Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 33, Num. 4 (2026) 53-67

RESEARCH ARTICLE

Fault Injection via Code Mutation for Dependability Assessment in Cloud Environments

Injeção de Falhas por Mutação de Código para Avaliação de Dependabilidade de Ambientes de Nuvens

Guilherme Silva[1]*, Erica Sousa[1]

Abstract: Software fault injection assesses the resilience of complex systems, especially in cloud environments with multiple interdependent components. Since many software releases contain bugs not detected by traditional testing, this technique simulates realistic failures (e.g., by code mutation) to observe system behavior under defects, identify vulnerabilities before production, and understand how failures propagate. This paper presents CIMut, a fault injection tool through source code mutation in cloud environments, publicly available as a web application. The proposed approach was evaluated through an experimental study on the OpenStack cloud platform. A total of 62 experiments were performed on OpenStack, each injecting faults into different system components. The study showed that 96.7% of the injected faults resulted in bugs, classified as explicit errors (crashes, exceptions) or bugs with functional impact (incorrect behavior, data loss). These results demonstrate that the CIMut tool can generate representative faults that can be used to assess the resilience of complex software systems such as OpenStack.
Keywords: cloud environment — dependability — software fault injection — OpenStack

Resumo: A injeção de falhas de software avalia a resiliência de sistemas complexos, especialmente em ambientes de nuvem com múltiplos componentes interdependentes. Como muitos softwares são lançados com bugs não detectados por testes tradicionais, essa técnica simula falhas realistas (ex.: por mutação de código) para observar o comportamento do sistema sob defeitos, identificar vulnerabilidades antes da produção e entender como as falhas se propagam. Este artigo propõe a avaliação de dependabilidade de ambientes de nuvem por meio do CIMut, uma ferramenta de injeção de falhas por mutação de código-fonte acessível publicamente como aplicação web. A ferramenta proposta foi avaliada através de um estudo experimental realizado na plataforma de nuvem OpenStack. Foram realizados 62 experimentos no OpenStack, cada um injetando falhas em diferentes componentes do sistema. Os resultados do estudo mostraram que 96,7% das falhas injetadas resultaram em bugs, classificados como erros explícitos (travamentos, exceções) ou bugs com impacto funcional (comportamento incorreto, perda de dados). Esses resultados demonstram que a ferramenta CIMut é capaz de gerar falhas representativas que podem ser utilizadas para avaliar a resiliência de sistemas de software complexos como o OpenStack.
Palavras-Chave: ambiente de nuvem — dependabilidade — injeção de falha de software — OpenStack

[1] *Departamento de Computação, Universidade Federal Rural de Pernambuco (UFRPE), Brasil*
***Corresponding author**: guilherme.silvad@ufrpe.br, erica.sousa@ufrpe.br
DOI: http://dx.doi.org/10.22456/2175-2745.149183 • **Received:** 02/10/2025 • **Accepted:** 27/06/2026

1. Introdução

A computação em nuvem revolucionou a forma como empresas e indivíduos utilizam recursos computacionais, transformando fundamentalmente a maneira como os serviços de TI são provisionados e consumidos. Sua adoção abrange diversos setores, como negócios, redes sociais, computação científica e de alto desempenho, oferecendo flexibilidade, escalabilidade e eficiência de custos [1]. A nuvem computacional se caracteriza pela oferta de acesso ubíquo, conveniente e sob demanda a um conjunto compartilhado de recursos computacionais configuráveis, com provisionamento ágil e escalonamento dinâmico. Esse modelo tem se mostrado um catalisador para a transformação digital e inovação em organizações [2], além de gerar impactos positivos no desempenho financeiro no curto prazo [3].

A escalabilidade e flexibilidade inerentes à computação

em nuvem tornaram-na uma tecnologia essencial para a economia moderna, facilitando a expansão de empresas em todo o mundo. Estimativas indicam que os gastos globais com serviços de nuvem pública atingiram US$ 592 bilhões em 2023, refletindo sua importância estratégica para organizações que buscam agilidade, resiliência e crescimento [4]. A migração para a nuvem está experimentando um crescimento significativo, e espera-se que essa tendência continue nos próximos anos [5].

Apesar dos inúmeros benefícios, a computação em nuvem também enfrenta o desafio da ocorrência de falhas. Devido à interdependência entre os componentes da nuvem, uma falha em um único componente pode gerar impactos em cascata, afetando diversos outros. Os prejuízos anuais associados a falhas na nuvem são estimados em 700 bilhões de dólares [6].

Nesse contexto, a injeção de falhas destaca-se como uma técnica fundamental para analisar a resiliência de sistemas em nuvem. Por meio da introdução controlada de falhas, é possível avaliar a capacidade do sistema de lidar com cenários adversos e desenvolver mecanismos de tolerância a falhas. A implementação desses mecanismos visa mitigar a ocorrência de falhas e garantir a continuidade das operações, proporcionando maior disponibilidade e confiabilidade no ambiente de nuvem [7, 8].

Diante da importância da computação em nuvem e dos desafios impostos pela ocorrência de falhas, este trabalho apresenta o CIMut (*Cloud Injection Mutator*), uma ferramenta de injeção de falhas por mutação de código-fonte para avaliação de dependabilidade de ambientes de nuvem. O CIMut realiza a injeção em três etapas (verificação, mutação e monitoramento) e é acessível publicamente como aplicação web em ⟨https://cimut.vercel.app/⟩, com arquitetura em três camadas: interface gráfica para configuração e execução de experimentos, API *backend* para processamento das requisições e agente de execução residente na infraestrutura de nuvem sob teste. A ferramenta possibilita a análise do impacto das falhas nos serviços da nuvem, bem como a avaliação da resiliência e da dependência entre seus módulos. Por meio da injeção controlada de falhas, é possível identificar a ocorrência de falhas e erros, observar sua propagação entre componentes e fundamentar decisões de projeto voltadas à confiabilidade. Para validar a ferramenta, foi conduzido um estudo experimental na plataforma OpenStack, envolvendo 62 experimentos de injeção de falhas baseadas em *bugs* reais previamente classificados pela metodologia *Orthogonal Defect Classification* (ODC).

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os principais conceitos utilizados neste trabalho. A Seção 3 discute trabalhos relacionados sobre injeção de falhas em ambientes de nuvem. A Seção 4 detalha a ferramenta CIMut, incluindo o algoritmo de injeção e sua arquitetura. A Seção 5 descreve o estudo de caso conduzido na plataforma OpenStack e discute os resultados obtidos. Por fim, a Seção 6 apresenta as considerações finais e os trabalhos futuros.

2. Referencial Teórico

Nesta seção serão definidos alguns conceitos importantes para o entendimento deste trabalho.

2.1 Computação em Nuvem

A computação em nuvem representa um modelo para disponibilizar recursos computacionais (como processamento, armazenamento e rede) de forma acessível e universal. Esse modelo é caracterizado por cinco atributos essenciais: autoatendimento sob demanda, amplo acesso à rede, agrupamento de recursos, elasticidade rápida e serviço mensurado. Essas características permitem que os recursos sejam alocados de maneira prática conforme a necessidade, promovendo escalabilidade e eficiência operacional [9, 10].

A computação em nuvem oferece diversos modelos de implantação para atender a diferentes necessidades de negócios. No modelo de nuvem privada, a infraestrutura é provisionada para uso exclusivo de uma única organização, garantindo controle integral sobre gestão, segurança e conformidade [11, 12]. Com o avanço das tecnologias de virtualização e orquestração, surgiram plataformas que permitem configurar ambientes altamente customizáveis e seguros [11]. Ferramentas como OpenShift [13], Apache CloudStack [14], OpenNebula [15], Cloudify [16] e Eucalyptus [17] são amplamente adotadas para implantar nuvens privadas, atendendo a requisitos específicos de desempenho, escalabilidade e conformidade [4, 18, 19].

O OpenStack é uma plataforma de nuvem *open source* que capacita os usuários a construir e gerenciar seus próprios ambientes de nuvem privada. Amplamente utilizada por grandes empresas como Blizzard Entertainment, Daum Communication, LINE Corporation e Adobe, o OpenStack oferece uma estrutura modular para atender a diversas necessidades da nuvem. O Nova permite a criação e gerenciamento de instâncias (máquinas virtuais), enquanto o Neutron gerencia as redes virtuais. O Cinder e o Swift gerenciam o armazenamento em volumes e objetos, respectivamente. Além disso, o OpenStack conta com funcionalidades como monitoramento (Ceilometer), automação (Heat) e gerenciamento de identidade (Keystone), fornecendo uma solução completa para a implementação e operação de nuvens privadas [20].

2.2 Injeção de Falhas

A dependabilidade é uma característica essencial para sistemas computacionais, especialmente em aplicações críticas. Um sistema confiável é aquele que executa suas funções corretamente, mesmo em condições adversas ou na presença de falhas. A injeção de falhas, ao simular essas condições, permite identificar vulnerabilidades, contribuindo para o desenvolvimento de sistemas mais robustos e resilientes. Ao testar e avaliar a capacidade de um sistema em lidar com falhas, é possível aprimorar sua arquitetura, seus mecanismos de detecção e recuperação de erros e, consequentemente, aumentar sua confiabilidade geral [21, 22].

Os métodos de injeção de falhas em hardware atuam diretamente nos componentes físicos, como pinos de *chip* e

circuitos internos. Essa abordagem, embora precisa, exige equipamentos especializados, acesso físico ao sistema e pode ser complexa e cara de implementar [23]. Por outro lado, os métodos de software introduzem falhas diretamente no estado do software, como a alteração de valores em variáveis ou a interrupção de fluxos de execução. Essa técnica oferece maior flexibilidade, menor custo e menor intrusão no sistema [22]. Falhas de software podem ser introduzidas de duas maneiras: modificando o código-fonte ou alterando o comportamento do sistema em execução. Isso pode ser feito usando recursos existentes ou desabilitando processos. A mutação de código altera o código-fonte do programa para simular erros de programação. Por exemplo, um operador aritmético pode ser trocado (+ por -) ou uma instrução condicional pode ser invertida. A mutação permite avaliar a robustez do código contra erros lógicos e falhas de implementação, contribuindo para a construção de sistemas mais confiáveis [24, 25].

