UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA - CIn
BACHARELADO EM SISTEMA DE INFORMAÇÃO

ELIARDO CLÁUDIO FERREIRA

# FRAMEWORK DE INJEÇÃO DE FALHAS E MONITORAMENTO

RECIFE
2018

ELIARDO CLÁUDIO FERREIRA

# FRAMEWORK DE INJEÇÃO DE FALHAS E MONITORAMENTO

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Sistemas da Informação da Universidade Federal de Pernambuco como requisito à obtenção do título de Bacharel em Sistemas da Informação.

Orientador: Prof. Dr. Paulo Romero Martins Maciel

RECIFE
2018

ELIARDO CLÁUDIO FERREIRA

# FRAMEWORK DE INJEÇÃO DE FALHAS E MONITORAMENTO

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Sistemas da Informação da Universidade Federal de Pernambuco como requisito à obtenção do título de Bacharel em Sistemas da Informação.

Paulo Romero Martins Maciel
Orientador - Departamento de Sistemas da Informação (UFPE)

_____

Sergio Ricardo de Melo Queiroz
Avaliador - Departamento de Sistemas da Computação (UFPE)

_____

Recife, 17, Dezembro de 2018

# RESUMO

Em decorrência da evolução dos sistemas computacionais, surge a necessidade do aumento de pesquisas que tenham por objetivo oferecer aos usuários sistemas de fácil usabilidade com funcionalidades diversificadas, confiáveis, de baixo custo e de alto desempenho. Podemos considerar que o desempenho e a confiabilidade são características importantes para o desenvolvimento de um sistema, visto que toda organização preza em assegurar que os serviços providos aos seus usuários sejam confiáveis e de alto desempenho, tornando-se, dessa maneira, prioridade para sobrevivência dos sistemas atuais, bem como um requisito fundamental para os novos sistemas a serem estabelecidos. Apoiando-se nesse contexto, surge a necessidade do desenvolvimento de uma ferramenta de geração de eventos sintéticos com o intuito de contribuir para a realização de experimentos nas áreas de avaliação de desempenho e dependabilidade de sistemas.
Neste trabalho será desenvolvida uma ferramenta injetora de falhas e reparos que tem como objetivo injetar falhas e reparos em ambientes de teste, neste trabalho os testes serão realizados através da ferramenta Eucalyptus, mas a ferramenta pode ser utilizada em outros ambientes que suportam essa funcionalidade sem a necessidade de mudanças em seu código fonte, apenas adicionando os comandos de falha e reparo em sua interface.

**Palavras-chave:** Injetor de falhas, Avaliação da Dependabilidade, Eucalyptus, Computação em Nuvem

# ABSTRACT

With the evolution of computational systems, it arises the need to increase the number of researches that aim to offer to its user's easy-to-use systems with a variety of features, such as reliability, low cost and high performance. We can consider that performance and reliability are fundamental features to develop a system, once every single organization values making sure that the services provided to its users are reliable and also presents a high performance, which has become a priority for current systems survival and also a crucial requirement to new systems that will be established. Based on this context, it arose the need to develop a tool to generate synthetic events in order to contribute to execute experiments in the fields of performance evaluation and dependability systems.
In this work, a fault injection and repair tool will be developed that aims to inject faults and repairs in test environments, in this work the tests will be performed through the Eucalyptus tool, but the tool can be used in other environments that support this functionality without the need to change its source code, just adding the fault and repair commands in its interface.

**Keywords:** Fault Injector, Dependability Assessment, Eucalyptus, Cloud Computing

# SUMÁRIO

**RESUMO** ..... 3
**ABSTRACT** ..... 4
**LISTA DE FIGURAS** ..... 7
**1 Introdução** ..... 8
1.1 Objetivos ..... 9
1.2 Trabalhos Relacionados ..... 10
1.3 Estrutura do Trabalho ..... 13
**2 Fundamentos** ..... 14
2.1 Computação em Nuvem ..... 14
2.1.1 Vantagens ..... 14
2.1.2 Maior controle dos Gastos com Software ..... 15
2.1.3 Maior mobilidade ..... 15
2.1.4 Ganho de escalabilidade ..... 15
2.1.5 Riscos ..... 16
2.1.6 Infraestrutura como serviço (IaaS) ..... 16
2.1.7 Eucalyptus ..... 18
2.2 Injeção de Falhas e Monitoramento de Defeitos ..... 19
2.2.1 Injeção de Falhas ..... 20
2.2.2 Injeção de Falhas de Comunicação ..... 21
2.2.3 Ferramentas de teste para Injeção de Falhas ..... 23
2.3 Ferramentas de Injeção de Falhas ..... 24
2.3.1 Injeção de Falhas para Avaliação de Segurança ..... 28
2.3.2 Ferramentas para Injeção de Falhas em Segurança de Redes ..... 29
2.3.3 Ferramentas para Injeção de Pacotes ..... 30
2.3.4 Ferramentas para Varredura de Vulnerabilidade ..... 30
**3 Protótipo do Injetor** ..... 32
3.1 Visão Geral do Injetor de Falhas ..... 34
3.1.1 Estrutura do Injetor ..... 35
3.1.2 Obtendo o Tempo de Falha e Reparo (MTBF e MTTR) ..... 37

3.1.3 (Mean Time Between Failures) – MTBF ..... 38
3.1.4 (Mean Time to Repair) - MTTR ..... 38
3.1.5 Disponibilidade ..... 39
3.1.6 Indisponibilidade ..... 41
3.1.7 Tempo Up ..... 42
3.1.8 Tempo Down ..... 43
3.1.9 #9s - Alta disponibilidade ..... 43
3.2 Integração com FlexLoadGenerator ..... 44
3.2.1 Descrição do FlexLoadGenerator ..... 47
3.2.2 Análise dos Resultados ..... 48
3.2.3 Processo de injeção de falhas ..... 48
**4 Estudo de Caso** ..... 51
4.1 Ambiente de Teste – Eucalyptus ..... 51
4.2 Resultados ..... 52
**5 Considerações Finais** ..... 54
**REFERÊNCIAS** ..... 55

# LISTA DE FIGURAS

**Figura 1:** Exemplo de como funciona a IaaS ..... 17
**Figura 2:** Exemplo do ambiente Eucalyptus ..... 18
**Figura 3:** Protótipo do Injetor ..... 32
**Figura 4**: Protótipo do Monitor ..... 33
**Figura 5:** O Injetor e seus componentes ..... 35
**Figura 6:** Modo monitoramento do Injetor ..... 37
**Figura 7:** Função que calcula a Disponibilidade ..... 40
**Figura 8:** Verificando se a falha foi injetada ..... 40
**Figura 9:** Injetando o Reparo ..... 41
**Figura 10:** Função que calcula a Indisponibilidade ..... 42
**Figura 11:** Função que calcula o Tempo Up ..... 42
**Figura 12:** Função que calcula o Tempo Down ..... 43
**Figura 13:** Função que calcula o #9s – Alta Disponibilidade ..... 44
**Figura 14:** Hierarquia de composição do Injetor ..... 45
**Figura 15:** Interface do Injetor ..... 46
**Figura 16:** Função Running ..... 49
**Figura 17:** Função Timer Inject Failure ..... 49
**Figura 18:** Função Failed ..... 50
**Figura 19:** Função Timer Repair ..... 50
**Figura 20:** Estrutura da rede de teste ..... 52
**Figura 21:** Resultado obtidos no ambiente de teste ..... 53

# 1 Introdução

Em virtude dos avanços tecnológicos empresas de tecnologia tiveram que aumentar de forma exponencial suas estruturas para suprir as necessidades imposta pela demanda de seus clientes, com isto também aumento a possibilidade de ocorrer falhas que podem provocar a morte de pessoas, prejuízos de grande escala ou até mesmo danos irreparáveis ao seu cliente. Por esses fatores impactantes surge a necessidade de desenvolver um Injetor de falhas que ofereça aos usuários fácil usabilidade com funcionalidades diversificadas, confiáveis, de baixo custo e de alto desempenho, que possa simular como possíveis falhas nesses ambientes para evitar que o problema só seja resolvido quando de fato uma falha real acontecer. Existem muitas pesquisas na área de injeção de falhas, contudo percebeu-se que todas as ferramentas estudadas possuem um objetivo fixo, ou seja, funcionam apenas no ambiente de teste pesquisado; e não oferecem suporte ou documentação para serem adaptadas em outros ambientes. Com base nisto, surge o Injetor, que tem como foco principal simular falhas e reparos para entender como o sistema pode se comportar em determinado cenário, oferecendo resultados complexos de se obter, mas fáceis de entender de forma prática e rápida em pesquisas de injeção de falhas e avaliação de desempenho.

## 1.1 Objetivos

O presente trabalho tem por objetivo implementar uma ferramenta geradora e analisadora de eventos sintéticos, com capacidade de realizar o cálculo da Disponibilidade, Indisponibilidade, Quantidade de falhas, tempo que o sistema passa UP e Down, o tempo total de execução da injeção de falhas, e por fim, o #9s que serve para analisar sistemas com alta disponibilidade. Tudo isso ocorrerá em tempo real de execução, otimizando destarte o tempo dos pesquisadores e diminuindo o retrabalho de desenvolvedores de software. Utilizando essa ferramenta, o usuário não precisará esperar até que o processo de geração de eventos acabe para obter seus resultados; o usuário poderá acompanhar todos os resultados, citados acima, de forma automática pela interface do injetor em tempo de execução. Dessa maneira, será possível identificar possíveis problemas e propor melhorias aos sistemas analisados. Por fim, ele também vai extinguir a necessidade de sempre refazer os cálculos de acordo com andamento do processo de injeção de falhas. O estudo de caso será executado em cima de uma Infraestrutura como serviço - IaaS. Todos os cálculos serão realizados em VMs instanciadas através ferramenta para teste de injeção de falhas e reparos, neste trabalho a ferramenta utilizada é denominada de Eucalyptus que será explicada de forma detalhada mais a frente.

## 1.2 Trabalhos Relacionados

Neste tópico, mencionarei alguns trabalhos que foram desenvolvidos com base em geração de eventos e geração de falhas em ambientes e estruturas variadas, realizados em contextos também variados.

O primeiro trabalho a ser explanado foi realizado por SOARES GALINDO (2010), nele foi apresentada uma nova metodologia para o desenvolvimento de um gerador de traces sintéticos voltado para o planejamento de capacidade em um ambiente de servidores virtuais. O mais interesse no trabalho foram as técnicas de carga criadas para simularem um ambiente real. Em decorrência disso, podemos saber como um determinado sistema pode se comportar em momentos de extremo estresse. Além disso, o autor desenvolveu um gerador genérico de variáveis aleatórias *(Random Variates)*. Durante o desenvolvimento do Injetor foram implementadas várias metodologias para a geração de *Random Variates*, incluindo algumas das principais distribuições de probabilidade contínuas e discretas. Para a visualização e validação dos valores gerados, foi criada uma ferramenta com uma interface gráfica bastante útil, que além de ser essencial para a geração dos valores no gerador de carga, poderia ser utilizada para diversas outras finalidades e aplicações.

Outro trabalho bastante importante foi realizado por Souza (2013), que desenvolveu uma plataforma chamada *FlexLoadGenerator*, que pode servir de auxílio no processo de desenvolvimento de ferramentas geradoras de eventos sintéticos. O *framework* sugerido oferece ao desenvolvedor um conjunto de métodos encarregado de realizar a comunicação entre o gerador de evento e o sistema alvo, criação e gerenciamento de eventos, onde o programador necessita apenas especificar em qual evento deve ser criado e escolher como esse deve se comportar. A flexibilidade provida pelo *framework* é uma de suas principais vantagens, no qual o desenvolvedor opta por um dos meios disponibilizados para criação e gerenciamento de eventos. Caso não seja interessante utilizar a estrutura recomendada, pode-se fazer uso da biblioteca de geração de números aleatórios disponibilizada pelo *FlexLoadGenerator* de forma independente.

