Para tabular os resultados obtidos, foram considerados os tempos de execução da simulação final, ou seja, com a injeção e o tratamento da falha. Se a simulação foi realizada sem precisar ter sido interrompida, o simulador retorna que sua eficiência foi de 100%. Uma vez que a simulação é realizada e pode ser salva no iSPD, foi considerado o tempo de simulação, disponibilidade e estabilidade do simulador durante o processo de simulação.

Cada resultado obtido após o experimento foi comparado com os demais resultados obtidos pertencentes ao mesmo grupo. Foi estabelecido o critério de compatação entre os resultados extraídos do relatório do tempo de simulação, e eficácia da simulação já existentes no iSPD. Outro critério desenvolvido neste trabalho e utilizado na análise dos resultados dos experimentos foi a eficácia no tratamento de falhas injetadas.

Um dos instrumentos de qualidade de *software* e de melhoria contínua foram definidos como os principais objetivos a compreensão e controle. Seus objetivos estão relacionados com foco no custo, o risco, o tempo e a qualidade. A qualidade pode ser um fatos intríseco de acordo com as normativas analisadas, desta forma ela não foi considerada neste trabalho.

Para os experimentos realizados, foram considerados os objetivos relacionados ao custo, ao tempo e à qualidade. Todos os experimentos foram realizados em máquina local, no sistema operacional Windows 10.

Os experimentos que receberam a injeção de apenas uma falha, de *software* ou de *hardware,* obtiveram os resultados esperados, ou seja, as falhas foram injetadas e tratadas, porém há um maior tempo de execução da simulação em relação às simulações do ambiente sem a ocorrência destas, com um aumento significativo de até 50%.

Ao término de cada simulação, o simulador iSPD retorna se o custo computacional no ambiente simulado foi baixo, médio ou alto. Este é calculado a partir das configurações do ambiente de simulação configurado. O simulador iSPD também retorna a eficiência, ou seja: se ela ocorreu dentro do tempo previsto, ele retornará que a eficiência foi boa, senão a eficiência será classificada como ruim.

Nas simulações que houve a injeção de falhas, uma ou mais, em que ocorreu a sobrecarga de trabalho, o simulador retornou que as eficiências foram ruins, porque ultrapassaram o tempo médio de simulação previsto para o ambiente simulado sem falhas.

Foi verificado que, em testes de omissão de *hardware* e de *software*, quando há o desligamento de Máquinas Físicas ou Virtuais, há uma diminuição na taxa de transferência e aumento da sobrecarga de trabalho para as demais máquinas do ambiente. Isto auxilia para que o retorno da simulação em relação à eficiência seja ruim, uma vez que o desligamento de uma máquina pode impactar diretamente na escalabilidade.

Dos 48 experimentos realizados, três foram descartados porque houve a falha no processamento do iSPD relacionado à simulação, o que representa um índice de perda de 6%.

No Quadro 19 são apresentados os experimentos válidos realizados no Grupo 1, em que é organizado com 32 PM e 16 VMs. Foi verificado que o tempo médio de simulação é de 15,97952 segundos para as falhas injetadas referentes à omissão de *hardware*. Para as injeção das falhas do omissão de *software*, o tempo médio de simulação foi de 10,1284333 em segundos. Quando houve simulações das duas falhas, o tempo médio foi de 15,1841733 segundos.

Todos os experimentos realizados tiveram uma satisfação de 100% em relação ao ambiente simulado, no procedimento de injeção e no tratamento das

falhas, a qual é apresentada pelo iSPD na exibição dos resultados. Porém, devido ao tempo médio de simulação exibido pelo iSPD, houve uma taxa de eficiência ruim devido ao tratamento de falhas.

QUADRO 19: Experimento realizado no grupo G1 (32 PM com 16 VMs)

	Tempo em segundos	**Taxa de satisfação**	**Taxa de eficiência**
Injeção de falha de omissão de *hardware*	28.6045 12.4315 6.90256	100%	Ruim
Injeção de falha de omissão de *software*	10.0559 12.3673 7.96210	100%	Ruim
Injeção de falha de omissão de *hardware* e de *software*	7.98372 13.0778 24.4910	100%	Ruim

FONTE: Da própria autora (2020)

No Quadro 20, são apresentados os resultados válidos dos experimentos realizados no Grupo 2, em que é organizado com 32 PM e 32 VMs. Foi verificado que o tempo médio de simulação é de 16,2588227 segundos para as falhas injetadas referentes à omissão de *hardware*. Para as injeção das falhas do omissão de *software*, o tempo médio de simulação foi de 23,0749 em segundos. Quando houve simulações das duas falhas, o tempo médio foi de 16,6708667, com satisfação de 100% em todos os experimentos.