2.3 Orthogonal Defect Classification

A Classificação Ortogonal de Defeitos (ODC) representa uma metodologia estruturada para análise de defeitos de software, combinando aspectos quantitativos e qualitativos. Seu objetivo é fornecer um mecanismo eficaz para identificar e corrigir falhas de programação de forma precisa e eficiente. A metodologia ODC organiza-se em duas categorias principais: classes abertas e classes fechadas. As classes abertas auxiliam na identificação de defeitos, analisando as atividades em andamento quando a falha foi descoberta, os gatilhos que a manifestam e seu impacto potencial nos usuários. Já as classes fechadas, utilizadas após a correção, focam na natureza exata do defeito e no escopo da correção, categorizando o alvo da correção, o tipo de defeito corrigido, qualificadores para detalhar o defeito, o tempo da correção e a origem do código [26].

3. Trabalhos Relacionados

A injeção de falhas tem se consolidado como uma técnica fundamental para estudos sobre falhas em ambientes de nuvem. Sua capacidade de introduzir falhas controladas permite avaliar a confiabilidade e os mecanismos de recuperação desses sistemas sob condições adversas. A injeção de falhas também é empregada no desenvolvimento de ferramentas especializadas. Diversas ferramentas foram desenvolvidas para esse propósito, cada uma com características e funcionalidades específicas. A AFEX [27, 28], por exemplo, é uma ferramenta de propósito geral que simula o efeito de testes de caixa preta, injetando falhas em software. Já o DEFINE [29, 30] foca em ambientes distribuídos, simulando falhas em hardware e corrompendo informações armazenadas ou transmitidas. A EucaBomber [31, 32], desenvolvida para plataformas de nuvem Eucalyptus, desativa processos da plataforma, simulando falhas em seu funcionamento.

Outras ferramentas de injeção de falhas em software incluem: a FIAT [33, 34], que injeta falhas de forma semelhante à execução de um *trojan*; a G-SWIFT [24, 35] e a PREFAIL [36, 37], que alteram o código binário do programa para simular falhas; e a ProFIPy [38, 39], que oferece uma linguagem de domínio específico (DSL) para programar a injeção de falhas.

Além das ferramentas em software, existem simuladores de falhas em hardware, como o FIMSIM [40, 41], que utiliza o simulador M5, e o FOCUS [42, 34], que utiliza o simulador SPLICE para injetar falhas em designs VLSI. A FERRARI [43, 34] automatiza o processo de injeção de falhas em arquiteturas SPARC, enquanto a FTAPE [44, 34] estressa os recursos de uma máquina alvo em computadores tolerantes a falhas Tandem Integrity S2.

Há também ferramentas que combinam hardware e software para injeção de falhas. A MESSALINE [45], por exemplo, injeta falhas em diversos pontos simultaneamente em componentes físicos de plataformas específicas. A RIFLE [46, 47] utiliza software para injetar falhas em nível de pinos (*pin-level faults*). Já a Xception [48, 47] possui um sistema base com formato bem definido, permitindo a injeção de falhas em arquiteturas baseadas em processador, memória e barramentos.

Considerando a relevância crescente desta área para a confiabilidade de sistemas de nuvem, este trabalho apresenta o CIMut, uma ferramenta de injeção de falhas baseada em mutação de código-fonte, projetada para operar em diversos ambientes de computação em nuvem e preenchendo uma lacuna significativa nas soluções atualmente disponíveis.

A ferramenta privilegia a injeção de falhas via software por meio da mutação sistemática do código-fonte, fundamentada na constatação de que muitos sistemas são implantados com bugs residuais não detectados durante os processos convencionais de controle de qualidade [49]. O objetivo central desta pesquisa é apresentar a avaliação de dependabilidade de ambientes de nuvem através de injeção de falhas. O CIMut permite analisar o comportamento dos sistemas em modo de operação, simulando cenários de falhas. A ferramenta é descrita em detalhes na Seção 4.

A Tabela 1 apresenta uma análise comparativa de 15 soluções de injeção de falhas em ambientes de nuvem. Dessas soluções, 10 utilizam software e 5 utilizam hardware para injetar falhas. Entre as baseadas em software, 4 adotam a técnica de mutação de código. Dessas, apenas 2 modificam o código-fonte, enquanto as outras 2 alteram o código binário. Diferentemente das abordagens de alteração binária (G-SWIFT, PREFAIL) e baseadas em DSL (ProFIPy), o CIMut atua diretamente no código-fonte do ambiente de nuvem sob teste, com recursos de verificação de conteúdo e rastreamento de mutações.

4. Cloud Injection Mutator (CIMut)

Esta seção descreve a ferramenta de injeção de falhas proposta neste trabalho, denominada CIMut (*Cloud Injection Mutator*), que aplica a técnica de mutação de código-fonte para introduzir falhas de forma controlada em ambientes de nuvem. A ferramenta é acessível publicamente como aplicação web

Table 1. Soluções de injeção de falhas em ambientes de nuvem

Solução	Tipos de Falhas Injetadas	Ambiente/Plataforma
AFEX [27]	Software	Propósito Geral
DEFINE [29]	Software e simula falhas em hardware	Foco em ambientes distribuídos
EucaBomber [31]	Software	Plataforma de nuvem Eucalyptus
FIAT [33]	Software	Propósito Geral
G-SWIFT [24]	Software — Alteração de código binário	Propósito Geral
PREFAIL [36]	Software — Alteração de código binário	Propósito Geral
ProFIPy [38]	Software — Alteração de código via DSL	Plataforma OpenStack
FIMSIM [40]	Simulador de falhas de hardware	Simulador M5
FOCUS [42]	Simulador de falhas em hardware	Simulador SPLICE
FERRARI [43]	Software e simula falhas em hardware	Arquitetura SPARC
FTAPE [44]	Software	Tandem Integrity S2 fault-tolerant computer
MESSALINE [45]	Hardware usando componentes físicos	Testado em plataformas específicas
RIFLE [46]	Hardware usando software	Propósito Geral
Xception [48]	Hardware usando software	Sistema base com formato bem definido
CIMut (ferramenta proposta)	Software — Alteração de código-fonte	Propósito Geral, validado em OpenStack

em ⟨https://cimut.vercel.app/⟩ e foi projetada para realizar testes experimentais por meio da injeção de diferentes tipos de falhas e análise do comportamento do sistema frente a essas condições. A Subseção 4.1 apresenta o algoritmo de injeção implementado pelo CIMut, a Subseção 4.2 descreve a arquitetura da ferramenta e a Subseção 4.3 detalha o fluxo de trabalho dos experimentos.

4.1 Algoritmo de Injeção de Falhas

O algoritmo de injeção de falhas implementado pelo CIMut organiza o processo em três etapas principais (Verificação, Mutação e Monitoramento) e está estruturado em torno de quatro funções básicas: (i) *verificação prévia de conteúdo*, que valida a presença do trecho-alvo no arquivo e linha especificados, evitando mutações em locais inexistentes; (ii) *mutação controlada*, que substitui o conteúdo da linha pelo novo valor parametrizado pelo usuário; (iii) *rastreamento*, que registra cada mutação aplicada em um arquivo de log, com metadados como arquivo, linha, conteúdo original, conteúdo mutado e *timestamp*, garantindo auditoria e reversibilidade dos experimentos; e (iv) *coleta de métricas*, que captura o estado e o consumo de recursos dos serviços-alvo após a mutação. O Algoritmo 1 apresenta o procedimento de injeção em pseudocódigo.

A verificação prévia (linha 3) é um diferencial em relação a abordagens de mutação cega: ela garante que a mutação só é aplicada se o conteúdo esperado realmente existe no local indicado, o que evita falsos positivos em experimentos conduzidos sobre múltiplas versões de uma mesma plataforma. O reinício do host (linha 7) é necessário para que o ambiente de nuvem recompile o código-fonte mutado, no caso do OpenStack, isso garante que o serviço afetado passe a executar com a falha injetada. As operações de conexão, leitura, escrita e desconexão (linhas 1, 2, 5 e 12) abstraem o canal de comunicação entre a ferramenta e o ambiente de nuvem sob teste. Sua implementação concreta na ferramenta CIMut é descrita na Subseção 4.2.

Algorithm 1 Injeção de falha por mutação de código-fonte em ambiente de nuvem

Require: host remoto h, credenciais c, arquivo-alvo f, linha ℓ, conteúdo original s_o, conteúdo mutado s_m
Ensure: mutação aplicada e registrada em log

1: $conn \leftarrow$ CONECTAR(h, c)
2: $arquivo \leftarrow$ LER$(conn, f)$
3: **if** LINHACONTEM$(arquivo, \ell, s_o)$ **then**
4: SUBSTITUIRLINHA$(arquivo, \ell, s_m)$
5: ESCREVER$(conn, arquivo, f)$
6: REGISTRARLOG$(f, \ell, s_o, s_m, timestamp)$
7: REINICIARHOST$(conn)$
8: COLETARMETRICAS$(conn, serviços)$
9: **else**
10: **abortar:** conteúdo s_o não encontrado em f, linha ℓ
11: **end if**
12: DESCONECTAR$(conn)$

4.2 Arquitetura da Ferramenta

A ferramenta CIMut implementa o algoritmo descrito na Subseção 4.1 em arquitetura de três camadas, acessível publicamente como aplicação web em ⟨https://cimut.vercel.app/⟩:

- **Camada de Interface (*Frontend* Web)** — responsável pela configuração e execução de experimentos por meio de interface gráfica. Contempla módulos para injeção manual de falhas, monitoramento de tarefas em execução e visualização da infraestrutura da nuvem sob teste.
- **Camada de Aplicação (API *Backend*)** — implementada em FastAPI, recebe e processa as requisições enviadas pela interface gráfica e coordena o fluxo de

execução dos experimentos, comunicando-se com o agente de execução por meio de *webhook*.