Na sequência temos o trabalho de Silva (2016), que foi desenvolvido um *framework* chamado de *GeoClouds Modes.* Essa ferramenta avalia o desempenho,

confiabilidade e a disponibilidade dos sistemas de computação em nuvem, implantando *data centers* distribuídos geograficamente, levando em consideração a ocorrência de desastres. O quadro permite a avaliação do impacto da ocorrência de desastres, do tempo de transmissão da VM e de diferentes cargas na performabilidade do sistema. A abordagem proposta permite também avaliar a capacidade de sobrevivência de sistemas em nuvem, levando em consideração a distância entre os data centers, os requisitos de RPO e RTO.

Outro caso importante foi o trabalho de Wanner (2003), que implementou uma ferramenta específica para injetar falhas em sistemas de segurança em rede a fim de testá-los quanto a sua eficiência e o seu correto funcionamento. O injetor de falhas em sistemas de segurança em rede foi projetado com o objetivo de possuir uma granularidade de ataque pequena injetando ataques no nível de rede, similar aos injetores de pacotes, podendo assim testar problemas em protocolos e sistemas utilizados na rede. Tendo recursos e funcionalidades similares aos programas de varredura de vulnerabilidades através da utilização de plug-ins, permite que diversos tipos de ataque mais sofisticados possam ser realizados.

Seguindo temos o Vacaro (2006), desenvolveu uma ferramenta batizada de FIRMI, ela é um injetor de falhas para a avaliação de aplicações distribuídas baseadas em RMI, possuindo como uma de suas principais características a integração em diferentes ambientes de teste e desenvolvimento de aplicações, agregando as vantagens da técnica de injeção de falhas voltada a sistemas distribuídos para estes ambientes. A descrição de cenários de falhas baseada em linguagens de script e a integração de FIRMI em JUnit e ANT, permite que os engenheiros de teste usem a ferramenta de modo complementar nas fases de teste de aplicações; permite também aumentar a cobertura dos testes empregados na validação de sistemas. Os experimentos conduzidos pelo autor, ilustram a capacidade da ferramenta em injetar falhas em pontos específicos da execução de uma aplicação, permitindo, também, a definição de cenários de falhas mais genéricos que não dependem das características de uma aplicação em particular.

Outra ferramenta é a FIAT *(Fault Injection based Automated Testing environment)* [CIP95, ROS98]. Foi uma ferramenta desenvolvida para injetar falhas, ela utiliza inserção de código para corromper bytes em imagens de memória de programas. O código extra pode ser inserido no nível da camada de aplicação ou de

sistema operacional, falhas são disparadas quando determinadas posições de memória são atingidas na execução.

O FIAT procura avaliar a eficiência de detecção de erro e mecanismos de recuperação de aplicações distribuídas, seu método permite injetar falhas em código e dados da aplicação do usuário, bem como em mensagens e timers.

A próxima ferramenta chama-se Ftape *(Fault tolerance and performance evaluator)* [HTI97], esse Framework injeta falhas através da adição de drivers ao sistema operacional sobre o qual o sistema em teste está rodando. O Framework coloca alguns drivers especialmente desenvolvidos no sistema operacional, esses drivers irão injetar falhas mudando os valores de registradores da CPU, posições de memória e valores lidos do subsistema de disco.

Seguindo em frente tempos a ferramenta Doctor *(Integrated Softaware fault Injection environment)* este Framework combina diversos métodos para injetar falhas, ele utiliza *time-outs*, *traps* e alteração de código para fazer injeção de falhas. Para simular falhas de memória, o Injetor usa *time-out,* ele programa um timer, e quando este expira dispara uma interrupção, previamente programa no vetor de tratadores de interrupção do sistema. Portanto, o framework é capaz de mudar o valor da posição de memória. *Traps* são usados para simular falhas de processador numa base não permanente. Como exemplo, seria uma simples alteração da instrução a ser executada.

Por fim, não menos importante, o framework ProFi *(Processor Fault Injection)* [LoE93] utiliza o mecanismo de traço disponível na família de processadores Motorola para injetar falhas, utilizando traço para executar uma instrução do sistema de cada vez, podendo então injetar as falhas. Quando a ferramenta atinge uma instrução, os registradores e os flags do Processador, o resultado de alguma computação ou contador de programa. Ela também pode modificar a instrução que está sendo executada, assim o Injetor pode simular falhas de processador, as falhas a serem injetadas pode ser especificada utilizando uma linguagem de script especialmente projetada chamada ProFIL.

O ProFIL é capaz de injetar permanentemente praticamente todo tipo de falhas relacionadas com o processador, ela pode mudar o valor dos registradores dos flags, dos resultados da computação, do contador de programa e até mudar a própria instrução a ser executada.

## 1.3 Estrutura do Trabalho

Neste capítulo, serão apresentados a estrutura do trabalho e os conceitos essenciais para seu entendimento.

Na seção 2, são apresentados os Fundamentos para melhor compreensão deste trabalho. Na seção 3, veremos o Protótipo e processo de desenvolvimento do Injetor de Falhas e Monitoramento de Defeitos. Mais adiante na seção 4, será mostrado o Estudo de Caso da Ferramenta. E por fim, na seção 5, apresentaremos as Considerações Finais sobre o presente trabalho.

# 2 Fundamentos

Este capítulo apresenta os principais conceitos para o melhor entendimento deste trabalho. Primeiramente, veremos as vantagens e riscos da computação em nuvem, depois vamos entender como funciona a ferramenta utilizada no ambiente de teste, e por fim, vamos explanar os trabalhos mais conhecidos da área de injeção de falhas e avaliação de defeito, conhecendo também as ferramentas desenvolvidas e como elas funcionam em seus respectivos ambientes de teste.

## 2.1 Computação em Nuvem

De acordo com Velt et al. (2010), a computação em nuvem tem despertado o interesse de pequenas e grandes empresas, que passaram a investir em pesquisas sobre o assunto na tentativa de conquistar espaço e tornar-se referência; a título de exemplo, podemos citar In Loco Media, Ifood, Google e Amazon.

Dentre as tendências de TI que ganharam espaço no ambiente corporativo ao longo dos últimos anos, poucas se tornaram tão conhecidas e adotadas na prática quanto o *cloud computing*, isto é, a infraestrutura de computação em nuvem. A execução de serviços com aplicações que utilizam essa operacionalidade possibilita mais mobilidade para empresas, e aumenta a capacidade de resposta à demandas do mercado, otimizando uma série de rotinas. Com esse ganho de performance, as organizações tornam-se mais ágeis no processo de desenvolvimento de novas aplicações, na viabilização de recursos computacionais para novos projetos e no ganho de elasticidade para suportar o aumento de demanda.

Como os testes do injetor serão realizados em ambientes físicos e virtuais com VMs simuladas em nuvens é interessante entender como funciona a computação e nuvens, e saber quais são as vantagens e riscos em utiliza-las.

### 2.1.1 Vantagens

Atualmente, a maioria das empresas de software já oferece seus produtos (sistemas) on-line, sem que o cliente precise instalar nada ou comprar servidores.

Em geral, esse modelo de negócio está se tornando popular com as empresas de Software as a Service (SaaS), que oferecem serviços em nuvens, sem a necessidade de instalação de um software na sua máquina. Abaixo veremos uma pequena lista das vantagens da implantação do sistema de Computação em Nuvem.

### 2.1.2 Maior controle dos Gastos com Software

O uso de sistemas de gestão integrada e plataformas de SaaS (Software as a Service, ou Software como Serviço), que podem ser integrados de modo eficiente em infraestruturas de computação em nuvem, podem ocasionar uma redução drástica de gastos com licenciamento de software em médio e longo prazos. Sem precisar pagar por licenças e recursos que não estão em uso, a empresa elimina parte do seu TCO (*Total Cost of Ownership*, ou Custo Total de Propriedade), podendo assim direcionar parte do seu orçamento para investimentos em novas tecnologias.

### 2.1.3 Maior mobilidade

Assim como a nuvem pode tornar as rotinas internas de uma empresa mais flexíveis, serviços como os de *cloud storage* (armazenamento em nuvem), podem ampliar o nível de mobilidade operacional de uma empresa significativamente.

Isso acontece devido aos sistemas que utilizam infraestrutura de computação em nuvem possibilitarem acesso, modificações e administração de informações diversas por meio de telefones, tablets e/ou interfaces web.

Dessa forma, profissionais podem executar atividades de suporte e processos de tomada de decisão em qualquer local, uma vez que informações corporativas ficam disponíveis por meio de dispositivos com acesso à internet.

### 2.1.4 Ganho de escalabilidade

A infraestrutura de computação em nuvem possui como uma de suas principais características a alta escalabilidade. A quantidade de recursos contratada pode ser alterada conforme for necessário pelo gestor de TI que oferece o serviço na nuvem, dando mais capacidade para que os empreendimentos se beneficiem do

sistema e estejam prontos para as mudanças do mercado, evitando reduções no seu nível de produtividade. Como consequência, a empresa, ao utilizar sistemas com infraestrutura de computação em nuvem, terá sempre a quantidade de recursos necessária para a execução dos seus serviços.

### 2.1.5 Riscos

Um dos maiores problemas enfrentados pelas empresas que possuem serviços baseados em computação em nuvem, é o uso incorreto de ferramentas que, como consequência, pode resultar em perda de dados.

Muitos gestores temem que, com a adoção de soluções que usam a nuvem, os dados corporativos fiquem expostos a ataques externos, tentativas de roubo e captura de informações. No entanto, com um conjunto de boas práticas e medidas preventivas, é possível reforçar a segurança do usuário e impedir que arquivos fiquem disponíveis para terceiros.

A maioria das falhas de segurança ocorre por engenharia social. Isto é, os dados são roubados sem que hackers consigam invadir computadores ou programar arquivos maliciosos. E-mails falsos e sites *phishing*, por exemplo, são muito utilizados para que os usuários coloquem suas senhas e dados corporativos deliberadamente. Uma dica para minimizar esse tipo de risco é, sempre que possível, utilizar autenticação de dois passos.

### 2.1.6 Infraestrutura como serviço (IaaS)

A IaaS é o modelo no qual a infraestrutura de servidores é contratada como serviço para fazer o *hosting* de uma ou mais aplicações. Em outras palavras, é contratado armazenamento, memória e processamento como serviço. Assim, é possível controlar e gerenciar máquinas virtuais, aplicações, arquivos armazenados etc.

Caso a companhia mantenha os servidores *in-house*, ou quiser reduzir os custos de manutenção, se tornará mais satisfatório optar pela terceirização, em prol, também, de mais agilidade o que implica mais competitividade. De certa maneira,

optar por serviços de infraestrutura (IaaS) pode ser comparado entre adotar a rede elétrica ou não possuir um gerador dentro de casa que forneça energia elétrica. No lugar de fios de alta tensão, o serviço chegará até o destino via internet; o IaaS, não l lida com a infraestrutura: há uma terceirização desse serviço para o vendedor.

Destarte, os gastos e a demora de correções feitas nos próprios servidores são reduzidos ou praticamente eliminados. Obviamente, não é possível terceirizar tudo. É neste momento em que o negócio da empresa é analisado. É preciso distinguir quais são os pontos chave da T.I. que devem ser mantidos *in-house*, seja por questões de segurança ou pelo fato de oferecer mais agilidade para a empresa em determinadas situações. Fornecedores de IaaS são, caracteristicamente, responsáveis pela replicação, pelo backup e pelo arquivamento de informações do cliente. Dessa maneira, praticamente não haverá complicações em lidar com a infraestrutura que não pertence à companhia, possibilitando controlar o software utilizado.