Em relação à eficiência retornada após a injeção e o tratamento de falhas dos experimentos realizados pelo simulador, foi abaixo do esperado no ambiente simulado sem falhas. Esperava-se que o tempo de simulação seria maior devido ao tempo gasto para injetar e tratar a falha, mas que não comprometeria a escalabilidade do ambiente de forma significativa, como foi demonstrado nos experimentos realizados com 32 Máquinas Virtuais.

QUADRO 20: Experimento realizado no grupo G2 (32 PM com 32 VMs)

	Tempo em segundos	**Taxa de satisfação**	**Taxa de eficiência**
Injeção de falha de omissão de *hardware*	10.0134 8.46368 30.2967	100%	Ruim
Injeção de falha de omissão de *software*	26.9855 21.6876 20.5516	100%	Ruim
Injeção de falha de omissão de *hardware* e de *software*	20.0341 17.7912 12.1873	100%	Ruim

FONTE: Da própria autora (2020)

No Quadro 21 são apresentados os experimentos realizados no Grupo 3 (G3), em que é organizado com 32 PM e 64 VMs. Neste grupo, há o dobro de Máquinas Virtuais simuladas, o no final há o retorno de uma taxa ruim de satisfação. Após estudos, foi verificado que isto se dá em razão da sobrecarga dos métodos no processo de injeção e tratamento das falhas.

Neste grupo foi verificado que o tempo médio de simulação é de 22,550667 segundos para as falhas injetadas referentes à omissão de *hardware*, em que fica evidente um aumento de tempo de 10% em relação ao Grupo 2 (G2). Para as injeção das falhas do omissão de *software*, o tempo médio de simulação foi de 14.9403333 em segundos, ou seja, 35% menor e mais rápido em relação ao G2.

Quando houve simulações das duas falhas, o tempo médio foi de 7,20457333, também com uma redução de 56% em relação ao Grupo 2 e com satisfação de 100% em todos os experimentos e eficiência ruim quando simulado o ambiente de 64 Máquinas Virtuais.

QUADRO 21: Experimento realizado no grupo G3 (32 PM com 64 VMs)

	Tempo em segundos	**Taxa de satisfação**	**Taxa de eficiência**
Injeção de falha de omissão de *hardware*	14.0837 23.1220 30.4463	100%	Ruim
Injeção de falha de omissão de *software*	13.9548 15.4648 15.4014	100%	Ruim
Injeção de falha de omissão de *hardware* e de *software*	11.6898 5.20694 4.71698	100%	Ruim

FONTE: Da própria autora (2020)

No Quadro 22 são apresentados os experimentos realizados no Grupo 4 (G4), em que é organizado com 32 PM e 128 VMs. O tempo médio de simulação é de 11,15974 segundos, ou seja, com uma redução no tempo em 50,50% em relação ao Grupo 3, quando os experimentos realizados são referentes às falhas injetadas referentes à omissão de *hardware.*

Para as injeção das falhas do omissão de *software,* o tempo médio de simulação foi de 13,1045333 em segundos, ou seja, 12% menor e mais rápido em relação ao G2.

Quando houve simulações das duas falhas, o tempo médio foi de 10,28775 em relação à 7,20457333 do Grupo 2, o que evidencia que houve um aumento de tempo médio de simulação de 42,79% nos grupos comparados. Em relação aos demais experimentos nesta categoria, este é o único experimento que está destoando, o que pode ter sido ocasionado pela injeção da falha no final da simulação o que demanda mais tempo para o seu tratamento.