- **Agente de Execução** — componente residente na infraestrutura de nuvem sob teste, acionado via *webhook*. Opera em ciclo de dois estágios para cada operação de mutação: no estágio de *Read*, o agente extrai o código-fonte atual do arquivo-alvo e o retorna à camada de aplicação para verificação. Em seguida, no estágio de *Modify*, o agente recebe as coordenadas da mutação (linha e novo conteúdo) e aplica a alteração diretamente no arquivo do servidor. O agente também é responsável pelo registro estruturado das mutações em arquivo de log (com host, arquivo, linha, conteúdo original, conteúdo mutado e *timestamp*, garantindo auditoria e reversibilidade) e pela coleta de métricas dos serviços-alvo após a mutação.

Essas camadas comunicam-se via HTTP/REST (entre *frontend* e API) e *webhook* (entre API e agente), e são orquestradas pelo fluxo de trabalho ilustrado na Figura 1. O código-fonte da ferramenta encontra-se em processo de registro de patente e será disponibilizado publicamente após a conclusão desse processo. A acessibilidade da ferramenta como aplicação web pública constitui um diferencial em relação a soluções correlatas, que não disponibilizam código nem interface acessível a terceiros, conforme discutido na Subseção 5.3.

4.3 Fluxo de Trabalho

A Figura 1 ilustra o fluxo de trabalho da injeção de falhas utilizando o CIMut. Neste exemplo, o CIMut foi utilizado com o OpenStack. A ferramenta habilita a análise e modificação do OpenStack, facilitando a identificação do impacto de bugs no desempenho do sistema. O fluxo de trabalho se divide em três etapas principais: Verificação, Mutação e Monitoramento.

O processo se inicia com uma inspeção minuciosa de um repositório de falhas (*Visualizar bug no repositório de falhas*). A partir da análise de um bug específico, o componente do OpenStack onde a falha se manifesta é identificado. Posteriormente, o código-fonte do OpenStack, disponível no repositório GitHub, é consultado para determinar o arquivo preciso a ser modificado com o objetivo de induzir a falha (*Conferir código do OpenStack no GitHub*). Para auxiliar nesse processo, o agente do CIMut é acionado para validar a existência do conteúdo a ser alterado no ambiente onde o OpenStack está provisionado. O agente recebe como parâmetros a identificação do ambiente, o caminho do arquivo e o número da linha para garantir a precisão da modificação (*CIMut: Verificar se o conteúdo que deseja alterar existe*).

Após a confirmação da existência do conteúdo desejado, o agente executa a substituição, aplicando a mutação no arquivo conforme o número da linha e o novo conteúdo recebidos da camada de aplicação (*CIMut: Alterar código-fonte do sistema*). Após a mutação, o ambiente é reiniciado para que o OpenStack recompile o código com as alterações realizadas (*Reiniciar host*). Em seguida, uma carga de trabalho, com a criação de uma instância, é executada para simular o uso do sistema sob a influência da mutação.

Por fim, o agente captura dados dos serviços do OpenStack (*Coletar dados com o script de monitoramento*). As informações coletadas são agrupadas por módulo e apresentadas em um *dashboard*. Essa visualização facilita a análise do impacto das mutações no desempenho e funcionamento do OpenStack, permitindo uma compreensão mais profunda da relação entre bugs e a dependabilidade do sistema.

5. Estudo de Caso

Esta seção apresenta um estudo de caso com o objetivo de avaliar a dependabilidade do ambiente de nuvem configurado com a plataforma OpenStack, através de experimentos de injeção de falhas com mutação de código utilizando através de experimentos de injeção de falhas com mutação de código utilizando a ferramenta CIMut. Para este estudo, o OpenStack foi configurado utilizando o DevStack [20], uma ferramenta que permite a implantação do OpenStack em um ambiente de desenvolvimento ou teste. Uma configuração *all-in-one* foi utilizada, onde uma única máquina virtual executa as funções de nó Controlador, nó de Computação e nó de Rede. Essa abordagem simplifica a implantação e permitiu a criação de um ambiente de teste completo sem a necessidade de uma infraestrutura complexa. A Tabela 2 descreve as especificações da máquina utilizada.

No ambiente DevStack, todos os serviços seguem o padrão de nomenclatura `devstack@<nome>.service`. Para concisão, utilizaremos apenas o `<nome>` ao longo desta seção. Os serviços disponíveis representam os componentes do OpenStack. O Nova, responsável pelo gerenciamento de instâncias virtuais, conta com serviços como: `n-api`, que atua como a API principal, processando solicitações de gerenciamento de instâncias; `n-cpu`, que executa as instâncias na máquina *host*; e `n-sch`, que agenda a execução das instâncias em *hosts* apropriados. O `n-api-meta` fornece metadados para as instâncias, enquanto os serviços `n-cond-cell1`, `n-novnc-cell1` e `n-super-cond` são utilizados em implantações com múltiplas células, auxiliando na comunicação e no acesso remoto ao console das instâncias.

O Neutron, responsável pelas redes virtuais, utiliza o serviço `q-svc` como seu serviço principal. O `q-ovn-metadata-agent`, quando o *plugin* OVN é empregado, fornece informações de rede para as instâncias. O Glance, que gerencia imagens de disco, utiliza o serviço `g-api` para sua API. O Cinder, responsável pelos volumes de disco, possui o `c-sch` como agendador, o `c-api` para sua API e o `c-vol` para gerenciar a criação e exclusão de volumes nos *backends* de armazenamento. O *Placement*, que gerencia o inventário de recursos, utiliza o serviço `placement-api` para sua API.

Além dos serviços específicos de cada componente, o DevStack também utiliza serviços gerais. O `dstat` coleta e exibe estatísticas de desempenho, o `etcd` atua como um banco

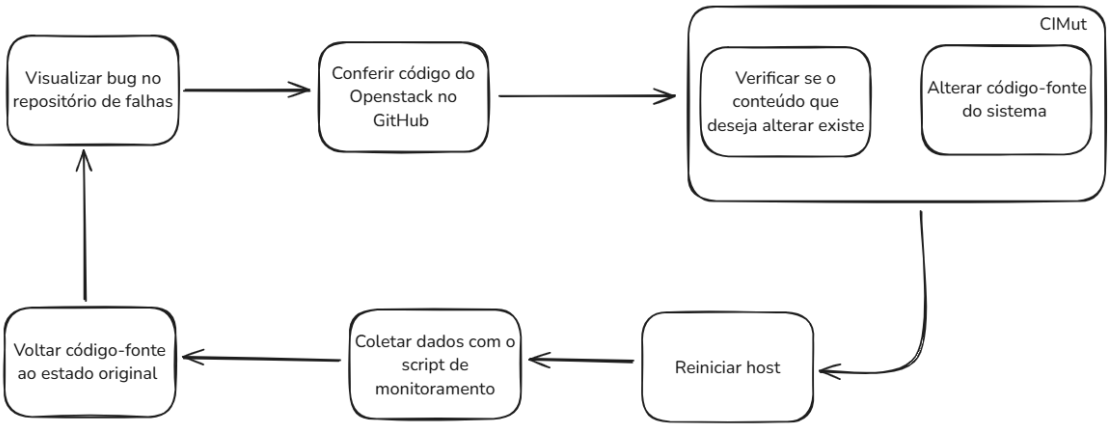

Figure 1. Fluxo de trabalho para injeção de falhas com o CIMut

Table 2. Configuração da máquina virtual utilizada no estudo de caso

Nó	Processador	Memória	Disco	Sistema Operacional	Módulos OpenStack
Compute, Controller, Network	AMD Ryzen 3 4300U	8 GB	1 TB	Ubuntu 22.04.4 LTS (Jammy Jellyfish)	Nova, Neutron, Cinder, Swift, Glance, Keystone

de dados distribuído para armazenar dados de configuração e estado, e o `keystone` gerencia a autenticação e autorização dentro do OpenStack.

Os experimentos tiveram como objetivo injetar falhas utilizando o CIMut na nuvem privada configurada, utilizando a técnica de mutação do código-fonte. Isso envolveu a modificação de trechos específicos do código, introduzindo erros que pudessem resultar em falhas operacionais. Neste trabalho, as falhas foram injetadas no Nova, o módulo do OpenStack responsável pela computação, incluindo o gerenciamento de recursos e a criação de instâncias virtuais. Assim, a introdução de erros nesse serviço impacta diretamente a administração das máquinas virtuais.

Para direcionar a escolha dos arquivos a serem modificados, foram utilizados relatórios de bugs disponíveis no Bugzilla [50], plataforma oficial de reporte de falhas do OpenStack. Esses bugs foram previamente organizados em categorias como *crash*, *hang*, degradação de desempenho e funcionalidade incorreta com base em uma análise manual, considerando os critérios da Classificação Ortogonal de Defeitos [51]. Essa categorização não está presente originalmente no Bugzilla, tendo sido realizada como parte de trabalhos anteriores desenvolvidos no mesmo grupo de pesquisa. Após essa etapa, os arquivos mencionados nos relatos foram localizados no repositório GitHub do OpenStack. A definição da linha exata a ser modificada foi feita manualmente, com base na descrição técnica do *bug* e na análise do código relacionado. Em seguida, foram inseridas falhas deliberadas com alto potencial de causar erros de execução em Python, como erros de indentação, remoção de parâmetros obrigatórios de funções ou substituições que violassem a estrutura sintática do código. A escolha dessas alterações foi feita de forma a simular falhas de comportamento semelhantes às relatadas nos *bugs*, mesmo sem replicar exatamente a causa original.