**Figura 1:** Exemplo de como funciona a IaaS

De acordo com a figura acima, a IaaS funciona em conjunto com mais duas estruturas: SaaS e PaaS. No presente trabalho, optamos por utilizar a IaaS em razão dos fatores de redução de custos, desperdícios, ineficiência, aumento da produtividade e segurança. Em consequência dos benefícios que a IaaS nos oferece, utilizamos uma infraestrutura de computadores com o sistema operacional Centos 6.5 instalado, configurados em uma rede com duas máquinas simulando VMs instanciadas. No tópico seguinte entenderemos o que é, como funciona e qual a utilidade do Eucalyptus.

### 2.1.7 Eucalyptus

Eucalyptus é um software de computador pago, porém, de código aberto para a criação de ambientes de computação em nuvem privados e híbridos, compatíveis com *Amazon Web Services (AWS)*, desenvolvido pela empresa Eucalyptus Systems. Eucalyptus é uma abreviação de *Elastic Utility Computing Architecture* para vincular seus programas a sistemas úteis. O software permite a junção de recursos computacionais, armazenamento e rede que podem ser ampliados ou reduzidos dinamicamente conforme as cargas de trabalho de aplicativos são alteradas.

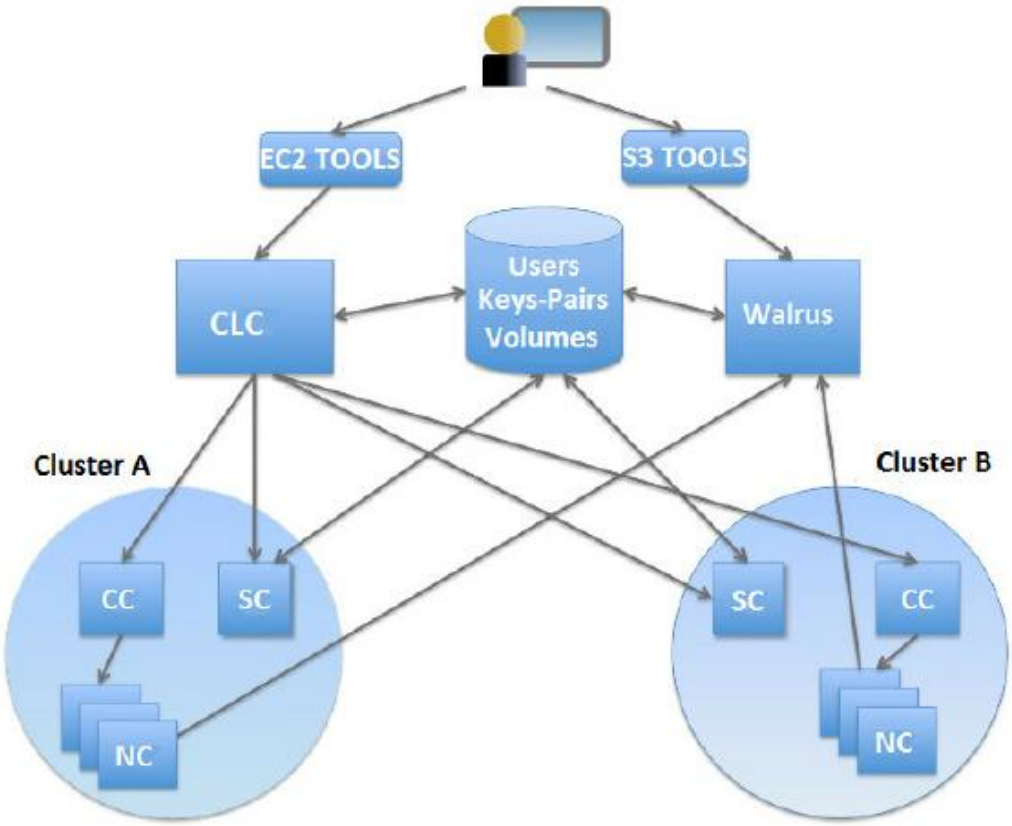


**Figura 2:** Exemplo do ambiente Eucalyptus

De modo geral, a plataforma de computação em nuvem da Eucalyptus usa os recursos de virtualização (*hypervisor*) do sistema de computador subjacente para permitir a alocação flexível de recursos desacoplados do hardware específico. Há cinco componentes de alto nível na arquitetura do Eucalyptus, cada um com sua própria interface de serviço da Web: Os componentes são o *Cloud Controller, Node*

*Controller, Cluster Controller, Storage Controller e Walrus*. No próximo paragrafo vamos entender como cada um desses componentes funcionam.

O primeiro se chama *Cloud Controller* (CLC) é o *Front-end* para toda a infraestrutura de nuvem. O *CLC* é responsável por expor e gerenciar os recursos virtualizados subjacentes (servidores, rede e armazenamento) por meio da *API* do *Amazon EC2*. *Node Controller* (NC) é executado em cada nó de cálculo e controla o ciclo de vida das instâncias em execução no nó. NC faz consultas para descobrir os recursos físicos do nó, a título de exemplo: número de núcleos de CPU, tamanho da memória principal, espaço em disco disponível etc, além de aprender sobre o estado das instâncias de VMs desse nó. O *Cluster Controller* (CC) reúne informações sobre um conjunto de VMs e agenda a execução de VMs em NCs específicos. *O Cloud Controller* possui três funções principais: agendar, para nós específicos, as solicitações de entrada para execução de instâncias de VMs, controlar a sobreposição de rede virtual composta por um conjunto de VMs e reunir informações sobre um conjunto de controladores de nó, para relatar seu status ao Cloud Controller.

O controlador de armazenamento (SC) fornece armazenamento persistente para ser utilizado por instâncias de máquina virtual. Ele implementa o armazenamento em rede acessado por block, semelhante ao fornecido pelo *Amazon Elastic Block Store EBS*, e suporta vários protocolos do sistema de armazenamento, por exemplo: *NFS, iSCSI etc*. Além do que foi descrito acima temos o *Walrus* que é um serviço de armazenamento de dados baseado em arquivos compatível com o *Simple Storage Service* (S3) da *Amazon*. Porém como ele não será utilizado na estrutura de testes deste projeto no momento isso é o necessário que precisamos saber sobre ele. O Eucalyptus traz inúmeras funcionalidades para que os pesquisadores possam retirar o máximo de suas pesquisas. Em consequência disso, esta será a ferramenta utilizada neste trabalho para a realização dos testes de injeção de falhas.

## 2.2 Injeção de Falhas e Monitoramento de Defeitos

Com o passar dos anos houve um avanço continuo nos meios tecnológicos, hoje temos redes de computadores de alta velocidade que permitem que centenas

de dispositivos espalhados pelo mundo possam trocar informações quase na velocidade da luz. Tal comportamento infere na possibilidade de cooperação entre os dispositivos, formando um sistema distribuído.

De acordo com Birman (1997), um sistema distribuído consiste em um grupo de aplicações, executando em um ou mais nodos, que coordenam atividades através da troca de mensagens. Tanenbaum e Steen (2006), por sua vez, definem um sistema distribuído como um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente.

Apesar de agregar uma série de problemas inexistentes em sistemas autocontidos, um sistema distribuído tem como objetivo construir um sistema que possa recuperar-se de falhas parciais automaticamente, sem efeitos colaterais aos demais nodos (TANENBAUM; STEEN, 2006). Ou seja, um sistema distribuído deve permanecer em funcionamento de maneira aceitável mesmo na presença de falhas.

Em virtude da possibilidade ocorrência de falhas em sistemas distribuídos, aplicações que operam nestes ambientes devem incorporar técnicas de tolerância a falhas que visam, primeiramente, cobrir quaisquer ocorrências fora do padrão, de modo que a aplicação mantenha um serviço íntegro e confiável. Entretanto, tais mecanismos de tolerância a falhas devem ser testados e avaliados, pois, caso seu papel não seja cumprido, há margem para a propagação de falhas que, por sua vez, podem levar a aplicação-alvo a apresentar defeito. Além disso, a avaliação de uma aplicação-alvo não é uma tarefa trivial, visto que são formadas por protocolos de comunicação e sistemas distribuídos que estão cada vez mais difíceis de testar (DAWSON; JAHANIAN; MITTON, 1997).

### 2.2.1 Injeção de Falhas

Há mais de 40 anos, a utilização de injeção de falhas como um método experimental da verificação de dependabilidade em diversos sistemas vem resultando em diferentes técnicas para a reprodução de comportamentos anômalos seja por simulação, por hardware, ou ainda por software. A característica de injetores de falhas está relacionada com diversos fatores, tais como: controlabilidade, indicador de tempo (quando) e espaço (onde) para a injeção de falhas; repetibilidade, que diz respeito à precisão na habilidade de repetir um

experimento inúmeras vezes sem a necessidade de reconfiguração; observabilidade, atributo indicador da capacidade de prover mecanismos para coleta de métricas e análise dos efeitos causados por falhas no sistema.

Outro aspecto importante no projeto do injetor de falhas é a diminuição do impacto da intrusividade na aplicação-alvo, uma vez que não pode ser evitada. Desta forma, cabe aos desenvolvedores destas ferramentas buscarem a minimização deste impacto.

De acordo com sua natureza, a intrusividade pode ser de duas formas: temporal e espacial. A intrusividade temporal, também chamada de perturbação, é identificada quando a ação de um injetor de falhas implica no aumento do tempo de processamento da aplicação-alvo. Isto, pois o *Wifi Access Control Solution* (SWIFI) demanda tempo para execução de suas rotinas de interceptação do fluxo da aplicação-alvo, bem como de espaço adicional em memória, sendo impossível anular este tipo de intrusividade.

A intrusividade espacial, geralmente, está relacionada à instrumentação direta do código fonte da aplicação-alvo. Ou seja, a aplicação já vem compilada com rotinas necessárias para injeção de falhas. Entretanto, seu uso pode levar a inserção de bugs não intencionais e alterar o ciclo de execução da aplicação, caracterizando-se também um cenário com intrusividade temporal.

Independentemente de sua abordagem para injeção de falhas, a maior parte das ferramentas de injeção de falhas não estão acessíveis à comunidade ou, se disponibilizadas, há escassez de documentação e, muitas vezes, são pouco funcionais, seja por sua obsolescência, seja sua arquitetura voltada para ambientes muito específicos. Por essa razão, injetores de falhas tornaram-se ferramentas muito valiosas no âmbito de avaliação e análise de dependabilidade de sistemas.

Entretanto, conforme citado anteriormente, este trabalho é voltado para área de injeção de falhas e monitoramento de sistemas. Logo, o foco desta pesquisa concentra-se em injetores de falhas de falha e reparo.

É com base nisto que veremos diferentes tipos de injetores de falhas e os ambientes utilizados para realização de seus testes.

### 2.2.2 Injeção de Falhas de Comunicação

As estratégias de tolerância a falhas do protocolo de comunicação devem ser ativadas para avaliar sua correção. Como a detecção de falhas só entra em operação na ocorrência de falhas, tais falhas devem estar presentes durante a avaliação do protocolo.

Para não depender da ocorrência natural de falhas no ambiente real de operação, uma técnica eficaz (Hsueh, Tsai, and Iyer 1997) é emular falhas por software e injetá-las no componente alvo do teste. Através da introdução controlada de falhas, é possível determinar se o protocolo tolera ou não as falhas injetadas.