QUADRO 22: Experimento realizado no grupo G4 (32 PM com 128 VMs)

	Tempo em segundos	**Taxa de satisfação**	**Taxa de eficiência**
Injeção de falha de omissão de *hardware*	15.8695 7.87978 9.72994	100%	Ruim
Injeção de falha de omissão de *software*	10.3564 17.8404 11.1168	100%	Ruim
Injeção de falha de omissão de *hardware* e de *software*	9.63250 11.7815 9.44925	100%	Ruim

FONTE: Da própria autora (2020)

No Quadro 23 são apresentados os experimentos realizados no Grupo 5 (G5), em que é organizado com 32 PM e 256 VMs. Foi observado que o tempo médio de simulação é de 13,4992133 segundos, em que é apresentando um aumento de 20% em relação aos demais grupos de experimentos nesta modalidade de falha.

Para as injeção das falhas do omissão de *software*, o tempo médio de simulação foi de 13,5658 com o melhor resultado nos grupos de testes realizados para este tipo de injeção e tratamento da falha, com uma melhora de 10% em relação aos testes do Grupo 4 (G4).

Quando houve simulações das duas falhas, o tempo médio foi de 10,8385833 segundos, com um aumento de 5% em relação aos experimentos do Grupo G4. Neste grupo também houve a satisfação de 100%, ou seja, foram executados todos os experimentos, mas a eficiência medida no iSPD foi classificada como ruim quando simulado o ambiente de 256 Máquinas Virtuais.

QUADRO 23: Experimento realizado no grupo G5 (32 PM com 256 VMs)

	Tempo em segundos	**Taxa de satisfação**	**Taxa de eficiência**
Injeção de falha de omissão de *hardware*	17.4963 9.55644 13.4449	100%	Ruim
Injeção de falha de omissão de *software*	14.6845 14.8535 11.1594	100%	Ruim
Injeção de falha de omissão de *hardware* e de *software*	9.08968 13.8723 9.55377	100%	Ruim

FONTE: Da própria autora (2020)

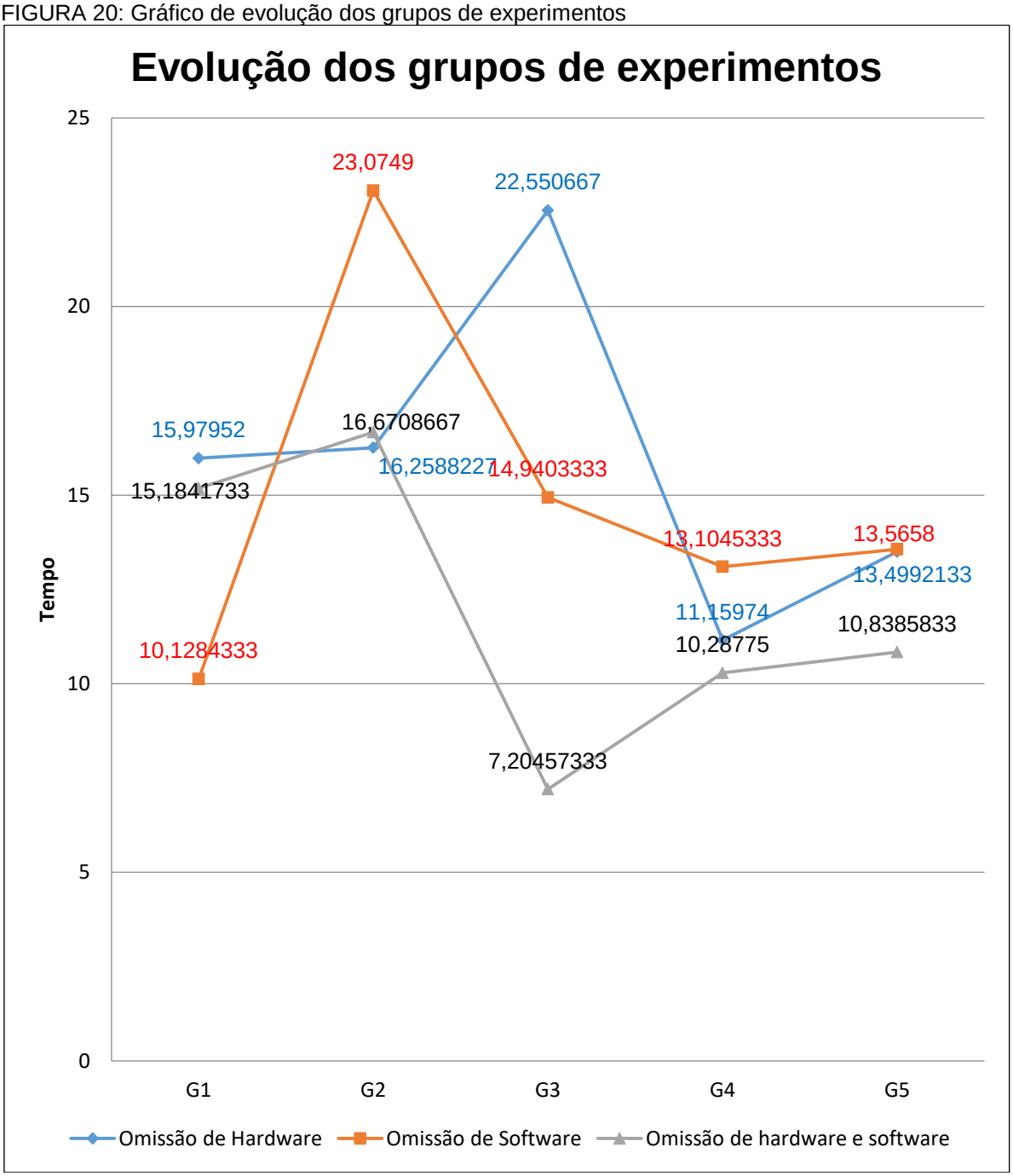
O tempo médio de simulação foi maior quando foram realizados a injeção e tratamento de falhas de *hardware*, o que resultou em um tempo médio de 15,88959 segundos no ambiente de simulação, em virtude de tipo de falha injetada e tratada.