Embora nem todas as falhas tenham reproduzido de forma idêntica o cenário descrito nos relatórios do Red Hat Bugzilla, muitas geraram comportamentos anômalos próximos aos observados, como falhas na criação de instâncias ou panes no painel de controle. Essa abordagem contribuiu para validar a eficácia da técnica de mutação como forma de injetar falhas realistas, com base em problemas previamente observados pela comunidade de desenvolvedores do OpenStack. Cada cenário foi testado separadamente utilizando o fluxo apresentado na Seção 4 (Figura 1), e cada bug foi replicado duas vezes, totalizando 62 experimentos. Além do uso do CIMut, foi utilizada uma carga de trabalho baseada na criação e exclusão de máquinas virtuais com a menor configuração possível (1 vCPU e 1 GB de RAM), sem volumes de armazenamento anexados. Essas instâncias foram empregadas unicamente para gerar alocação de recursos de computação e rede. A instanciação sem volumes se justifica pelo fato de o armazenamento persistente ser gerenciado por um módulo distinto do OpenStack, o Cinder, que não foi o foco direto dos testes neste trabalho.

No entanto, é importante destacar que, embora as máquinas virtuais criadas não utilizassem volumes, o comportamento do sistema em relação ao serviço de volumes ainda foi indiretamente afetado em alguns cenários. *Bugs* como "*Volume attached at two instances after failover because compute outage*", "*Live Migration*" e "*Trailing Number Launch Block*", por exemplo, envolvem interações com o serviço de volumes e, durante os testes, observou-se que a aba de volumes no painel de controle sumia ou apresentava inconsistências (mesmo sem volumes efetivamente criados). Além disso, em certos testes,

ações como a tentativa de acessar ou criar um volume resultaram em falhas ou comportamentos anômalos, evidenciando a sensibilidade do sistema a falhas nos módulos relacionados, mesmo que não diretamente utilizados nas instâncias criadas.

5.1 Classificação de Bugs

Os *bugs* mencionados neste relatório foram extraídos do Bugzilla [50], plataforma dedicada ao reporte de *bugs* relacionados ao OpenStack. Nos testes, foram considerados apenas *bugs* que envolvem o módulo Nova, totalizando 31 *bugs*, distribuídos da seguinte forma: 12 de *crash*, 3 de *hang*, 3 de degradação de desempenho e 13 de funcionalidade incorreta, conforme a ODC [51] (Tabela 3).

A Tabela 3 apresenta uma lista dos *bugs* utilizados nos experimentos. Cada *bug* é detalhado em colunas específicas, fornecendo informações como: um identificador único (ID do *Bug*) para facilitar a referência, o módulo do sistema afetado, um título resumido do problema, e uma classificação baseada na metodologia ODC [51]. Essa classificação, distribuída nas Colunas 4 a 6, organiza os defeitos em diferentes dimensões. O campo *activity* indica a fase do ciclo de vida de desenvolvimento de software (SDLC) em que o defeito foi descoberto, como requisitos, *design*, codificação, testes (unitários, funcionais, de sistema, etc.) ou operação. O *trigger* representa a condição ou ação específica que levou à manifestação do defeito, como entrada inválida, uso inesperado, falhas de *hardware* ou integração. Já o *impact* descreve a consequência do defeito na funcionalidade, desempenho ou experiência do usuário, podendo variar de falhas menores a interrupções críticas, perda de dados ou brechas de segurança. Por fim, a Coluna 7 da tabela apresenta a severidade atribuída ao *bug* conforme relatado no Bugzilla.

5.2 Resultados dos Experimentos de Injeção de Falhas

A Figura 2 apresenta a relação entre *bugs* classificados como *crash* e o status dos serviços. Os seguintes *bugs* como *Failover volume*, *Post-Reboot Desync*, *Libvirtd Hang (RHOSP16)*, *Live Migration with SR-IOV*, *Cold Evacuation (Down Hosts)* e *Messaging Timeout (BDM Creation)* não impactaram o status dos serviços, obtendo um resultado de 17 serviços ativos. No entanto, esses *bugs* causaram problemas funcionais durante a criação de instâncias. Isso pode ser explicado pela natureza lógica dos erros, que comprometem a funcionalidade sem alterar o status do serviço.

Em contrapartida, outros *bugs* causaram mudanças no status dos serviços. No caso do *Instance Resize (No-Swap)*, os serviços `n-cpu`, `n-sch`, `n-cond-cell1`, `n-novnc-cell1` e `n-super-cond` ficaram inativos, enquanto apenas `n-api` e `n-api-meta` permaneceram ativos. Isso representa que 71,47% dos serviços do módulo Nova tiveram degradação, resultando em status de falha. Em relação a todos os módulos, 12 serviços permaneceram ativos enquanto 5 tiveram status de falha.

Os demais *bugs* alteraram o status apenas do serviço `n-cpu`, resultando em 16 serviços ativos e 1 com falha. Essa observação sugere que o impacto nos serviços pode variar de acordo com a natureza específica do *bug*.

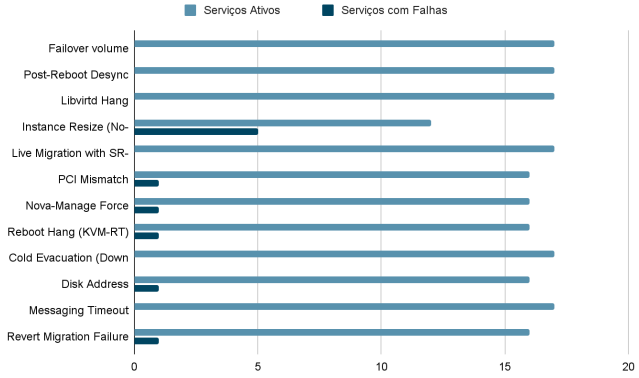

Figure 2. Resultados do experimento com *bugs* do tipo *crash*

A Figura 3 analisa a relação entre *bugs* classificados como *hang* e o status dos serviços. O *bug ed25519 Key Support (Nova)* não impactou o status dos serviços. O sistema manteve 17 serviços ativos. No entanto, esse *bug* causou falhas na criação de instâncias.

Em contraste, outros *bugs* causaram mudanças no status dos serviços. No caso do *SSL Provisioning Failure*, observou-se 12 serviços ativos e 5 com falhas. Uma parcela significativa dos serviços do módulo Nova foi afetada, como `n-cpu`, `n-sch`, `n-cond-cell1`, `n-novnc-cell1` e `n-super-cond`, enquanto apenas os serviços `n-api` e `n-api-meta` permaneceram ativos. Isso representa 71,47% dos serviços do módulo Nova.

O *bug Migration Error (Failed Compute)* provocou a ocorrência de falha no serviço `n-cpu`. Apesar disso, o sistema manteve 16 serviços ativos.

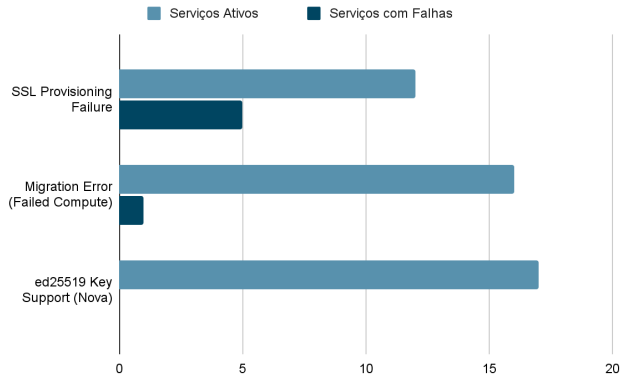

Figure 3. Resultados do experimento com *bugs* do tipo *hang*

A Figura 4 analisa a relação entre *bugs* classificados como funcionalidade incorreta e o status dos serviços no OpenStack. Uma observação importante é a propagação de erros causada por dois *bugs* específicos: *Live Migration fails* e *Trailing Number Launch Block*.