Um experimento de injeção de falhas é um teste executado em condições controladas na qual são introduzidas falhas. Seguindo uma carga de falhas pré-definida, falhas são introduzidas no sistema durante a execução de uma carga de trabalho. Ambas a carga de trabalho e a carga de falhas devem ser representativas do uso do sistema em condições reais de operação. Para avaliação de protocolos de comunicação, a carga de trabalho são as mensagens trocadas entre os componentes da rede e a carga de falhas inclui as falhas que afetam a troca de mensagens e estão previstas na especificação do protocolo, como por exemplo: corrupção, repetição, perda, inserção, troca de ordem, atraso e mascaramento de mensagens.

Em um experimento, parâmetros que podem ser variados para a carga de trabalho incluem número, tamanho dos pacotes e frequência de transmissão. Parâmetros de carga de falhas incluem: tipos de falhas, instantes de ativação, frequência de ocorrência de falhas e distribuição de probabilidade. Um ambiente experimental de injeção de falhas injeta falhas na troca de mensagens e monitora o comportamento do protocolo enquanto executa uma carga de trabalho.

Um problema neste tipo de teste é a falta de injetores adequados. Apesar dos vários injetores reportados (Natella, Cotroneo, and Madeira 2016), ferramentas funcionais e com usabilidade adequada são difíceis de serem encontradas. A maioria das ferramentas apresenta problemas de portabilidade (Schirmeier et al. 2012) que vão desde terem sido desenvolvidas para ambientes muito específicos, a exigirem sistema operacional ou bibliotecas específicas. Grande parte dos injetores não tem código disponível para uso ou não é mais mantido por seus desenvolvedores.

Em virtude deste ponto de vista é explícito a falta de ferramentas que funcionem em mais de um ambiente específico e que principalmente tenha o código aberto ao público, reaproveitamento de código é essencial em alguns casos, se existe algo que já faz o que você precisa ou até mesmo é parecido. Não seria inteligente desenvolver tudo do zero, pois, isso seria mais custoso e complexo. Portanto, podemos concluir que esta área precisa de mais estudos em correlação disto, surge a necessidade de criar novos métodos, novos ambientes de teste e novas ferramentas mais flexíveis.

### 2.2.3 Ferramentas de teste para Injeção de Falhas

Dentre várias ferramentas existentes para testes de injeção de falhas, veremos a seguir as mais populares neste âmbito:

1. DBS (Distributed Benchmark System) (MURAYAMA; YAMAGUCHI, 1997), uma ferramenta de avaliação de desempenho de redes TCP/IP, permite múltiplos envios de dados de forma coordenada para posterior avaliação do comportamento do protocolo TCP;
2. Dummynet (RIZZO, 1997), uma ferramenta com alto grau de flexibilidade para gerenciamento de banda a partir da simulação de condições desejáveis no tráfego de rede, como atrasos, filas, descartes de pacotes, entre outros; por fim, uma ferramenta de propósito geral;
3. NIST Net (CARSON; SANTAY, 2003), que permite a emulação de características críticas presentes em redes de larga escala, como congestionamentos e descarte de pacotes, para fins de avaliação do desempenho dinâmico em redes IP (Internet Protocol).
4. HexInject (ACRI, 2010), é um injetor de pacotes e um sniffer de rede baseado no framework Netfilter, que provê mecanismos para leitura, interceptação e alteração do tráfego de rede de maneira transparente à aplicação-alvo e permite a construção de poderosos scripts em conjunto com outras ferramentas de linha de comando.
5. NetEm (HEMMINGER, 2005), ou Network Emulation é outra ferramenta de teste de protocolos de comunicação que emula propriedades de redes de larga escala para avaliação de propriedades de qualidade de serviço. Tais

propriedades consistem no atraso, descarte, duplicação e reordenamento de mensagens através do framework Traffic Control, disponível no Kernel do Linux 2.6. Outra ferramenta, também baseada no framework Traffic Control, é o WANEm (Wide Area Network Emulator). Além das propriedades emuladas pela ferramenta NetEm, este permite a emulação de corrompimento, colapsos de link e Jitter para possibilitar a reprodução de comportamentos peculiares ao tráfego de voz.

Entretanto, apesar da alta disseminação destas ferramentas de teste, as necessidades de avaliação dos requisitos mínimos para alcançar dependabilidade não são satisfeitos. Mesmo com o suporte à reprodução de comportamentos característicos de ambientes de comunicação, como o descarte e atraso de mensagens, ferramentas de teste não suportam a construção de um cenário de falhas elaborado para a realização de testes bem definidos de acordo com o engenheiro de testes, de maneira que funções específicas do alvo sob teste possam ser executadas corretamente.

## 2.3 Ferramentas de Injeção de Falhas

Em correlação às ferramentas de teste de injeção de falhas, e para proporcionar melhores condições de descrição de cenários, diversos injetores de falhas de comunicação vêm sendo propostos pela comunidade cientifica. Estes, por sua vez, visam garantir a realização de testes, seguindo especificações definidas pelo engenheiro de testes, seja para avaliação de desempenho e monitoramento, seja para avaliação de aplicações distribuídas. De acordo com sua natureza, injetores de falhas de comunicação podem ser locais ou distribuídos.

Apesar de tratarem de falhas de comunicação, alguns injetores de falhas podem ser classificados como locais por não possuírem suporte à coordenação de atividades. Desta forma, para elaborar um ambiente onde há interação entre diversas estações com um determinado comportamento sobre o canal de comunicação, é necessário percorrer os nodos da rede um a um para efetuar o preparo da ferramenta, bem como o disparo da execução dos testes e os seus reparos caso seja a intenção do teste.

Abaixo serão listados alguns tipos de injetores de falhas que foram encontrados no decorrer desta pesquisa com os intuitos que foram mencionados.

1. ORCHESTRA (DAWSON; JAHANIAN; MITTON, 1996) é uma ferramenta de injeção de falhas para avaliação e validação de características temporais e de dependabilidade de protocolos distribuídos. Baseada em um simples e poderoso framework que introduz um conceito de sondagem e injeção de falhas orientada scripts (DAWSON; JAHANIAN, 1995), ORCHESTRA foca-se no desenvolvimento de técnicas de injeção de falhas que podem ser empregadas para (1) detecção de erros de modelo ou implementação de protocolos, (2) identificação de violação de sua especificação, e (3) obtenção de conhecimento sobre decisões e definições de modelos por parte dos desenvolvedores.

Criada com o objetivo de operar em diferentes plataformas, a portabilidade e capacidade de injetar falhas diretamente na pilha de protocolos são objetivos a serem alcançados com um mínimo de intrusividade no protocolo alvo. Para tanto, ORCHESTRA define uma arquitetura baseada em camadas onde nenhuma distinção é feita entre protocolos de nível de aplicação, comunicação ou físico. Esta abordagem consiste na inserção de uma camada PFI (*Protocol Fault Injection*) à pilha de protocolos, abaixo da camada a ser testada. Ao inserir esta camada entre duas camadas da pilha de protocolos, torna-se possível o filtro e manipulação de mensagens trocadas entre participantes da comunicação sem a necessidade de alteração do código da camada alvo.

2. NFTAPE (STOTT et al., 2000) é um *framework* voltado para avaliação de dependabilidade de sistemas distribuídos. Desenvolvido a partir de dificuldades em encontrar ferramentas automatizadas para injeção de falhas que satisfaçam uma série de requisitos, NFTAPE emprega uma arquitetura modular com suporte a múltiplos modelos de falhas, múltiplos mecanismos de disparo de falhas, múltiplos alvos e métodos versáteis de coleta e reporte de erros.

Ao inverso das demais ferramentas, normalmente, a função dos componentes de um injetor de falhas é assumida pelos módulos *(*LWFI) (*LightWeight Fault Injector*,

injetor de falhas leve). Estes módulos correspondem a pequenos programas que se responsabilizam única e exclusivamente pela injeção de falhas, despreocupando-se com demais funções desempenhadas pelo próprio NFTAPE, como gatilhos, *logging*, comunicação e outros. Além disso, há uma independência entre tais componentes, o que permite a execução de diferentes combinações de módulos *LWFI*, gatilhos e cargas de trabalho para a construção do ambiente de testes.

3. FIRMAMENT (DREBES, 2005) (acrônimo para *Fault Injection Relocatable Module for Advanced Manipulation and Evaluation of Network Transports*), também desenvolvido pelo grupo de tolerância a falhas da UFRGS, destina-se à validação experimental de técnicas de tolerância a falhas de protocolos de comunicação e sistemas distribuídos. Sua construção a partir do uso da extensibilidade do núcleo *Linux* através de módulos relocáveis minimizam o efeito de sondagem no núcleo.

A ferramenta permite diversas opções para a injeção de falhas de comunicação e permite a especificação de testes através de *faultlets*, uma linguagem de baixo nível para descrição de cenários de falhas. Diferentemente da abordagem proposta pela ferramenta ORCHESTRA, FIRMAMENT foi desenvolvido para utilizar características específicas do núcleo de código fonte aberto *Linux*, de modo que os *faultlets* são convertidos para um conjunto de instruções (*bytecode*) e interpretados pela ferramenta através de sua máquina virtual, proporcionando maior poder para especificação de cenários. Esta linguagem de *bytecode* ainda permite a inspeção e seleção de mensagens de forma determinística ou estatística e provê ações a serem realizadas que o fazem imitar o comportamento de falhas reais, como descarte e duplicação de mensagens, atraso e modificação de conteúdo.

4. FIRMI (VACARO; WEBER, 2006) (acrônimo para *Fault Injector for RMI*), é voltado para avaliação do comportamento mediante a presença de falhas em aplicações *Java* baseadas no protocolo *RMI* (*Remote Method Invocation*). Para possibilitar a integração com diferentes ambientes de desenvolvimento e teste (*JUnit*, *ANT*, etc.), FIRMI foi projetado para ter uma arquitetura simples e modular, e seu mecanismo de descrição de cenários de falhas é realizada através da criação de classes *Java*.

Apesar de suportar em seu modelo de falhas como colapso de nodo, colapso de *link*, falhas de temporização e falhas bizantinas, FIRMI não apresenta uma arquitetura distribuída. Desta forma, não é possível emular falhas de particionamento de rede, alvo deste trabalho, pois não considera a ação coordenada entre injetores de maneira simultânea em um experimento.

Para injeção de falhas de colapso de nodo e colapso de *link*, o injetor combina a interceptação de requisições *RMI* no nível da *JVM* (*Java Virtual Machine*) e a interação com o sistema operacional através de arquiteturas de *firewall*. Desta forma, a emulação de falhas de *RMI* não se prende unicamente no nível da *JVM*, visto que o protocolo opera sobre o protocolo *TCP* e este, por sua vez, é gerenciado pelo *kernel* do sistema operacional. Entretanto, (VACARO; WEBER, 2006) inferiu que a emulação de falhas apenas no nível de *kernel* do sistema operacional é inviável, pois o fluxo de mensagens *RMI* neste nível de abstração se encontra serializada. Em outras palavras, é necessária a análise do fluxo de dados com o objetivo de extrair apenas as informações úteis ao injetor.

5. FAIL-FCI (HOARAU; TIXEUIL, 2005) é um framework voltado para avaliação de dependabilidade de aplicações distribuídas. Sua proposta consiste em dois componentes principais, (1) FAIL (FAult Injection Language), uma linguagem abstrata de alto nível para fácil construção de cenários de falhas, e (2) FCI (FAIL Cluster Implementation), uma plataforma para injeção de falhas distribuídas onde a linguagem de entrada para a descrição de cenários de falhas é feita em FAIL.