No Grupo 2 é observado o pior caso em relação ao tempo médio de simulação, que ocorreu nos experimentos realizados com a injeção e tratamento de falhas de omissão de *software*, em que é apresentado a média de tempo de segundos, quando comparada a todos os grupos de experimentos e tipos de experimentos realizados.

Nos grupos dos experimentos realizados, foram validados os estudos referentes à tolerância de falhas em simuladores, sobre quando há o aumento de Máquinas Virtuais, há uma diminuição no tempo médio de simulação, que pode chegar em até 50% em alguns casos observados. Essa diminuição fica mais evidente quando comparados os grupos G1 e G2 e depois os grupos G4 e G5.

Na Figura 20 é apresentado um gráfico em que é possível acompanhar estes comparativos dos grupos de experimentos.

FIGURA 20: Gráfico de evolução dos grupos de experimentos



FONTE: Da própria autora (2020)

5.2. Considerações finais

A realização de diferentes simulações para os experimentos em um ambiente controlado possibilitou a verificação da validação e tabulação dos experimentos realizados a partir da solução proposta. No capítulo 6 serão apresentadas conclusões dos experimentos realizados para a modelagem de falhas em nuvem.

CAPÍTULO 6 CONCLUSÃO

Neste trabalho foi analisada a modelagem de falhas em simuladores, em específicos, o iSPD, simulador já existente no Laboratório de Sistemas Paralelos e Distribuídos. Pelo fato de ser um sistema legado, o trabalho não se restringiu somente a modelagem de falhas, mas também na compreensão da estrutura do simulador por meio de utilização de técnicas de engenharia reversa para exploração dos algoritmos de escalonamento já existentes.

Foi realizada a revisão sistemática da literatura, estudo e análise das taxas de ocorrência de falhas nos principais gestores de nuvem.

Após a compreensão da arquitetura e funcionalidades já existentes no iSPD, foi elaborada a modelagem do sistema injetor de falhas e sua respectiva biblioteca de tratamento de falhas.

Na sequência, foi implementado o sistema injetor de falhas, a biblioteca de tratamento de falhas (FaultLib). Em relação à interface do iSPD, foi implementado a interface icônica “JSelecionarFalhas.java” e ajustes das interfaces icônicas “JPrincipal.java” e “Simulacao.java” para adição de funcionalidades relacionadas à injeção e tratamento de falhas.

É possível concluir que, de acordo com os resultados apresentados, a modelagem de falhas para o iSPD foi trabalhada para que este possua a possibilidade de injeção de outros tipos de falhas além das descritas neste trabalho.

A realização dos experimentos possibilitou a modelagem das falhas em nuvem, em cenários severos e que há a possibilidade da injeção aleatória de falhas e seu respectivo tratamento.