Embora injetados no módulo Nova, esses *bugs* impactaram

Table 3. Bugs classificados

Failover volume	openstack-nova	Volume attached at two instances after failover because compute outage	System Test	Software Configuration	Crash	High
Post-Reboot Desync	openstack-nova	Nova is out of sync with ironic as hypervisor list in nova does not agree with ironic list after reboot of the nodes	System Test	Software Configuration	Crash	Unspecified
Libvirt Hang (RHOSP16)	openstack-nova	[rhosp16] [rt-lxm] Failure in instances build as libvirtd enters uninterruptible state	System Test	Software Configuration	Crash	Urgent
Instance Resize (No-Swap)	openstack-nova	Resizing from flavor with swap to one without swap puts instance into Error status	System Test	Blocked Test	Crash	High
Live Migration with SR-IOV	openstack-nova	[RFE] [NFV] Support live migration of a VM with an SRIOV PF port	System Test	Software Configuration	Crash	High
PCI Mismatch (Host Evacuation)	openstack-nova	nova host-evacuation returns erroneous pci addresses and an error: Unable to correlate PCI slot	System Test	Startup/Restart	Crash	High
Nova-Manage Force Refresh	openstack-nova	Provide a workaround option and/or nova-manage command to force the refresh of connection_info during a hard reboot	System Test	Startup/Restart	Crash	High
Reboot Hang (KVM-RT)	openstack-nova	KVM-RT guest with 10 vCPUs hangs on reboot	System Test	Startup/Restart	Crash	High
Cold Evacuation (Down Hosts)	openstack-nova	[OSP 17] [RFE] Allow admins to cold evacuate instances from down compute hosts	Function Test	Test Interaction	Crash	Medium
Disk Address Persistence (OSP17)	openstack-nova	[RFE] [OSP 17] Persist disk addresses in the bdms of an instance	Design Review	Backward Compatibility	Crash	Medium
Messaging Timeout (BDM Creation)	openstack-nova	MessagingTimeout while in nova.compute. api.API.create_volume_bdm can leave orphaned BDM records in the database	Code Inspection	Concurrency	Crash	Medium
Revert Migration Failure (Faulty Host)	openstack-nova	Revert migration does not work when destination host is faulty	System Test	Recovery/ Exception	Crash	Medium
SSL Provisioning Failure	openstack-nova	Unable to provision an instance after nova.conf is configured for ssl = true in [oslo_messaging_rabbit] section	System Test	Software Configuration	Hang	High
Migration Error (Failed Compute)	openstack-nova	When trying to migrate VMs from a failed compute node, we receive a ResourceProviderCreationFailed traceback	System Test	Blocked Test	Hang	Urgent
ed25519 Key Support (Nova)	openstack-nova	[RFE] [TestOnly] – For inclusion of ed25519 type key pairs in nova	System Test	Recovery/ Exception	Hang	Low
RFE: QEMU VFIO (NVMe)	openstack-nova	RFE: QEMU VFIO based block driver for NVMe devices (OSP)	System Test	Blocked Test	Performance Degradation	Unspecified
nova-compute RP Update Failure	openstack-nova	[OSP16.01] nova-compute error occurred while updating compute node resource provider status to “enabled”	System Test	Startup/Restart	Performance Degradation	Low
Tempest Failures (j626 to 16.1)	openstack-nova	Following an update from j626 to 16.1 multiple tempest tests fail with internal server error	Code Inspection	Backward Compatibility	Performance Degradation	High
Live Migration fails	openstack-nova	Live Migration fails	System Test	Blocked Test	Incorrect Functionality	Unspecified
n-api BDM Validation	openstack-nova	[OSP 17] n-api should reject requests to attach a volume to an instance if an active bdm record exists	System Test	Software Configuration	Incorrect Functionality	Unspecified

NOSTATE after FFU & Stop	openstack-nova	Overcloud node Power State is NOSTATE after FFU and nova stop	System Test	Software Configuration	Incorrect Functionality	Medium
API Compute Server DELETE Issue	openstack-nova	DELETE race	Design Review	Design Conformance	Incorrect Functionality	Urgent
Unshelve Restriction (Migrating/Resize)	openstack-nova	When unshelving an SR-IOV instance, the binding profile isn't reclaimed or rescheduled, and this might cause PCI-PT conflicts	System Test	Recovery/ Exception	Incorrect Functionality	High
Live Migration CPU Mapping	openstack-nova	Live migration of non-NUMA instances does not update cpu_shared_set mapping	Code Inspection	Logic/Flow	Incorrect Functionality	High
Overcommit Launch Conflict (isolcpus)	openstack-nova	[RFE] Nova should refuse to launch overcommit guests on hosts where cpu_dedicated_set overlaps with isolcpus	Code Inspection	Backward Compatibility	Incorrect Functionality	Medium
RFE: Raw Ephemeral Disk	openstack-nova	[RFE] Provide the ability to create ephemeral disk with no filesystem	Function Test	Test Interaction	Incorrect Functionality	Medium
virtio-scsi to virtio-blk (Rescue)	openstack-nova	nova virtio-scsi drivers turned to virtio-blk after rescue mode	System Test	Hardware Configuration	Incorrect Functionality	Medium
Trailing Number Launch Block	openstack-nova	[OSP 16.1] Can't launch instance if name ends with a number	Design Review	Design Conformance	Incorrect Functionality	Medium
RFE: Unique RBD Users (OSP17)	openstack-nova	[OSP 17] [RFE] [rbd] Create a unique rbd user for each host volume attachment	Design Review	Side Effects	Incorrect Functionality	Unspecified
Unshelve Scheduling Issue (SR-IOV)	openstack-nova	Unshelved instance can't migrate and resize	System Test	Blocked Test	Incorrect Functionality	Medium
Snapshot Deletion Error (Cinder NFS)	openstack-nova	Snapshot cannot be deleted when the instance is shutoff after multiple snapshots creation if Cinder nfs backend is used	System Test	Startup/Restart	Incorrect Functionality	Medium

o módulo Cinder, impedindo a criação de volumes. Ambos alteraram o status do serviço `c-vol` para falha. O *bug Trailing Number Launch Block* apresentou um comportamento mais crítico: a aba de volumes desapareceu do ambiente de nuvem e a reversão do código para a versão original não restaurou o módulo Cinder. Para continuar os testes, foi necessário provisionar um novo ambiente.

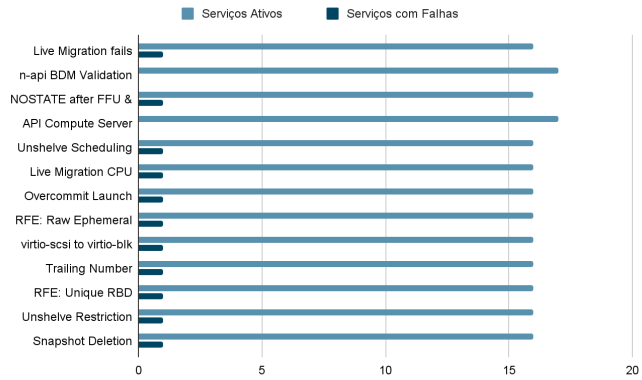

Figure 4. Resultados do experimento com *bugs* do tipo Funcionalidade Incorreta

A análise da Figura 4 também revela diferentes padrões de impacto nos serviços. O *bug Overcommit Launch Conflict (isolcpus)* causou um erro de status falha no serviço `n-sch`. Um grupo de *bugs*, incluindo *NOSTATE after FFU & Stop*, *Unshelve Scheduling Issue (SR-IOV)*, *Live Migration CPU Mapping Error*, *RFE: Raw Ephemeral Disk*, *virtio-scsi to virtio-blk (Rescue)*, *RFE: Unique RBD Users (OSP17)*, *Unshelve Restriction (Migration/Resize)* e *Snapshot Deletion Error (Cinder NFS)*, deixaram o serviço `n-cpu` com o status de falha.

Por fim, os *bugs n-api BDM Validation* e *API Compute Server DELETE Issue* se destacaram por não alterarem o status dos serviços. No entanto, esses *bugs* causaram erros na criação de instâncias, indicando problemas funcionais no OpenStack.

A Figura 5 analisa a distribuição de serviços ativos, com falhas e inativos para *bugs* classificados como Degradação de Desempenho. O *bug RFE: QEMU VFIO (NVMe)* se destaca por apresentar um status inativo no serviço `n-cpu`. Esse status indica que o serviço não foi iniciado devido ao *bug*, representando uma interrupção completa da funcionalidade. Em contraste, os *bugs nova-compute RP Update Failure* e *Tempest Failures (16z2 to 16.1)* apresentaram um status de falha no serviço `n-cpu`. O status de falha sugere que o serviço foi iniciado, mas encontrou erros durante sua execução, resultando em uma degradação do desempenho.

A Figura 6 apresenta um agrupamento de todos os *bugs* analisados.

Dentre eles, apenas um *bug* não resultou em falhas após a injeção da mutação no código. Os resultados obtidos podem ser classificados em duas categorias: *bugs* com erros

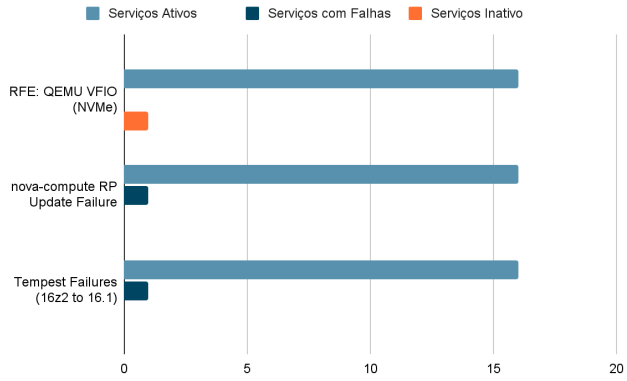

Figure 5. Resultados do experimento com *bugs* do tipo Degradação de Desempenho

explícitos, que causaram erros evidentes refletidos no status dos serviços do módulo Nova, e *bugs* com impacto funcional, que, apesar de não apresentarem falhas explícitas no status dos serviços, impediram o uso de funcionalidades específicas da nuvem, como a criação de máquinas virtuais.

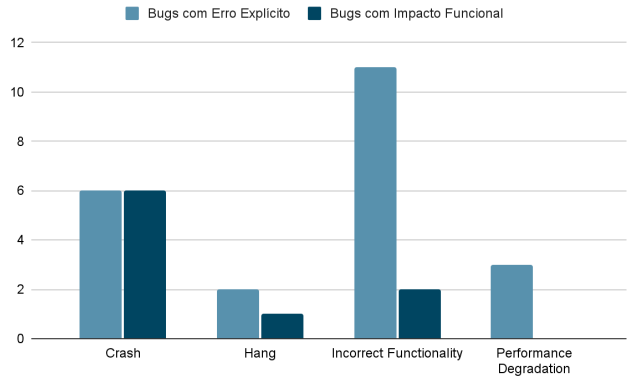

Figure 6. Resultados dos experimentos: classificação por *bugs*

Dentre os tipos de falha, *Crash* apresentou uma distribuição equilibrada (50%) entre *bugs* com impacto funcional e erros explícitos. *Hang* exibiu maior incidência de erros explícitos (66%) em relação aos *bugs* de impacto funcional (34%). Funcionalidade Incorreta apresentou predominância de erros explícitos (84,62%) em comparação aos *bugs* de impacto funcional (15,38%). A Degradação de Desempenho teve 100% das falhas classificadas como erros explícitos.