A definição de cenários de falhas descreve máquinas de estado com seu respectivo modelo de ocorrência de falhas, como também a associação entre estas máquinas com um ou mais computadores componentes da rede. A plataforma FCI, por sua vez, é composta por três blocos, (1) um compilador FCI responsável pela pré-compilação dos cenários descritos em FAIL, que produz códigos-fonte na linguagem C++ e arquivos padrão de configuração, (2) uma biblioteca FCI, encarregada do empacotamento e distribuição dos arquivos gerados pelo compilador às máquinas participantes do experimento, e (3) um daemon FCI, executáveis gerados por cada máquina na rede de acordo com os arquivos submetidos pela biblioteca FCI.

6. LOKI (CHANDRA et al., 2004) é um injetor de falhas para aplicações distribuídas. A ocorrência de defeitos em sistemas distribuídos e sua possível dependência de um estado global do mesmo para o disparo de falhas levou à implementação deste injetor. Entretanto, o desenvolvimento de mecanismos sofisticados baseados em um estado global do sistema requer certos cuidados, como a (1) baixa intrusividade, para não inferir na degradação de desempenho no sistema, (2) alta precisão no disparo e análise de falhas, e (3) alta flexibilidade em um âmbito de reprodução e metrificação de testes.

Para o seu desenvolvimento, LOKI utiliza a ideia de visão parcial do estado global do sistema, em conjunto com mecanismos de sincronização *offline*, como solução para obter uma implementação pouco intrusiva. Em outras palavras, é utilizado um algoritmo de sincronização de *clock offline* para tradução dos tempos locais de cada nodo em uma linha de tempo global.

O processo de execução, por sua vez, inclui uma máquina de estados que contém códigos responsáveis pela manutenção da visão parcial do estado global do sistema, bem como injeção falhas ao passo que o mesmo atinge um estado específico, e coleta de informações acerca de troca de estados e comportamentos do sistema frente às falhas injetadas. Desta forma, após a execução do experimento é realizada uma análise com base em dados coletados durante a execução, a fim de verificar se falhas foram injetadas adequadamente e inferiram no comportamento esperado do sistema.

### 2.3.1 Injeção de Falhas para Avaliação de Segurança

A área de segurança pode ser relacionada com a área de tolerância a falhas no sentido que investiga apenas os defeitos causados por falhas maliciosas. Por isso, a injeção de falhas na área de segurança não visa a testar todas as características de dependabilidade de um sistema. As características que se desejam testar são: disponibilidade, confidencialidade e integridade.

A disponibilidade é medida pelo tempo total de funcionamento do sistema, mas no caso de segurança deseja-se saber se o sistema é capaz de resistir a

ataques e manter o atendimento aos serviços prestados mesmo havendo queda de performance.

A confidencialidade se refere à segurança oferecida no acesso a dados, ou seja, os dados somente podem ser acessados pelas pessoas ou programas que tenham permissão. Além disso, deseja-se que todo e qualquer dado seja protegido contra modificações indesejáveis, ou, pelo menos, que qualquer modificação de um dado possa ser detectada.

As redes de computadores atuais são mais difíceis de serem protegidas, pois o número e a variedade de computadores existentes em uma rede tendem a aumentar cada vez mais, o que aumenta em muito a complexidade para se garantir a segurança individual de cada máquina. Assim, a tendência que existe atualmente é a de aumentar a segurança das redes de computadores, controlando os dados que trafegam na rede, o acesso às diversas máquinas e aos tipos de serviços fornecidos por cada computador em uma rede, em vez de se aumentar a segurança de cada computador individualmente.

### 2.3.2 Ferramentas para Injeção de Falhas em Segurança de Redes

Após pesquisas sobre ferramentas deste contexto foram encontradas duas ferramentas disponíveis para injeção de falhas em sistemas de segurança em rede, observou-se a existência de basicamente dois tipos de programas para esse cenário. Existem as ferramentas para geração de pacotes e os programas de varredura de vulnerabilidades.

Os aplicativos para geração de pacotes, também conhecidos como injetores de pacotes, são ferramentas que permitem a criação e envio de pacotes de protocolos da pilha TCP/IP para uma determinada estação. Dessa forma, é possível criar qualquer tipo de pacote, forjando-se comunicações e origens de dados com a finalidade de testar-se sistemas como *firewalls* e sistemas de detecção de intrusão.

A segunda ferramenta encontrada foi os *scanners* de vulnerabilidades. Essa ferramenta foi desenvolvida com o objetivo de automatizar a procura por falhas em serviços e estações existentes em uma determinada rede. Apesar de não serem aplicativos que realizem testes em rede, são bastante usados para testar os

mecanismos de detecção de um IDS por serem utilizados por intrusos que buscam vulnerabilidades em um determinado alvo.

### 2.3.3 Ferramentas para Injeção de Pacotes

As ferramentas para injeção de pacotes permitem que pacotes de protocolos da pilha TCP/IP sejam criados e injetados em uma rede. Os tipos de protocolos suportados e os dados que podem ser configurados variam de aplicativo para aplicativo. De modo geral, os protocolos suportados são o IP, TCP, UDP e ICMP.

Foram encontradas algumas das ferramentas existentes para a geração e envio de pacotes de protocolos da pilha TCP/IP como hping [SAN 2001], nemesis [GRI 2000] e sendip [RIC 2000]. Apesar de existirem inúmeros aplicativos destinados a esse fim, de modo geral todos possuem características semelhantes.

### 2.3.4 Ferramentas para Varredura de Vulnerabilidade

Estas ferramentas foram desenvolvidas para automatizar a busca por falhas em serviços e em estações de trabalho existentes em uma rede de computadores. Podem ser utilizadas como ferramentas administrativas para o gerenciamento de segurança, e também como ferramentas de ataque por intrusos à procura de falha que possam ser exploradas para invadir uma estação. Dentre os aplicativos encontrados para esse fim podemos citar: Nessus [DER 2000], SAINT [WOR 2000] e SARA [TOD 1999].

Esse tipo de aplicativo primeiramente realiza uma varredura na estação alvo a fim de verificar os serviços ativos, as portas abertas e o tipo de sistema operacional existente nela. Após a varredura inicial, e com base nos seus resultados obtidos é feita uma série de testes a procura de falhas conhecidas que possam existir nos serviços ou no sistema operacional. Ao final dos testes, é gerado um relatório das características das estações e dos problemas encontrados.

Os programas para varredura de vulnerabilidades realizam a maioria dos seus testes no nível de aplicação, procurando falhas em serviços, aplicativos, sistemas operacionais e representam um grande risco à segurança de uma rede quando

utilizado por pessoas não autorizadas. Assim, as ferramentas de segurança de rede, devem ser capazes de detectar a ação de tais aplicativos.

Existem várias outras ferramentas que abrangem varreduras e vulnerabilidade por um fator simples, a necessidade de desenvolver sistemas cada vez mais sistemas simples e seguros, porém isso não é uma atividade fácil nem tão pouco barata devido a grade complexibilidade deste nicho. Portanto, com os estudos realizado nesta pesquisa percebemos que existe um grande aglomerado de ferramentas excepcionais publicadas, com objetivos na maioria dos casos específicos, devido a isso sempre que uma nova ferramenta ou arquitetura for criada, novos ambientes de testes e ferramentas de injeção serão criados para colocar o sistema a prova, prevendo possíveis falhas e evitando prejuízos.

# 3 Protótipo do Injetor

Antes de iniciar o processo de desenvolvimento da ferramenta foram criados alguns protótipos para facilitar a produção do Injetor de falhas, o primeiro passo foi realizado em papel com o desenho de um mapa mental, facilitando o entendimento.

Segundo passo descrever as funções que a ferramenta irá oferecer e seus protocolos de comunicação com os sistemas.

A terceira etapa foi decidir qual linguagem de programação o injetor será desenvolvido, a linguagem escolhida foi o JAVA devido a sua gama de recurso e facilidade em desenvolvimento de injetores orientados a objetos com interface gráfica, e também pela sua compatibilidade.

Dentro do pacote *Interface,* no qual se encontra o Injetor, foi criada uma classe chamada *Dependability*, responsável por todos os cálculos que o Injetor nos fornece através do Monitor. A classe é composta por 105 linhas de código e tem a função de ler e calcular as quantidades de paradas do sistema gerados pelo processo de injeção de falhas. De acordo com as informações de tempo e quantidade de falhas realizar os cálculos e mostrar os seus respectivos resultados.

A figura abaixo é uma prévia do injetor sendo representada através de uma modelagem criada com ajuda da ferramenta online *Balsamiq*.

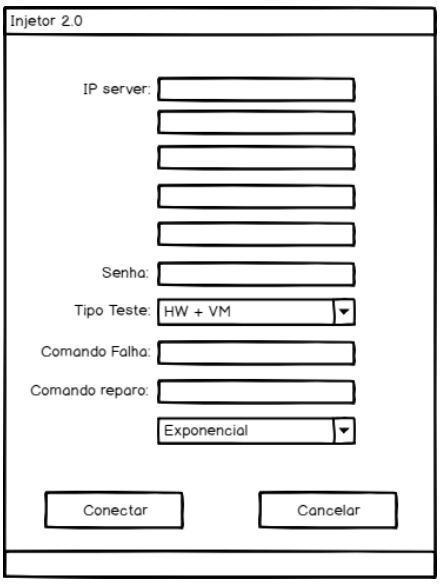


**Figura 3:** Protótipo do Injetor

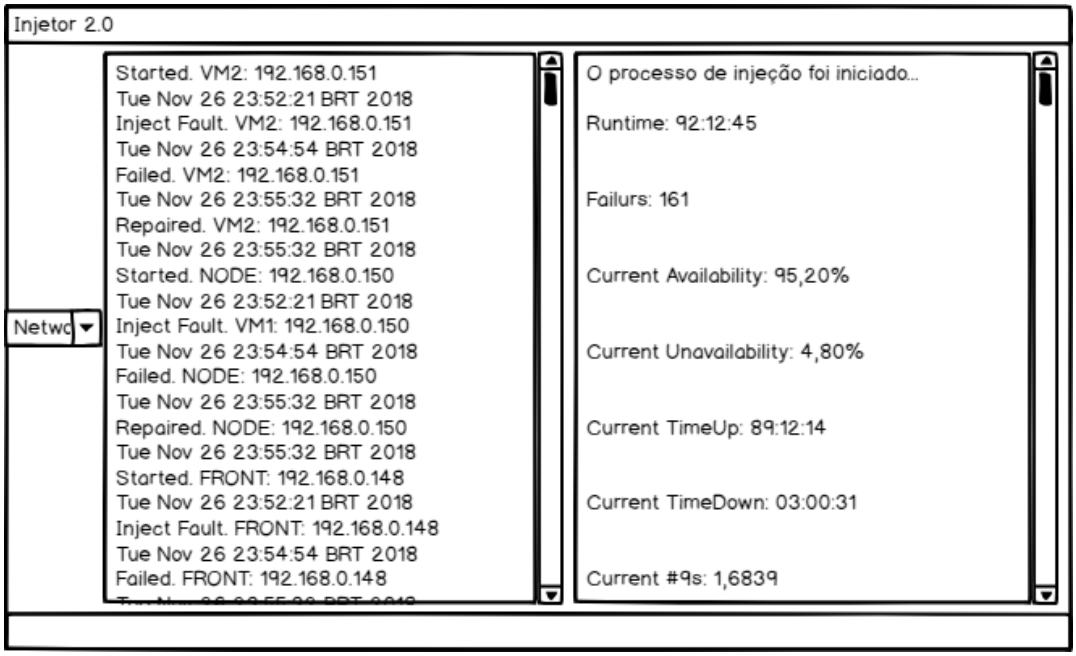


**Figura 4**: Protótipo do Monitor

Como mostra a figura 4, ao lado esquerdo temos as informações do processo de injeção de falhas. Ao lado direito temos as informações de Tempo Total de execução, quantidade de falhas, Disponibilidade, Indisponibilidade, Tempo Up, Tempo Down, e por fim o #9s, sendo calculadas em tempo de execução através do Monitor. Esse protótipo foi um modelo inicial gerado antes mesmo de qualquer validação, no próximo capítulo veremos uma versão funcional do Framework através da primeira versão do Injetor.