Mesmo que os estudos indiquem que, entre as técnicas de tratamento de falhas amplamente utilizadas são a replicação e a de *checkpoint,* elas podem ser ineficientes para sistemas amplos e podem apresentar um alto custo computacional a alta complexidade quando adotadas em grandes conjuntos de dados.

Na implementação das falhas de omissão de *hardware* e de *software* neste trabalho, foi evidenciado nos experimentos que há variação de eficiência, custo e escalabilidade em decorrência do universo simulado como também na quantidade de falhas que foram inseridas, de acordo com os resultados apresentados.

Considerando os cinco grupos de experimentos realizados no iSPD, foi possível observar que o tamanho do *cluster* simulado com o uso de Máquinas

Virtuais interfere no tempo final de simulação em que houve janelas com índices de até 50% de diferença entre os experimentos realizados. É possível concluir que a modelagem de falhas é importante para que sejam realizados simulações próximas de ambientes reais, assim como a implementação de possíveis técnicas reativas e ativas nos ambientes de computação em nuvem.

Para trabalhos futuros é sugerida a continuidade da implementação de injeção de falhas ainda não implementadas descritas por Argawal *et al.* 2015, como falhas de negação de serviço, falhas de resposta, falhas de temporização e falhas de integração.

REFERÊNCIAS

ACM – Digital Library, **Biblioteca digital de artigos científicos.** Disponível em https://dl.acm.org/ acesso no período de setembro/2017 a maio/2019.

AGARWAL, Himanshu; SHARMA, Anju. **A comprehensive survey of fault tolerance techniques in cloud computing.** In: 2015 International Conference on Computing and Network Communications (CoCoNet). IEEE, 2015. p. 408-413.
AGARWAL, ANDERSON, T.; LEE, P. A. Fault tolerance -principles and practice. [S.l.]:

ALONSO-MONSALVE, Saúl; GARCÍA-CARBALLEIRA, Félix; CALDERÓN, Alejandro. **A heterogeneous mobile cloud computing model for hybrid clouds**. Future Generation Computer Systems, v. 87, p. 651-666, 2018.

ANDRADE, V. S. Dissertação de Mestrado em ciência da computação, apresentada no Ibilce. **Tolerância a falhas em OpenStack** - Survey. 2019. São José do Rio Preto

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 14724: **Informação e documentação: trabalhos acadêmicos: apresentação.** 3. ed. Rio de Janeiro: ABNT, 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 6023: **Informação e documentação: referências: elaboração.** 2. ed. Rio de Janeiro: ABNT, 2018.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 6024: **Informação e documentação: numeração progressiva das seções de um documento: apresentação**. 2. ed. Rio de Janeiro: ABNT, 2012.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. ABNT NBR 6027: **Informação e documentação: sumário: apresentação**. Rio de Janeiro: ABNT, 2012.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 10520: **Informação e documentação: citações em documentos: apresentação.** Rio de Janeiro: ABNT, 2002.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6028: **Informação e documentação: resumo: apresentação.** Rio de Janeiro: ABNT, 2003.

AZIZ, Khadija; ZAIDOUNI, Dounia; BELLAFKIH, Mostafa. **Real-time data analysis using Spark and Hadoop.** In: 2018 4th International Conference on Optimization and Applications (ICOA). IEEE, 2018. p. 1-6.

BATISTA, Glauber C. *et al.* Using Externals IdPs on OpenStack: **A Security Analysis of OpenID Connect, Facebook Connect, and OpenStack Authentication**. In: 2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA). IEEE, 2018. p. 920-927.

BHADAURIA, Rohit *et al.* A survey on security issues in cloud computing. arXiv preprint arXiv:1109.5388, p. 1-15, 2011.

BHARDWAJ, Sushil; JAIN, Leena; JAIN, Sandeep. Cloud computing: A study of infrastructure as a service (*IAAS*). **International Journal of engineering and information Technology**, v. 2, n. 1, p. 60-63, 2010.

BITTENCOURT, Luiz F. *et al.* Scheduling in distributed systems: A cloud computing perspective. **Computer Science Review**, v. 30, p. 31-54, 2018.

BUYYA, Rajkumar; BELOGLAZOV, Anton; ABAWAJY, Jemal. Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges. **arXiv preprint arXiv:1006.0308**, 2010.

BUYYA, Rajkumar; BROBERG, James; GOSCINSKI, Andrzej M. (Ed.). **Cloud computing: Principles and paradigms**. John Wiley & Sons, 2010.