Para complementar essa análise quantitativa dos tipos de falha, buscou-se validar qualitativamente a relevância das falhas injetadas pela ferramenta CIMut através da comparação com *bugs* reais do OpenStack, catalogados no Bugzilla. Observou-se que, embora nem todas as falhas simuladas tenham replicado de forma idêntica os cenários descritos nos relatórios, muitos dos comportamentos anômalos induzidos (como falhas na criação de instâncias ou instabilidade no painel de controle) são consistentes com os problemas frequentemente reportados pela comunidade.

Discussão: status de serviço versus impacto funcional
Os resultados apresentados nesta seção revelam um padrão recorrente que merece análise mais detalhada: uma parcela significativa dos *bugs* injetados não alterou o status reportado pelos serviços do `systemctl`, ainda que tenha comprometido o funcionamento esperado da nuvem (por exemplo, impedindo a criação de instâncias ou desestabilizando o painel de controle). Essa discrepância entre *status de serviço* e *impacto funcional* tem natureza estrutural e exige atenção tanto na interpretação dos resultados quanto no projeto de mecanismos de monitoramento de ambientes de nuvem.

O status reportado pelo `systemctl` responde essencialmente a duas condições: (i) o processo está em execução? e (ii) seu código de saída foi bem-sucedido? Nenhuma dessas condições verifica a *corretude semântica* das respostas que o processo produz. Mutações que afetam a lógica de negócio do serviço, como ramos condicionais alterados, validações de entrada removidas, ordenação modificada ou atribuições trocadas, tipicamente preservam o processo vivo e em execução, mas corrompem o resultado das requisições. O serviço continua respondendo às chamadas de API; apenas responde de forma incorreta.

Esse fenômeno é coerente com observações anteriores da literatura. Natella et al. [25] já haviam apontado que falhas residuais em sistemas reais frequentemente se manifestam como comportamento incorreto sem causar terminação abrupta do processo, justamente o padrão observado em parte dos *bugs* deste estudo. Os resultados aqui reportados reforçam essa observação no contexto específico de uma nuvem privada baseada em OpenStack.

Três implicações práticas derivam dessa observação. Em primeiro lugar, o monitoramento baseado exclusivamente em status de processo, comum em ambientes operacionais, é insuficiente para detectar uma classe importante de falhas. Mecanismos complementares de *observabilidade funcional*, tais como verificação de invariantes nas respostas das APIs, validação contínua de fluxos fim-a-fim e testes sintéticos periódicos, são necessários para uma cobertura adequada. Em segundo lugar, a propagação de falhas observada do módulo Nova para o Cinder (nos *bugs Live Migration fails* e *Trailing Number Launch Block*) demonstra que mesmo arquiteturas declaradamente modulares apresentam acoplamento operacional não-óbvio, o que reforça a importância de estudos de injeção de falhas para mapear essas dependências. Em terceiro lugar, a irreversibilidade observada em um dos cenários experimentais, onde foi necessário reprovisionar o ambiente para continuar os testes, aponta para a necessidade de estratégias robustas de *checkpoint* e *rollback* em ambientes de produção, especialmente em operações que envolvem múltiplos módulos interdependentes.

Em conjunto, essas observações posicionam o CIMut não apenas como uma forma de medir resiliência a falhas isoladas, mas como uma ferramenta de diagnóstico para identificar lacunas no monitoramento operacional e no isolamento entre

componentes em ambientes de nuvem.

5.3 Comparação entre o CIMut e Outras Soluções de Injeção de Falhas por Mutação de Código

As ferramentas G-SWIFT [24], PREFAIL [36], ProFIPy [38] e CIMut realizam injeção de falhas através da modificação de código. Entretanto, apenas CIMut, PREFAIL e ProFIPy direcionam-se a ambientes de nuvem. Dessas, somente CIMut e ProFIPy injetam falhas na plataforma de nuvem OpenStack. O CIMut foi aplicado para injetar 31 falhas reportadas pelos usuários da plataforma OpenStack, as quais foram classificadas com a técnica ODC.

Embora G-SWIFT, PREFAIL e ProFIPy tenham sido propostas para injeção de falhas por mutação de código, nenhuma delas disponibiliza seu código completo ou implementações funcionais, o que inviabiliza a aplicação efetiva dos experimentos por terceiros. O G-SWIFT não oferece acesso ao código da solução; o PREFAIL apresenta apenas fragmentos isolados de código ao longo do artigo [36]; e o ProFIPy limita-se a exibir um diagrama arquitetural da solução. Em contraste, o CIMut é acessível publicamente como aplicação web em ⟨https://cimut.vercel.app/⟩, com arquitetura em três camadas, interface gráfica de configuração e monitoramento integrado de infraestrutura. O código-fonte encontra-se em processo de registro de patente e será disponibilizado publicamente após a conclusão desse processo. Mesmo sem acesso ao código, a ferramenta é utilizável por terceiros para configurar e executar experimentos de injeção de falhas em ambientes de nuvem.

Adicionalmente, a Tabela 5 expande essa comparação para aspectos metodológicos relevantes, com base nas descrições publicadas nos artigos originais de cada solução. Optou-se por uma comparação qualitativa e metodológica, em vez de uma comparação numérica direta de resultados, pelos seguintes motivos: (i) ProFIPy [38] utiliza operadores de mutação sintáticos genéricos como fonte de falhas, enquanto o CIMut parte de *bugs* reais classificados pela metodologia ODC; (ii) PREFAIL [36] foi avaliada em sistemas distribuídos como HDFS e ZooKeeper, não em plataformas de nuvem IaaS; e (iii) G-SWIFT [24] foca em programas escritos em C, em contexto distinto de ambientes de nuvem. Dada essa heterogeneidade de métricas, fontes de falhas e ambientes de avaliação, uma comparação numérica direta entre os resultados publicados seria metodologicamente questionável. A comparação metodológica apresentada na Tabela 5 fornece, portanto, uma base mais sólida para situar o CIMut em relação às soluções correlatas. As limitações dessa abordagem comparativa são discutidas na Subseção 5.4.

5.4 Ameaças à Validade

Esta subseção discute as principais ameaças à validade do estudo experimental conduzido, organizadas em quatro categorias: validade de construto, validade interna, validade externa e validade de conclusão. Para cada ameaça, são apresentadas as medidas de mitigação adotadas e, quando aplicável, encaminhamentos para trabalhos futuros.

Validade de Construto

A principal ameaça à validade de construto está relacionada à representatividade das falhas injetadas. Mutações sintáticas no código-fonte (como erros de indentação, remoção de parâmetros ou substituições que violem a estrutura do código) não reproduzem necessariamente os mecanismos exatos pelos quais os *bugs* originais se manifestam em produção. Para mitigar essa ameaça, a seleção das mutações foi guiada por *bugs* reais reportados no Red Hat Bugzilla [50], previamente classificados pela metodologia ODC [51], em vez de mutações sintéticas arbitrárias. Conforme apresentado na Subseção 5.2, observou-se que muitos dos comportamentos anômalos induzidos foram consistentes com os relatos da comunidade, o que aumenta a confiança na representatividade, embora não a garanta em todos os casos. Essa limitação é coerente com observações da literatura sobre representatividade em injeção de falhas de software [25].

Validade Interna

Três fatores constituem ameaças à validade interna do estudo. O primeiro é a **seleção manual da linha a ser mutada**: a partir do relato do *bug* e da análise do código relacionado, a linha de código alvo foi escolhida por inspeção manual. Essa abordagem introduz subjetividade no processo. Para mitigar esse risco em trabalhos futuros, planeja-se a adoção de mecanismos baseados em processamento de linguagem natural para identificar automaticamente trechos candidatos a partir da descrição textual do *bug*, conforme detalhado na Seção 6.

O segundo fator é a **configuração *all-in-one* do ambiente**, em que um único nó executa simultaneamente as funções de Controlador, Computação e Rede. Essa configuração simplifica a reprodutibilidade, mas pode mascarar ou amplificar efeitos que apareceriam de forma distinta em infraestruturas distribuídas. Estudos complementares em configurações multi-nó estão previstos como trabalho futuro.

O terceiro fator é a **carga de trabalho simplificada** adotada nos experimentos, baseada na criação e exclusão de instâncias com configuração mínima (1 vCPU, 1 GB de RAM) e sem volumes anexados. Embora essa carga seja suficiente para acionar os caminhos de código mutados, ela não representa cenários de produção com múltiplas instâncias, volumes persistentes e carga concorrente intensa. A ampliação da carga de trabalho é encaminhada como trabalho futuro.