## 3.1 Visão Geral do Injetor de Falhas

O Framework de injeção de falha e monitoramento é um gerador de eventos de falhas com opção de reparo, que emula e simula a ausência de operações no sistema de nuvem gerenciado por plataformas computacionais. As falhas causadas pelo Injetor são transientes, ou seja, ele simula um possível estado de interrupção da execução do sistema, de modo que ele possa ser reparado. As perturbações causadas através da inserção de falhas no sistema de nuvem, buscam afetar a execução dos componentes de alto-nível da plataforma de nuvem. A ação de reparo objetiva recuperar o sistema de eventuais falhas causadas pelo injetor.

O reparo é executado após uma falha ser inserida, o que implica que a ferramenta não perpetua o reparo de uma falha que não foi injetada por ela mesma. O disparo de falhas e o reparo destas, processasse quando solicitado, e após um período de tempo determinado de forma aleatória com base no algoritmo de distribuição aleatória que são encontrados no Kernel do *FlexLoadGenerator*.

O Injetor possui as seguintes operações:

- Falhas e reparos de software: Injeta falhas e calcula os resultados em componentes de alto nível de plataformas de nuvens, os comandos são previamente estabelecidos. A ferramenta atua diretamente, suspendendo a execução de um processo selecionado pelo usuário. Vale ressaltar que na mesma máquina pode-se injetar mais de um tipo de falha de software. O reparo é realizado através da restauração dos processos que foram anteriormente finalizados pelo Injetor.
- Monitor: O monitor possui a finalidade de checar o status da rede e informar ao usuário se os serviços estão ocorrendo da maneira devida, ao final exibe um relatório do que ocorreu durante o processo de injeção de falhas. Além disso, o mesmo processa o tempo de atividade e inatividade da rede, esta ocorrência é observada em tempo real.

O Injetor possibilita que o processo de injeção seja realizado de acordo com a necessidade do usuário. Podendo selecionar o ambiente em que deseja executar o

teste e selecionando o tipo de distribuição de números aleatórios, e por fim, salvando todos os dados importantes em arquivos com extensão TXT, para que os resultados dos testes não corram o risco de sumirem por algum motivo inesperado. Atualmente, o framework foi testado com sucesso em plataformas Windows e Linux. Caso o usuário queira utilizar outra plataforma de nuvem, basta realizar algumas pequenas modificações no código do programa, como por exemplo, algum comando novo que seja necessário passar para o sistema alvo. Neste caso, o desenvolvedor pode adicionar outras funcionalidades ao software. A ferramenta engloba núcleos de funções que suportam os meios básicos que são necessários para a construção do injetor. A Figura 7 expressa um usuário manuseando o Injetor. Inicialmente são estabelecidos os dados. Em seguida, ele é transferido para o Framework onde será empregado como mecanismo de injeção de falha.

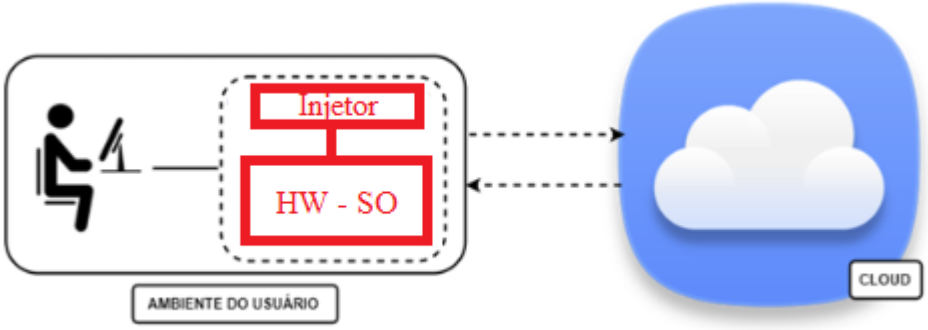


**Figura 5:** O Injetor e seus componentes

### 3.1.1 Estrutura do Injetor

O injetor é uma ferramenta desenvolvida para injetar falhas e reparos e monitorar os resultados em vários ambientes de teste que suportam tal função. Em sua interface temos o atributo IP onde o usuário especifica o IP do componente no qual ele deseja iniciar a comunicação; O *TypeTest* dá ao usuário a opção ao usuário de injetar no componente desejado, as opções disponíveis são: Cloud Controller (CLC), Node Controller (NC) e Cluster Controller (CC) e também a opção de injetar falhas no Hardware. Está versão do Injetor não dá suporte ao Walrus. Na opção Password colocamos a senha cadastrado no usuário Front ou Node. A opção *TestDuration* é tempo de duração do teste (1 dia, 2 dias e etc); Distribution é o tipo de número de distribuição aleatória que o usuário deseja utilizar no teste. Na classe

injector ocorre a injeção de falhas e reparos nos componentes da nuvem, essa classe possui o kernel responsável pelos números pseudorrandômicos, na qual atribui-se o rand *VariateGeneration*, e a conexão SSH2 através do sshCommand. É de responsabilidade do kernel encerrar os processos que geram os eventos ao final da execução, esses valores são disparados levando em consideração a probabilidade exponencial. É na classe injector que se verifica a inicialização através do InicializarComponentes de injeção de falha transitionFailure e reparo de sistema transitionRepair. Além dos status que verifica se os componentes estão ativos na rede com o uso do *isAlive*.

Caso a estrutura de teste criada pelo o usuário possua Máquinas Virtuais (VM) em sua estrutura, o injetor possui um método que verifica se as mesmas estão ativas, tanto na injeção de falha quanto no reparo. Visto que a VM está em uma camada virtualizada e o processo para verificação é mais complexo do que o das máquinas físicas, os comandos são: *isVM* e *eucaCommand*.

A classe monitor é responsável por informar ao usuário o que ocorre na rede durante a injeção de falha e reparos. Ela informa as ocorrências de falha e reparo dos componentes da nuvem. Estas ações transcorrem em tempo real; ao final da injeção de falha, é transmitido para o usuário um relatório com todas as ocorrências da nuvem durante o processo. Nesta classe, encontramos o atributo *Components* que está relacionado aos componentes da nuvem (*Frontend, Node e VM*); ele mesmo é responsável por realizar a verificação destes módulos. O temporizador indica o tempo em que o monitor percorre cada componente, averiguando o status da rede. Esta ação acontece a cada 6 segundos aproximadamente. O *geradatas* informa no final da injeção de falha os dados do sistema.

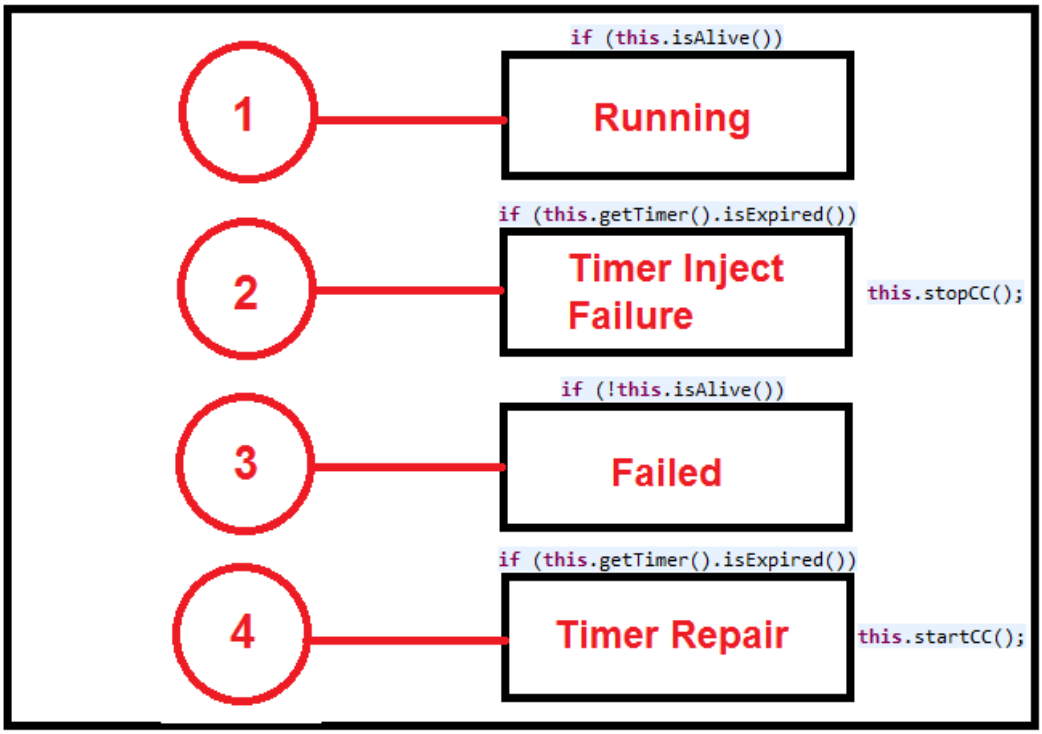


**Figura 6:** Modo monitoramento do Injetor

Ao analisarmos a Figura 10, podemos depreender o funcionamento do monitoramento do Injetor. Após o início do processo de injeção de falha e reparo, considerando que o sistema alvo via IP está configurado e operante, o tempo total de execução começa a ser contabilizado, o injetor injeta uma falha no Nó, e fica monitorando o estado do sistema, após o Nó parar de responder o injetor contabiliza uma falha na variável contador de falhas, injeta o comando de reparo no componente e fica contando esse período de tempo que o nó passa inoperante. O Tempo total de execução, tempo perdido e o número de paradas do Nó, são dados essenciais para realizar o cálculo do MTBF. Feito isso, o injetor verifica se o Nó voltou a responder, caso o resultado seja positivo será contabilizado na variável contador de reparos.

### 3.1.2 Obtendo o Tempo de Falha e Reparo (MTBF e MTTR)

Para darmos início a todos os dados que serão capturados pelo monitor precisamos entender o que é, e como funciona dois atributos chaves para este

processo. O MTBF (*mean time between failures* ou tempo médio entre falhas) e MTTR (*mean time to repair* ou tempo médio para reparo) são dois indicadores relacionado à disponibilidade de uma aplicação. Apesar de sua relevância no desempenho de processos, muitos gestores sub utilizam esses indicadores de desempenho em suas atividades de controle.

O MTBF é a métrica que se refere à média de tempo transcorrido entre uma irregularidade e o próximo lapso. Essas falhas de tempo podem ser prognosticadas por meio de uma fórmula. Já o indicador de MTTR refere-se ao prazo médio que demora para realizar uma correção depois da eventualidade, o erro. Ou seja, é o tempo gasto durante a intervenção em um determinado processo.

Os dois índices são empregados como base referencial para tomar decisões nas organizações. O objetivo de uma operação é sempre aumentar o MTBF e diminuir o MTTR.

### 3.1.3 (Mean Time Between Failures) – MTBF

MTBF são os intervalos de tempo que se perdem na operação de um determinado dispositivo, e sua média pode ser realizada por meio de uma fórmula. Devemos aplicar o tempo total do desempenho natural durante um ciclo preestabelecido sob o número de erros ocorridos durante esse tempo.