BUYYA, Rajkumar; VENUGOPAL, Srikumar. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. **arXiv preprint cs/0404027**, 2004.

CALHEIROS, Rodrigo N. *et al.* CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. ***Software*: Practice and experience,** v. 41, n. 1, p. 23-50, 2011.

CARVALHO, Marcus; MENASCÉ, Daniel A.; BRASILEIRO, Francisco. Capacity planning for *IaaS* cloud providers offering multiple service classes. **Future Generation Computer Systems**, v. 77, p. 97-111, 2017.

CIRNE, Walfredo. Grids computacionais: Arquiteturas, tecnologias e aplicações. **III ERAD-Escola Regional de Alto Desempenho,** Santa Maria, RS, 2003.

Comparação entre os serviços do AWS e do Azure. Publicado em 29 nov. 2017, disponível em https://docs.microsoft.com/pt-br/azure/architecture/aws-professional/services , acesso em 06.set.17h04.

COULOURIS, G.; DOLLIMORE J.; KINBERG T. . **Distributed Systems**, 3º Edition, Addison Wesley, 2001

COULOURIS, G.; DOLLIMORE J.; KINBERG T. . **Sistemas Distribuídos**, 4ª Edição, Addison Wesley, 2005

DANTAS, Jamilson *et al.* **Eucalyptus-based private clouds: availability modeling and comparison to the cost of a public cloud.** Computing, v. 97, n. 11, p. 1121-1140, 2015.

DIKAIAKOS, Marios D. *et al.* Cloud computing: Distributed Internet computing for IT and scientific research. **IEEE Internet computing**, v. 13, n. 5, p. 10-13, 2009.

FRITZ, Raphael; ZHANG, Ping. Overview of fault-tolerant control methods for discrete event systems. **IFAC-PapersOnLine**, v. 51, n. 24, p. 88-95, 2018.

GANGADHARAN, G. R**. Open source solutions for cloud computing**. Computer, n. 1, p. 66-70, 2017.

GARCÍA, Álvaro López; DEL CASTILLO, Enol Fernández; PLASENCIA, Isabel Campos. An efficient cloud scheduler design supporting preemptible instances. **Future Generation Computer Systems**, v. 95, p. 68-78, 2019.

GARCÍA, Álvaro López; DEL CASTILLO, Enol Fernández; PLASENCIA, Isabel Campos. An efficient cloud scheduler design supporting preemptible instances. Future Generation Computer Systems, v. 95, p. 68-78, 2019.

GUROV, DMYTRO; TRAVASSOS, GUILHERME HORTA, AMARAL, EDGAR, AUGUSTO GURGEL DO. **Introdução à engenharia de software experimental. Programa de engenharia de sistemas e computação**, COPPE/UFRJ, Rio de Janeiro, 2002.

IEEE.ORG, **Biblioteca digital de artigos científicos**. Uma organização sem fins lucrativos, o IEEE é a maior organização profissional técnica do mundo dedicada ao avanço da tecnologia para o benefício da humanidade. Disponível em https://ieeexplore.ieee.org/Xplore/home.jsp acesso no período de setembro/2017 a maio/2019.

Inderscience Publisher, Biblioteca Digital de Jornais, livros e artigos científicos. Disponível em https://www.inderscience.com/. Acesso no período de setembro/2017 a maio/2019.

JOSEP, Anthony D. *et al.* A view of cloud computing. **Communications of the ACM**, v. 53, n. 4, 2010.

KANSO, Ali *et al.* **Enhancing openstack fault tolerance for provisioning computing environments**. In: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE). IEEE, 2017. p. 77-83.

KOCHOVSKI, Petar; DROBINTSEV, Pavel D.; STANKOVSKI, Vlado. Formal Quality of Service assurances, ranking and verification of cloud deployment options with a probabilistic model checking method. **Information and *Software* Technology**, v. 109, p. 14-25, 2019.

KULKARNI, Bhagyashri; BHOSALE, Varsha. **Efficient storage utilization using erasure codes in openstack cloud.** In: 2016 International Conference on Inventive Computation Technologies (ICICT). IEEE, 2016. p. 1-5.

KUMARI, Priti; KAUR, Parmeet. A survey of fault tolerance in cloud computing. **Journal of King Saud University-Computer and Information Sciences**, 2018.