Validade Externa

A principal ameaça à validade externa é o **foco exclusivo no módulo Nova do OpenStack**. A escolha desse módulo foi motivada por dois critérios: (i) sua centralidade na arquitetura do OpenStack, uma vez que o Nova gerencia computação, o recurso mais frequentemente provisionado em nuvens IaaS; e (ii) o maior volume de *bugs* catalogados para esse módulo no Bugzilla, o que aumenta a base empírica disponível. No entanto, os resultados específicos sobre quais serviços apresentam status de falha ou impacto funcional não se generalizam diretamente para outros módulos (Neutron, Cinder, Glance,

Table 4. Comparação entre soluções de injeção de falhas com mutação de código

Ferramenta	Classificação ODC	Ambiente de Nuvem	Plataforma OpenStack	Ferramenta Acessível
CIMut	✓	✓	✓	✓
G-SWIFT	—	—	—	—
PREFAIL	—	✓	—	—
ProFIPy	—	✓	✓	—

Table 5. Comparação metodológica entre soluções de injeção de falhas por mutação de código

Aspecto	CIMut	ProFIPy	PREFAIL	G-SWIFT
Fonte das falhas	*Bugs* reais (Bugzilla) classificados por ODC	Operadores de mutação sintáticos	Composição de múltiplas falhas	Operadores G-SWFIT em código binário
Classificação ODC dos *bugs* injetados	Sim	Não	Não	Não
Ambiente de avaliação	OpenStack (módulo Nova)	OpenStack	HDFS, ZooKeeper, Cassandra	Programas em C de propósito geral
Métrica principal	Status de serviço e impacto funcional	Cobertura de operadores de mutação	Detecção de falhas múltiplas correlacionadas	Taxa de ativação de falhas injetadas
Ferramenta acessível a terceiros	Sim (aplicação web pública)	Não	Não	Não

Keystone) nem para outras plataformas de nuvem (CloudStack, OpenNebula, Eucalyptus). Para mitigar essa ameaça em trabalhos futuros, planeja-se uma amostragem estratificada por módulo do OpenStack e a aplicação do CIMut a outras plataformas de nuvem. Vale observar que o CIMut é, por construção, agnóstico de plataforma, opera diretamente sobre arquivos de código-fonte do ambiente de nuvem sob teste, sem dependências específicas do OpenStack, o que sustenta a expectativa de aplicabilidade em outras plataformas, ainda que os resultados quantitativos específicos observados neste estudo sejam particulares ao OpenStack/Nova e demandem validação empírica adicional em outros contextos.

Validade de Conclusão

A ausência de **comparação experimental *head-to-head* com outras soluções de injeção de falhas por mutação de código** é uma ameaça à validade de conclusão. Como discutido na Subseção 5.3, essa comparação não foi conduzida neste trabalho por duas razões complementares. A primeira é a indisponibilidade de implementações funcionais: G-SWIFT [24], PREFAIL [36] e ProFIPy [38] não disponibilizam código executável, o que inviabiliza, no momento, a reprodução de seus experimentos sobre os mesmos cenários adotados aqui. A segunda razão é de ordem metodológica: as soluções correlatas foram avaliadas em contextos significativamente distintos, diferentes fontes de falhas, ambientes e métricas (Tabela 5), o que tornaria uma comparação numérica direta entre os resultados publicados metodologicamente questionável. Por essas razões, a comparação realizada neste trabalho é de natureza qualitativa e metodológica. Como trabalho futuro, planeja-se conduzir uma comparação experimental direta no momento em que o código de ProFIPy, a solução mais próxima em termos de objetivo e ambiente, seja disponibilizado publicamente, ou através de uma reimplementação aproximada baseada na descrição publicada no artigo original.

Outra ameaça à validade de conclusão é o **número de replicações por *bug*** (duas replicações), o que limita a análise estatística da variância dos resultados. Embora esse número tenha sido suficiente para observar os padrões qualitativos descritos na Subseção 5.2, replicações adicionais fortaleceriam a confiança nas proporções reportadas (*e.g.*, 96,7% de falhas geradas).

6. Conclusão

Este trabalho apresentou a avaliação de dependabilidade de um ambiente de nuvem configurado com a plataforma OpenStack por meio do CIMut, uma ferramenta de injeção de falhas por mutação de código-fonte acessível publicamente como aplicação web em ⟨https://cimut.vercel.app/⟩. Um estudo de caso baseado no módulo Nova da plataforma OpenStack foi conduzido para avaliação da ferramenta. Por meio da ferramenta proposta, foram injetados 31 *bugs* classificados pela metodologia *Orthogonal Defect Classification* (ODC), distribuídos da seguinte forma: 12 *crash*, 3 *hang*, 3 de degradação de desempenho e 13 de funcionalidade incorreta.

A análise dos *bugs* revelou impactos variados nos serviços do OpenStack. Enquanto alguns não alteraram o status dos

serviços, outros resultaram em erros com status de falha ou inativo, indicando interrupções parciais ou completas. O impacto variou de acordo com o *bug* específico, com alguns afetando apenas um serviço e outros impactando vários simultaneamente. Os resultados demonstraram que 96,7% das falhas injetadas resultaram em *bugs*, classificados como erros explícitos (travamentos, exceções) ou *bugs* com impacto funcional (comportamento incorreto, perda de dados), evidenciando a capacidade da ferramenta em gerar falhas representativas em ambientes de nuvem.

É importante destacar que, mesmo nos casos em que o status dos serviços permaneceu inalterado, alguns *bugs* causaram problemas funcionais durante operações como a criação de instâncias. Essa discrepância entre o status do serviço e o impacto funcional, já apontada em trabalhos anteriores [25, 22], demonstra a necessidade de novos métodos de monitoramento em ambientes de nuvem.

Três implicações práticas derivam dos achados deste estudo. A primeira é que o monitoramento baseado exclusivamente em status de processo é insuficiente para detectar falhas que comprometem a corretude semântica das respostas, sendo necessária a adoção de mecanismos de observabilidade funcional, como verificação de invariantes nas respostas das APIs e testes sintéticos periódicos. A segunda é que a propagação de falhas observada entre módulos do OpenStack (notavelmente do Nova para o Cinder) demonstra que arquiteturas declaradamente modulares apresentam acoplamento operacional não-óbvio, o que justifica estudos contínuos de injeção de falhas para mapear essas dependências. A terceira é que a irreversibilidade observada em determinados cenários experimentais aponta para a necessidade de estratégias robustas de *checkpoint* e *rollback* em ambientes de produção, especialmente em operações que envolvem múltiplos módulos interdependentes.

Para aprimorar a ferramenta proposta, os trabalhos futuros se concentrarão em quatro frentes complementares. A primeira é a evolução do algoritmo de injeção com mecanismos de decisão baseados em inteligência artificial, permitindo a injeção de falhas em locais mais estratégicos e relevantes para a análise do ambiente de nuvem. Pretende- se utilizar processamento de linguagem natural para interpretar comandos do usuário e automaticamente identificar os trechos de código-fonte do OpenStack onde as falhas devem ser introduzidas, mitigando a subjetividade da seleção manual de linhas discutida na Subseção 5.4.

A segunda frente é a ampliação do escopo experimental, incluindo: (i) amostragem estratificada por módulo do OpenStack (Neutron, Cinder, Glance, Keystone) e por categoria ODC, garantindo cobertura proporcional dos tipos de falha; (ii) execução em infraestruturas distribuídas multi-nó, em substituição à configuração *all-in-one* adotada neste estudo; e (iii) uso de cargas de trabalho mais representativas de cenários de produção, com múltiplas instâncias concorrentes, volumes persistentes e operações intensivas de rede.

A terceira frente é a validação empírica da generalidade da abordagem CIMut em outras plataformas de nuvem. O CIMut é, por construção, agnóstico de plataforma, opera diretamente sobre arquivos de código-fonte do ambiente de nuvem sob teste, sem dependências específicas do OpenStack. Os experimentos deste trabalho validaram a ferramenta no contexto do OpenStack; planeja-se conduzir estudos de caso adicionais em outras plataformas de nuvem *open source*, como Apache CloudStack [14], OpenNebula [15] e Eucalyptus [17], bem como em orquestradores de contêineres como Kubernetes. Essa extensão permitirá verificar se os padrões observados no OpenStack, notavelmente a discrepância entre status de serviço e impacto funcional, se repetem em outros contextos arquiteturais.

A quarta frente é a condução de uma comparação experimental direta com soluções correlatas de injeção de falhas por mutação de código, em particular a ProFIPy [38], quando seu código for disponibilizado publicamente, ou através de uma reimplementação aproximada baseada na descrição publicada. Essa comparação fornecerá evidências quantitativas adicionais sobre cobertura, representatividade das falhas geradas e impacto operacional, complementando a comparação qualitativa apresentada na Subseção
5.3.

Contribuição dos Autores

Guilherme Silva Duarte foi responsável pela conceitualização do trabalho, design do algorítmo de injeção de falhas, desenvolvimento da ferramenta CIMut, execução dos experimentos no ambiente OpenStack, curadoria e análise dos dados obtidos e escrita do manuscrito; Erica Teixeira Gomes de Sousa foi responsável pela supervisão acadêmica da pesquisa, revisão crítica da metodologia e resultados experimentais e revisão do manuscrito;.

References

[1] BRADLEY, A. *Cloud computing transformation: leveraging AI for predictive resource management*. 2023. ResearchGate. Disponível em: ⟨https://www.researchgate.net/publication/385781616⟩. Acesso em: 24 maio 2025.

[2] SAYEGH, E. *How cloud computing revolutionized business operations and what lies ahead*. 2023. Forbes. Disponível em: ⟨https://www.forbes.com/sites/emilsayegh/2023/11/28/how-cloud-computing-revolutionized-business-operations-and-what-lies-ahead/⟩. Acesso em: 24 maio 2025.

[3] CLOUD computing and firm performance: Evidence from IT investment. 2023. International Review of Financial Analysis. Disponível em: ⟨https://l1nq.com/mtREk⟩. Acesso em: 24 maio 2025.

[4] RED HAT. *Red Hat Recognized as a Leader in 2024 Gartner® Magic Quadrant™ for Cloud Application Platforms*. 2024. Press release.

[5] LI, Y. et al. Predicting node failures in an ultra-large-scale cloud computing platform: An AIOps solution. *ACM Transactions on Software Engineering and Methodology*, v. 29, n. 2, 2020.

[6] ABED, A. H. The techniques of authentication in the context of cloud computing. *International Journal of Advanced Networking and Applications*, v. 16, n. 04, p. 6515–6522, 2025.

[7] AHMAD, A. A.-S.; ANDRAS, P. Scalability resilience framework using application-level fault injection for cloud-based software services. *Journal of Cloud Computing: Advances, Systems and Applications*, v. 11, n. 1, p. 1–17, 2022. Disponível em: ⟨https://encr.pw/Yp3Sy⟩. Acesso em: 24 maio 2025.