Fórmula: MTBF = (Tempo total disponível – Tempo perdido) / (Número de paradas)

Exemplo: Ao longo de um determinado período disponível para atuar, foi notado: período integral disponível para operar = 48 horas de teste aconteceram 3 paralisações a cada uma delas: 12 minutos, 6 minutos e 23 minutos

MTBF = [48 – (1 + 3 + 2)] / 3 = 14,3333 horas ou 859 minutos

Tendo essa conclusão, pode-se criar estratégias para enfrentar um problema paulatinamente associado ao equipamento.

### 3.1.4 (Mean Time to Repair) - MTTR

O MTTR é calculado contabilizando o tempo que um sistema passa sem funcionar após um evento de falhas até o momento que ele volta a funcionar.
Fórmula: MTTR = (Tempo total de reparo) / (quantidade de falhas)
Caso use o exemplo mostrado no MTBF, deverá obter a seguinte solução:
MTTR = (4 + 5 + 2) / 3 = 3,6666 horas ou 219 minutos
No exemplo acima temos o tempo que três dispositivos analisados ficaram sem funcionar durante o período de teste, primeiro somamos os três períodos de tempo e dividimos pela sua quantidade, com base nisto obtemos o resultado em horas do tempo que o sistema passou falhando, ou seja sem funciona.

### 3.1.5 Disponibilidade

Os atributos de dependabilidade estão ligadas às metas a serem alcançadas por sistemas que almejam ser considerados confiáveis. Um destes atributos, a disponibilidade, recebe um destaque maior em sua explanação, por ser um dos focos deste trabalho.

A disponibilidade pode ser definida como a quantificação da alternância entre a provisão correta e incorreta do serviço em questão. Ou seja, na avaliação de disponibilidade, o que se leva em consideração são os eventos que podem levar à falha, tal como eventos que correspondem aos esforços de manutenção empreendidos para levar o sistema novamente ao funcionamento correto (OLIVEIRA(2014a)). Sendo assim, podemos dizer que a disponibilidade de um sistema pode ser entendida como a probabilidade de que ele esteja operacional durante um determinado período de tempo, ou tenha sido restaurado após a ocorrência de um evento de falha (DANTAS (2013)).

Para que a disponibilidade de um sistema seja calculada precisamos dos indicadores que foram citados no tópico anterior. Com a aplicação deles na forma entenderemos melhor com esse cálculo é realizado.

**Disponibilidade** = (MTTF / MTTF+MTTR)

Onde:

- MTTR é o tempo de reparo do sistema;

- MTTF é o tempo de falha do sistema;

Seguindo essa lógica a função que calcula a disponibilidade foi implementada no Injetor, como podemos ver na figura abaixo:

```
//Calcula disponibilidade
double disponibilidade = (mttf/(mttf+mttr));
String availability = String.format("%.4f", (disponibilidade*100));
WriteFile.logger("\n", "ResultsNode_log.txt");
WriteFile.logger("Current Availability: " + availability + "%", "ResultsNode_log.txt");
WriteFile.logger("\n", "ResultsNode_log.txt");
```

**Figura 7:** Função que calcula a Disponibilidade

A função disponibilidade funciona da seguinte forma, após o início do processo de injeção de falha e reparo, considerando que o sistema alvo via IP está configurado e operante, ao iniciar o processo de injeção o tempo total de execução começa a contabilizado, o injetor injeta uma falha no Nó, e fica monitorando o estado do sistema, após o Nó parar de responder ele contabiliza uma falha a variável contador de falhas, injeta o comando de reparo no componente e fica contato esse período de tempo que o nó passa inoperante. O Tempo total de execução, tempo perdido e o número de paradas do Nó, são dados essenciais para realizar o cálculo do MTBF. Feito isso, o injetor verifica se o Nó voltou a responder, caso o resultado seja positivo será contabilizado na variável contador de reparos.

```
case FAILED:
    if (!this.isAlive()) {
    int waitingTimeDown = this.generateRandomRepairTime();
    this.setTimer(new MyTimer(waitingTimeDown));
    long segundosDown = ( waitingTimeDown / 1000 ) % 60;
    this.setState(StateMachineEnum.TIMER_REPAIR);
    WriteFile.logger("Generated Repair Time: " + segundosDown, "Node_log.txt");
    WriteFile.logger(new Date().toString(), "Node_log.txt");
    } else {
    // Durma novamente até o CC parar
    }
    break;
```

**Figura 8:** Verificando se a falha foi injetada

```
case TIMER_REPAIR:
    if (this.getTimer().isExpired()) {
        this.startCC();
        int waitingTimeR = this.generateRandomRepairTime();
        this.setTimer(new MyTimer(waitingTimeR));
        WriteFile.logger("Repaired. Node: " + this.getSshConnection().getHost(), "Node_log.txt");
        WriteFile.logger(new Date().toString(), "Node_log.txt");
        Thread.sleep(30000);

        if(this.isAlive()){
            WriteFile.logger("Repaired - "+ new Date().toString(), "Node_log.txt");
            Thread.sleep(6000);
        }
```

**Figura 9:** Injetando o Reparo

Com base nestes dois tempos MTTF e MTTR podemos calcular e obter o valor da disponibilidade.

### 3.1.6 Indisponibilidade

Em razão do uso disseminado dos sistemas de informação, a indisponibilidade ou desempenho insuficiente dos serviços providos têm sido objetos de atenção particular dos provedores de serviços, infraestruturas, projetistas de aplicações e da comunidade científica.

O Planejamento de Capacidade (*Capacity Planning - CP [All08]*) consiste em uma definição dos recursos necessários para que os serviços providos (realizados) pelos sistemas atendam os níveis de qualidade (tempo de resposta, nível de disponibilidade etc) adequados aos processos que utilizam estes serviços ou recursos, considerando demandas futuras. De fato, para que esta atividade seja realizada com sucesso, é necessário que os recursos sejam dimensionados de maneira que os custos associados sejam minimizados e os níveis de serviços estabelecidos sejam atendidos. Por causa desses fatores, também foi implementada no Injetor a função que calcula o tempo de indisponibilidade dos sistemas analisados.

Nesse exemplo pegamos a disponibilidade do sistema que é de 95,50% subtraímos por 100 que seria o valor total que a disponibilidade poderia atingir, e encontramos o valor da indisponibilidade, tempo que o sistema ficou inoperante. O resultado desse cálculo seria indisponibilidade atual de 4,50%.

**Indisponibilidade =** (**1** - Disponibilidade)

Onde:

- 1 é a parte total ou seja 100%;
- Disponibilidade é o tempo que o sistema passou operante;

```
//Calcula indisponibilidade
double indisponibilidade = (1-disponibilidade);
String unavailability = String.format("%.2f", (indisponibilidade*100));
WriteFile.logger("Current Unavailability: " + unavailability + "%", "Results_log.txt");
WriteFile.logger("\n", "Results_log.txt");
```

**Figura 10:** Função que calcula a Indisponibilidade

De acordo com a figura acima, a função que calcula a indisponibilidade é simplesmente duas linhas de código, uma que recebe o valor da disponibilidade do sistema e faz a subtração e outra que multiplica o resultado por 100 para entregar ao usuário o resultado da indisponibilidade em porcentagem facilitando o seu entendimento.

### 3.1.7 Tempo Up

O tempo up é responsável por calcular o tempo pelo qual o sistema passou funcionando sem interrupções ocasionadas por falhas ou por inatividade elétrica. Para realizar esse cálculo, pegamos a disponibilidade do sistema e multiplicamos pelo tempo total de execução do sistema, que, neste exemplo, o tempo foi calculado em segundo, como é possível visualizar na figura abaixo:

**Tempo Up =** (**Disponibilidade *** Total de Unidade de Tempo)

Onde:

- Disponibilidade e o tempo que o sistema passou operante;
- Total de unidade de tempo é o período de execução do sistema.

```
//Calcula tempoUp
double tempoUp = (disponibilidade*tempoTotalSegundos);
String timeUp = String.format("%.3f", tempoUp);
WriteFile.logger("Current TimeUp: " + timeUp, "ResultsNode_log.txt");
WriteFile.logger("\n", "ResultsNode_log.txt");
```

**Figura 11:** Função que calcula o Tempo Up

### 3.1.8 Tempo Down

A seguir veremos a fórmula matemática em que o cálculo do tempo que o sistema passa inoperante durante o processo de injeção é realizada, bem como o tempo de indisponibilidade do sistema, que sempre é contabilizado; o tempo é multiplicado pelo Tempo Total de execução do processo de injeção.

**Tempo Down =** (**Indisponibilidade *** Total de Unidade de Tempo**)**

Onde:

- Indisponibilidade e o tempo que o sistema passou inoperante;
- Tempo Total é o tempo de execução do sistema;

```
//Calcula tempoDown
double tempoDown = (indisponibilidade*tempoTotalSegundos);
String timeDown = String.format("%.3f", tempoDown);
WriteFile.logger("Current TimeDown: " + timeDown, "ResultsNode_log.txt");
WriteFile.logger("\n", "ResultsNode_log.txt");
```

**Figura 12:** Função que calcula o Tempo Down

### 3.1.9 #9s - Alta disponibilidade

A alta disponibilidade é caracterizada por ser uma forma alternativa de especificar a disponibilidade do sistema. Essa representação é particularmente útil quando o sistema tem disponibilidades com diversos noves. Para não se perder com esses números próximos de 1, reescreve-se a disponibilidade em termos dos noves. A notação #9s é muito usada em sistemas de alta disponibilidade (Datacenters, Plantas elétricas e etc.)

**#9s = LOG10 (Indisponibilidade)**

Onde:

- #9s é anotação dada a fórmula;
- Logaritmo de base 10 de Indisponibilidade. É a potência 10 elevada para Indisponibilidade;

- Indisponibilidade e o tempo que o sistema passou inoperante;

```
//Calcula #9s - Alta disponibilidade
double altaDisponibilidade = (-Math.log10(1-disponibilidade));
String high_availability = String.format("%.3f", altaDisponibilidade);
WriteFile.logger("Current #9s: " + high_availability, "ResultsNode_log.txt");
WriteFile.logger("\n", "ResultsNode_log.txt");
```

**Figura 13:** Função que calcula o #9s – Alta Disponibilidade

## 3.2 Integração com FlexLoadGenerator

O FlexLoadGenerator é um framework que visa fornecer ao desenvolvedor uma infraestrutura que apoie a criação de ferramentas geradoras de eventos.

Para o desenvolvimento do Injetor na primeira etapa realizou-se um estudo para escolha e integração com o FlexLoadGenerator que se adaptasse da melhor maneira à estrutura do injetor e os cenários de testes que programa seria utilizado. Além disso, devido à compatibilidade com o projeto e suas funcionalidades já implementadas, este fato colaborou para diminuir o tempo de desenvolvimento do projeto e minimizar a complexibilidade na implementação do código. Existem ferramentas que também utilizam esse Framework, porém com outra finalidade, como por exemplo, a ferramenta *EUCAMBOBER* que o utiliza para geração de carga sintética. Para testar o Injetor optamos pelo Framework Eucalyptus, que por sua vez, é uma ferramenta bastante utilizada para esses tipos de testes dentro do grupo de pesquisa Modcs. A ferramenta é utilizada para simular uma rede de máquinas reais, oferecendo ao usuário interface gráfica via web para implementação das VMs. O diferencial do Framework está em sua capacidade de geração de eventos, falhas e reparos; analisar os dados gerados e calcular em tempo de execução a Disponibilidade, Indisponibilidade, Quantidade de falhas, o tempo que o sistema passa UP e Down, o tempo total de execução da injeção de falhas, e por fim, o #9s que é o cálculo da disponibilidade de sistemas com alta disponibilidade. O usuário pode injetar falha em uma infraestrutura de nuvem antes de iniciar a injeção de falhas. É importante salientar que o usuário forneça ao injetor todos os IPs cadastrados em sua rede teste, para que ele inicie o processo. Cada Nó está associado a um IP (Internet Protocol) de uma máquina de infraestrutura de nuvem. Antes da injeção de falha, o Injetor inicia o processo de comunicação com o sistema, verificando se cada componente está disponível e, em seguida, inicia o processo de

injeção. Neste trabalho, decidimos inserir e reparar falhas. Os reparos são selecionados no sistema alvo em tempo de execução para desencadear ambos os eventos.