KUMARI, Raghvendra; PANDEY, Arti. A Survey on Security Issues in Cloud Computing. **International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET)**, v. 2, n. 3, p. 506-517, 2016.

LIU, Jialei *et al.* **Using proactive fault-tolerance approach to enhance cloud service reliability.** IEEE Transactions on Cloud Computing, v. 6, n. 4, p. 1191-1202, 2016.

LUIZ, Aldelir Fernando; LUNG, Lau Cheuk; CORREIA, Miguel. Byzantine fault-tolerant transaction processing for replicated databases. In: 2011 **IEEE 10th International Symposium on Network Computing and Applications**. IEEE, 2011. p. 83-90.

MANACERO, A. *et al.* "iSPD: an iconic-based modeling simulator for distributed grid". In: **Annuals of 45th Annual Simulation Symposium**. Orlando, USA: [s.n.], 2012. (ANSS12, CDROM), p. 1-8.

MELL, Peter *et al.* The NIST definition of cloud computing. 2011.

MENEZES, Denison. **Geração de algoritmos de escalonamento para simulação de grades computacionais**. 2012. Disponível em https://repositorio.unesp.br/handle/11449/89339/ . Acesso no período de fevereiro a setembro de 2020.

Microsoft, **O que é o *IaaS* Infraestrutura como serviço**. Disponível em https://azure.microsoft.com/pt-br/overview/what-is-*IaaS*/ acesso em 14 jan. 2019. às 15h32

Microsoft, **O que é o *PaaS* Plataforma como serviço**. Disponível em https://azure.microsoft.com/pt-br/overview/what-is-*PaaS*/ acesso em 14 jan. 2019. às 15h09.

Microsoft, O que é o *SaaS Software* como serviço. Disponível em https://azure.microsoft.com/pt-br/overview/what-is-*SaaS*/ acesso em 14 jan. 2019. às 15h09.

MITTAL, Deepali; AGARWAL, Neha. A review paper on Fault Tolerance in Cloud Computing. In: 2015 . **2nd International Conference on Computing for Sustainable Global Development** (INDIACom). IEEE, 2015. p. 31-34.

MONTERO, Rubén S. *et al.* **Extending the Cloud to the Network Edge**. IEEE Computer, v. 50, n. 4, p. 91-95, 2017.

OPENNEBULA. OpenNebula Homepage. 2017.

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.

PIJANOWSKI, Keit ,"**Understanding public clouds** : *IaaS*, *PaaS*, *SaaS*" on KeithPij.com Disponível em http://www.keithpij.com/Home/tabid/36/EntryID/27/Default.aspx , acesso em , 5 nov. 2018 por Bhardwaj, Jain e Jain (2010).

POOLA, Deepak; RAMAMOHANARAO, Kotagiri; BUYYA, Rajkumar**. Fault-tolerant Workflow Scheduling using Spot Instances on Clouds**. In: ICCS. 2014. p. 523-533.

PRESSMAN, R. S.. **Engenharia de *software* – uma abordagem profissional.** 7ª EDIÇÃO, Mc Graw Hill & editora Bookman, AMGH Editora LTDA, 2011, ISBN 978-85-63308-33-7. Porto Alegre-RS. Prentice-Hall, 1981.

PRESSMAN, Roger; MAXIM, Bruce. **Engenharia de Software**-8ª Edição. McGraw Hill Brasil, 2016.

Research Gate, **Biblioteca Digital de Jornais, livros e artigos científicos**. Disponível em https://www.researchgate.net/. Acesso no período de setembro/2017 a maio/2019.

RODRIGUES, João Antonio Magri *et al.* **Improving Virtual Machine Consolidation for Heterogeneous Cloud Computing Datacenters**. In: 2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). IEEE, 2019. p. 176-179.

RODRIGUES, João Antonio Magri. **Otimização de alocação de Máquinas Virtuais em datacenter heterogêneo de sistema de computação em nuvem**. 2019.

SANAEI, Zohreh *et al.* Heterogeneity in mobile cloud computing: taxonomy and open challenges. **IEEE Communications Surveys & Tutorials**, v. 16, n. 1, p. 369-392, 2013.

Science Direct, **Biblioteca Digital de Jornais, livros e artigos científicos**. Disponível em https://www.sciencedirect.com/ acesso no período de setembro/2017 a maio/2019.