[8] COTRONEO, D. et al. *Fault Injection Analytics: A Novel Approach to Discover Failure Modes in Cloud-Computing Systems*. 2020. ArXiv preprint arXiv:2010.00331. Disponível em: ⟨https://arxiv.org/abs/2010.00331⟩. Acesso em: 24 maio 2025.

[9] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY (NIST). *The NIST Definition of Cloud Computing*. [S.l.], 2011. Special Publication 800-145. Disponível em: ⟨https://csrc.nist.gov/publications/detail/sp/800-145/final⟩. Acesso em: 24 maio 2025.

[10] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION (ISO). *Information technology – Cloud computing – Part 1: Vocabulary*. [S.l.], 2023. ISO/IEC 22123-1:2023(E).

[11] KHOMCHAK, M. Enterprise private cloud platforms: A systematic review of key vendors. *International Journal of Wireless and Microwave Technologies*, v. 14, n. 4, p. 1–14, 2024.

[12] TOZZI, C. *Why Private Clouds May See a Resurgence in 2024*. 2024. ITPro Today. Acesso em: 24 maio 2025.

[13] RED HAT. *OpenShift*. 2025. ⟨https://www.openshift.com/learn/what-is-openshift⟩. Acesso em: 19 fev. 2025.

[14] APACHE CloudStack: Open-Source Cloud Computing. 2025. Apache CloudStack website. Acesso em: 24 maio 2025.

[15] OpenNebula 2023: Year in Review. 2024. Blog OpenNebula Systems. Acesso em: 24 maio 2025.

[16] Cloudify: Open-Source Multi-Cloud Orchestration Platform. 2024. ⟨https://docs.cloudify.co/latest/about/what-is-cloudify/⟩. Cloudify Documentation Center. Acesso em: 23 maio 2026.

[17] NURMI, D. et al. The Eucalyptus open-source cloud-computing system. In: *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID)*. Shanghai, China: IEEE, 2009. p. 124–131. DOI: 10.1109/CCGRID.2009.93.

[18] CHOWDHURY, D. D. Cloud networking. In: *Future of Networks: Modern Communication Infrastructure*. Cham: Springer Nature Switzerland, 2025. p. 79–123.

[19] VANKAYALAPATI, R. K. The future of hybrid cloud: Trends, technologies, and predictions. In: *The Synergy Between Public and Private Clouds in Hybrid Infrastructure Models: Real-World Case Studies and Best Practices*. [S.l.: s.n.], 2025. p. 148.

[20] OPEN Source Cloud Computing Platform Software – OpenStack. 2024. Website. Disponível em: ⟨https://www.openstack.org/software/project-navigator/openstack-components⟩. Acesso em: 12 fev. 2024.

[21] MACIEL, P. *Performance, Reliability, and Availability Evaluation of Computational Systems. Volume I: Performance and Background*. Boca Raton: CRC Press, 2022. DOI: 10.1201/9781003306016.

[22] NATELLA, R.; COTRONEO, D.; MADEIRA, H. S. Assessing dependability with software fault injection: A survey. *ACM Computing Surveys*, v. 48, n. 3, p. 44:1–44:55, fev. 2016. DOI: 10.1145/2841425.

[23] SHUVO, A. M. et al. *A Comprehensive Survey on Non-Invasive Fault Injection Attacks*. 2023. Cryptology ePrint Archive, Paper 2023/1769. Disponível em: ⟨https://eprint.iacr.org/2023/1769⟩. Acesso em: 24 maio 2025.

[24] DURAES, J. A.; MADEIRA, H. S. Emulation of software faults: A field data study and a practical approach. *IEEE Transactions on Software Engineering*, v. 32, n. 11, p. 849–867, nov. 2006. DOI: 10.1109/TSE.2006.113.

[25] NATELLA, R. et al. On fault representativeness of software fault injection. *IEEE Transactions on Software Engineering*, v. 39, n. 1, p. 80–96, jan. 2013. DOI: 10.1109/TSE.2011.124.

[26] KUMAR, S.; MUTTOO, S. K.; SINGH, V. B. Classification of software defects using orthogonal defect classification. *International Journal of Open Source Software and Processes*, v. 13, n. 1, p. 1–16, 2022. DOI: 10.4018/IJOSSP.300749.

[27] BANABIC, R.; CANDEA, G. Fast black-box testing of system recovery code. In: *Proceedings of the 7th ACM European Conference on Computer Systems – EuroSys '12*. [S.l.: s.n.], 2012.

[28] MEIKLEJOHN, C. *Resilient Microservice Applications, by Design, and without the Chaos*. Tese (Doutorado) — Carnegie Mellon University, 2024.

[29] KAO, W.; IYER, R. K. DEFINE: A distributed fault injection and monitoring environment. In: *Proceedings of IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*. College Station, TX, USA: [s.n.], 1994. p. 252–259. DOI: 10.1109/FTPDS.1994.494497.

[30] DAI, Y.; LIU, S.; LIU, H. Mutation-based approach to supporting human-machine pair inspection. *Electronics*, v. 14, n. 2, p. 382, 2025.

[31] SOUZA, D. et al. EucaBomber: Experimental evaluation of availability in Eucalyptus private clouds. In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*. Manchester, UK: IEEE, 2013. p. 4080–4085. DOI: 10.1109/SMC.2013.696.

[32] ARAUJO, J. et al. Availability and reliability modeling of mobile cloud architectures. *IEEE Transactions on Industrial Informatics*, 2023.

[33] SEGALL, Z. et al. FIAT – fault injection based automated testing environment. In: *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, 'Highlights from Twenty-Five Years'*. Pasadena, CA, USA: IEEE, 1995. p. 394. DOI: 10.1109/FTCSH.1995.532663.

[34] AMYAN, A. et al. Automating fault test cases generation and execution for automotive safety validation via NLP and HIL simulation. *Sensors*, v. 24, n. 10, p. 3145, 2024.

[35] LATTNER, C. et al. MLIR: Scaling compiler infrastructure for domain specific computation. In: *Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. [S.l.]: IEEE, 2021. p. 2–14.

[36] JOSHI, P.; GUNAWI, H. S.; SEN, K. PREFAIL: A programmable tool for multiple-failure injection. *SIGPLAN Notices*, v. 46, n. 10, p. 171–188, out. 2011.

[37] KHAN, H. U.; ALI, F.; NAZIR, S. Systematic analysis of software development in cloud computing perceptions. *Journal of Software: Evolution and Process*, v. 36, n. 2, p. e2485, 2024.

[38] COTRONEO, D. et al. ProFIPy: Programmable software fault injection as-a-service. In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Valencia, Spain: IEEE, 2020. p. 364–372. DOI: 10.1109/DSN48063.2020.00052.

[39] SALIH, N. K.; GOPAL, R. Optimization software fault injection by using two-dimensional bin packing algorithms. *Journal of Electrical Systems*, v. 20, n. 10s, p. 6638–6645, 2024.

[40] YALCIN, G. et al. FIMSIM: A fault injection infrastructure for microarchitectural simulators. In: *2011 IEEE 29th International Conference on Computer Design (ICCD)*. Amherst, MA, USA: IEEE, 2011. p. 431–432. DOI: 10.1109/ICCD.2011.6081435.

[41] ASADOVA, F.; KERTESZ, G.; LOVAS, R. Simulation-based network fault injection in the CloudSim Plus cloud simulation environment. *Azerbaijan Journal of High Performance Computing*, v. 6, n. 1, 2023.

[42] CHOI, G. S.; IYER, R. K. FOCUS: An experimental environment for fault sensitivity analysis. *IEEE Transactions on Computers*, v. 41, n. 12, p. 1515–1526, dez. 1992. DOI: 10.1109/12.214660.

[43] KANAWATI, G. A.; KANAWATI, N. A.; ABRAHAM, J. A. FERRARI: A tool for validating system dependability properties. In: *Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*. Boston, MA, USA: IEEE, 1992. p. 336–344. DOI: 10.1109/FTCS.1992.243567.

[44] TSAI, T.; IYER, R. K. Measuring fault tolerance with the FTAPE fault injection tool. In: *Dependable Computing – EDCC-3*. [S.l.]: Springer Berlin Heidelberg, 1999, (Lecture Notes in Computer Science, v. 1667). p. 26–43. DOI: 10.1007/BFb0024305.

[45] ARLAT, J. et al. Fault injection for dependability validation: A methodology and some applications. *IEEE Transactions on Software Engineering*, v. 16, n. 2, p. 166–182, fev. 1990.

[46] MADEIRA, H. et al. RIFLE: A general purpose pin-level fault injector. In: ECHTLE, K.; HAMMER, D.; POWELL, D. (Ed.). *Dependable Computing – EDCC-1*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, (Lecture Notes in Computer Science, v. 852). p. 197–216. DOI: 10.1007/3-540-58426-9_132.

[47] SALIH, N. K. et al. A survey on software/hardware fault injection tools and techniques. In: *2022 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*. [S.l.]: IEEE, 2022. p. 1–7.

[48] CARREIRA, J.; MADEIRA, H.; SILVA, J. Xception: Software fault injection and monitoring in processor functional units. In: *Proceedings of the 5th IFIP Working Conference on Dependable Computing for Critical Applications*. [S.l.: s.n.], 2001.

[49] COTRONEO, D.; SIMONE, L. D.; NATELLA, R. ThorFI: A novel approach for network fault injection as a service. *arXiv preprint arXiv:2201.07521*, 2022.

[50] RED Hat Bugzilla. 2024. ⟨https://bugzilla.redhat.com/⟩. Acesso em: 12 fev. 2024.

[51] CHEN, E. et al. Application of orthogonal defect classification for software reliability analysis. In: *16th International Conference on Probabilistic Safety Assessment and Management (PSAM 2022)*. Honolulu, United States: [s.n.], 2022.