SHAOKA, Zhao *et al.* **Architecture and scheduling scheme design of TsinghuaCloud based on OpenStack.** Journal of Computer Applications, v. 33, n. 12, p. 3335-3338, 2013.

SHEN, Zhiming *et al.* Supercloud: A Library Cloud for Exploiting Cloud Diversity. **ACM Transactions on Computer Systems (TOCS)**, v. 35, n. 2, p. 6, 2017.

SILVA, Diogo Tavares da. **Abordagem icônica para modelagem e simulação de ambientes de computação em nuvem?**. 2015.

SOMASHEKHAR, Vikas Maheshwari; SINGH, R. P. **Analysis of Micro Inversion to Improve Fault Tolerance in High Speed VLSI Circuits**. International Research Journal of Engineering and Technology (IRJET), v. 6, n. 03, p. 5041-5044, 2019.

SU, Li; ZHOU, Yongluan. Passive and partially active fault tolerance for massively parallel stream processing engines. **IEEE Transactions on Knowledge and Data Engineering**, v. 31, n. 1, p. 32-45, 2017.

SYED, Hassan Jamil *et al.* Cloud monitoring: A review, taxonomy, and open research issues. **Journal of Network and Computer Applications**, v. 98, p. 11-26, 2017.

TAMASHIRO, Camila Baleiro Okado et al. Modelagem de falhas em nuvem Cloud Fault Modeling. **In: 2020 15th Iberian Conference on Information Systems and Technologies (CISTI).** IEEE, 2020. p. 1-6.

TANENBAUM A.; VAN STENN M. Sistemas Distribuídos - Princípios e Paradigmas. 2ª edição Prentice Hall, Pearson Education, 2002.

TOOSI, Adel Nadjaran; CALHEIROS, Rodrigo N.; BUYYA, Rajkumar. Interconnected cloud computing environments: Challenges, taxonomy, and survey. ACM Computing Surveys (CSUR), v. 47, n. 1, p. 7, 2014.

TORQUATO, Matheus *et al.* **An approach to investigate aging symptoms and rejuvenation effectiveness on *software* systems**. In: 2017 12th iberian conference on Information systems and technologies (CISTI). IEEE, 2017. p. 1-6.

TRAVASSOS, Guilherme Horta; GUROV, Dmytro; AMARAL, E. A. G. G. **Introdução à engenharia de *software* experimental**. 2002.

VARGHESE, Blesson; BUYYA, Rajkumar. Next generation cloud computing: New trends and research directions. **Future Generation Computer Systems**, v. 79, p. 849-861, 2018.

VERAS, M. Cloud Computing: nova arquitetura da TI. Brasport, 2015. Editora Brasport Livros e Multimídia Ltda. Rio de Janeiro/RJ.

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.

VOGEL, Adriano *et al.* **Private *IaaS* clouds: a comparative analysis of OpenNebula, CloudStack and OpenStack.** In: 2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP). IEEE, 2016. p. 672-679.

WILKINSON, B.; Grid **Computing – Techniques and Applications**, a CHAPMAN & Hall Book, ISNB 978-4200-6953-3 (Hardback), 2010, 1ª edição.

YU, Jia *et al.* **Gridbus workflow enactment engine**. CRC Press: USA, 2009.

YU, Jia; BUYYA, Rajkumar. A taxonomy of workflow management systems for grid computing. **Journal of Grid Computing**, v. 3, n. 3-4, p. 171-200, 2005.

YUSUF, Salisu Ibrahim; JUNAIDU, Sahalu B. Parallel and Distributed Intra Query Transient Fault Tolerance Model via Parity Checking. In: **2018 14th International Conference on Electronics Computer and Computation (ICECCO)**. IEEE. p. 206-212.

ZHANG, Deyu *et al.* Resource allocation for green cloud radio access networks with hybrid energy supplies. **IEEE transactions on vehicular technology**, v. 67, n. 2, p. 1684-1697, 2017.

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

TERMO DE REPRODUÇÃO XEROGRÁFICA

Autorizo a reprodução xerográfica do presente Trabalho de Conclusão, na íntegra ou em partes, para fins de pesquisa.

São José do Rio Preto, 20/11/2020

Camila B Okado Tamelio

Assinatura do autor