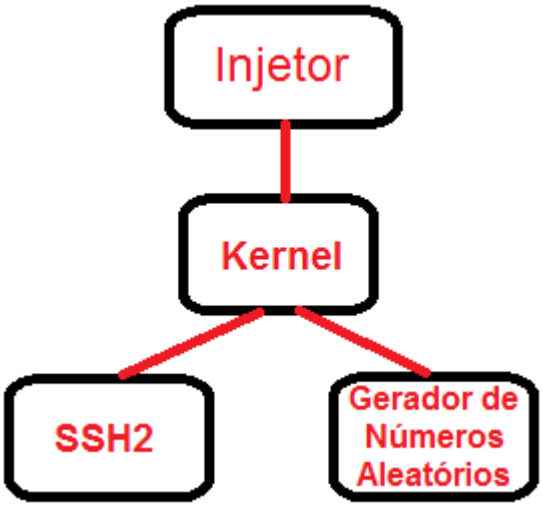


**Figura 14:** Hierarquia de composição do Injetor

Por conseguinte, os eventos gerados utilizaram o kernel para emular o tempo que injetor deve esperar entre falhas e reparos. O tempo entre os eventos foi provocado com base no gerador de números aleatórios, que utiliza a distribuição exponencial; contudo, o injetor também fornece outras opções de geração de números aleatórios, a título de exemplo: *TRIANGULAR, NORMAL e UNIFORM*. As unidades de tempo para a ocorrência dos eventos estão disponíveis em meses, dias, horas, minutos, segundos ou milissegundos. A unidade de tempo usada pelo injetor de falha foi de milissegundos e a distribuição de probabilidade utilizada foi exponencial. O tempo é importante porque as variáveis aleatórias são usadas como um atraso entre um estado do sistema e outro, até o próximo evento ser disparado. As falhas são independentes umas das outras. A única dependência existente entre as falhas é sua ação de reparo. O Injetor emula e simula uma injeção de falhas em um ambiente real.

O monitor do framework é responsável por supervisionar os serviços da plataforma de nuvem computacional e informar ao usuário através de mensagens se as falhas estão ocorrendo no sistema, a fim de que o pesquisador possa analisar e observar o comportamento da nuvem. Estas informações facilitam o entendimento das condições do ambiente, além de informar sobre possíveis erros. O monitor

informa o componente da nuvem juntamente com o IP no qual está ocorrendo a falha; informa também as ocorrências de falha e reparo, contabilizando-as, e a data; além de informar o tempo de atividade (UP) e inatividade (Down) do sistema. O monitor possui o importante papel de efetuar a situação do sistema e, dessa forma, apresenta todos os resultados obtidos a partir de uma interface. Isto significa que a aplicação possui a capacidade de verificar constantemente o estado das máquinas presentes na rede, como também os respectivos serviços da nuvem.

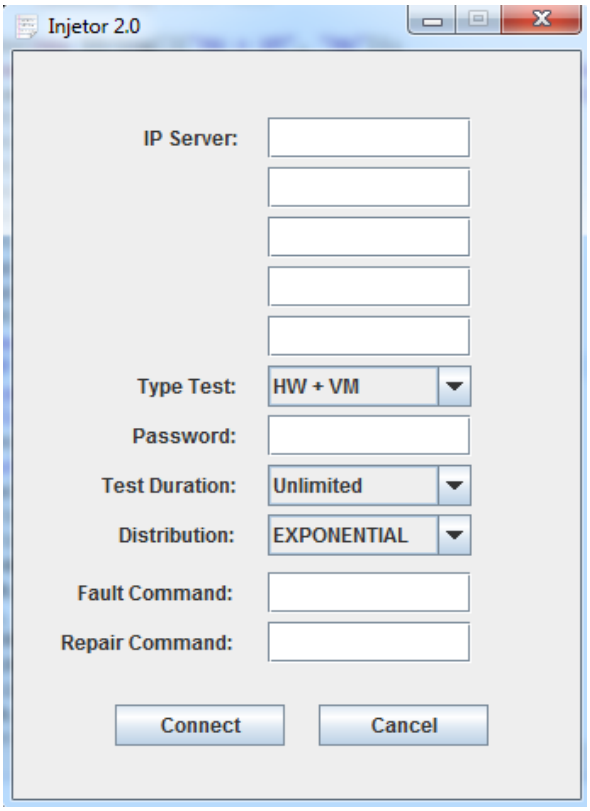


**Figura 15:** Interface do Injetor

Como mostra a figura 9, o Injetor tem duas importantes funções as que o torna funcional em qualquer dispositivo via IP que suportem comandos de falha e reparo, as funções são *Fault Command* e *Repair Command.*

*Fault Command* é onde o usuário deve inserir o comando de falha do componente que ele deseja injetar a falha, lembrando que só será necessária a informação de um novo comando, caso o cenário de teste seja diferente da plataforma *Eucalyptus.*

*Repair Command* é onde o usuário deve inserir o comando de reparo do componente. Informando esses dois comandos será possível iniciar o processo de

injeção de falhas e reparos na nova plataforma. Caso os comandos inseridos não estejam corretos, irá ocorrer um erro no processo de injeção e a ferramenta será encerrada automaticamente.

É importante que o usuário verifique antes de iniciar o processo de injeção, se o sistema alvo está de fato na mesma rede do Injetor para que não ocorra falha na comunicação.

### 3.2.1 Descrição do FlexLoadGenerator

O Framework FlexLoadGenerator oferece o meio de comunicação com o sistema alvo através dos protocolos TCP e UDP, e o gerenciamento da ocorrência de eventos. A ferramenta consiste em três partes fundamentais: módulo de comunicação SSH2, módulo de geração de números aleatórios e o uso de distribuição de probabilidade. O módulo de conexão é responsável por prover a comunicação entre o injetor de falhas e a máquina alvo. A conexão é estabelecida usando o protocolo SSH2 que permite que comandos sejam enviados diretamente ao shell do sistema operacional Ubuntu Linux presente nos servidores que compõem a nuvem. O módulo de geração de números aleatórios é responsável por gerar números pseudoaleatórios que seguem pelo menos uma dentre as várias distribuições de probabilidade disponíveis. A ferramenta utiliza estes valores como intervalo de tempo (time-out) para a ocorrência dos eventos. As distribuições de probabilidade são uma biblioteca de geração de números aleatórios, e é responsável pelo auxílio no gerenciamento entre a ocorrência de disparos de eventos. A biblioteca gera amostras com valores aleatórios baseados em algumas das principais distribuições de probabilidade existentes (contínuas e discretas), através da aplicação de técnicas de geração de variáveis aleatórias (*random variates*) (DEVROYE, 1986). As distribuições de probabilidade mais amplamente utilizadas nas áreas de avaliação de desempenho e dependabilidade de sistemas computacionais (JAYN, 1991; HAYTER, 2007) estão executadas nesta biblioteca; são elas: Erlang, Exponencial, Log-normal, Normal, Pareto, Triangular, *Weibull* e Uniforme (contínuas), e Geométrica e *Poisson* (discretas).

### 3.2.2 Análise dos Resultados

Neste capítulo, será apresentada uma análise dos resultados obtidos pelo Injetor através do processo de injeção de falhas.

Foram realizados testes de injeção de falha e reparo no ambiente com a plataforma Eucalyptus como ferramenta de teste. O processo teve duração de 4 dias, os resultados foram positivos no quesito disponibilidade acima dos 50%, a disponibilidade ficou maior que a indisponibilidade, porém, podemos perceber que o sistema no período de 96 horas de teste passou 43 horas e 17 minutos sem funcionar, com essas informações podemos concluir que das 96 horas de teste o sistema passou apenas 53:23 horas em pleno funcionamento e mesmo que o resultado tenha atingindo um valor maior que a Indisponibilidade, podemos afirmar que a estrutura de rede testada não apresentou uma confiabilidade aceitável.

### 3.2.3 Processo de injeção de falhas

Neste capítulo, compreenderemos como o algoritmo do injetor se comporta ao iniciar o processo de injeção de falhas dentro de um ambiente de testes.

O primeiro passo foi verificar se o sistema que seria testado estava respondendo. Como veremos na imagem abaixo, pode-se visualizar a funcionalidade da linguagem Java, chamada *switch case*. Com ela podemos verificar o status real do sistema. Dentro dessa classe são realizadas quatro comparações: *RUNNING, TIMER_INJECT_FAILURE, FAILED e TIMER_REPAIR*.

1. O primeiro status é denominado *RUNNING.* Dentro desse estado, verificamos se o sistema está ativo, como mostra o primeiro *IF* do código abaixo. Caso o estado do sistema seja *isAlive()*, será executado função de injetar falha, aguardando o sistema cair e gerar o tempo de falha.

```
public void runCCStateMachine() throws InterruptedException, IOException {
    switch (this.getState()) {
    case RUNNING:
        if (this.isAlive()) {
            int waitingTime = this.generateRandomFailureTime();
            this.setTimer(new MyTimer(waitingTime));
            WriteFile.logger("Started. Node: " + this.getSshConnection().getHost(), "Node_log.txt");
            WriteFile.logger(new Date().toString(), "Node_log.txt");
            this.setState(StateMachineEnum.TIMER_INJECT_FAILURE);
            } else {
            // Durma novamente até que o CC comece
            }
```

**Figura 16:** Função Running

2. O estado *TIMER_INJECT_FAILURE* é responsável por checar se a falha foi injetada com sucesso.

```
case TIMER_INJECT_FAILURE:
    if (this.getTimer().isExpired()) {
        this.stopCC();
        int waitingTime = this.generateRandomFailureTime();
        this.setTimer(new MyTimer(waitingTime));
        WriteFile.logger("Inject Fault. Node: " + this.getSshConnection().getHost(), "Node_log.txt");
        WriteFile.logger("Time Failure: " + waitingTime, "Node_log.txt");
        WriteFile.logger(new Date().toString(), "Node_log.txt");
        Thread.sleep(30000);
        this.setState(StateMachineEnum.FAILED);
        } else {
        // Sleep again until the timer expires
        }
        break;
```

**Figura 17:** Função Timer Inject Failure

O *this.stopCC()* chama a função com o código de parada do Nó, é nesta função que o usuário deve colocar o código do sistema que ele deseja injetar falha.

A variável waitingTime é responsável por receber o tempo gerado pela distribuição de números aleatórios, é com base neste valor que o injetor deve esperar até função de reparo do Nó ser executada novamente, o tipo de distribuição pode ser escolhido pelo usuário no início do processo na interface do próprio Injetor. Neste exemplo foi utilizado a distribuição exponencial.

3. O *FAILED* checa se o sistema realmente falhou e se parou de responder.

```
case FAILED:
    if (!this.isAlive()) {
        int waitingTime = this.generateRandomFailureTime();
        this.setTimer(new MyTimer(waitingTime));
        this.setState(StateMachineEnum.TIMER_REPAIR);
        WriteFile.logger("Failed. Node: " + this.getSshConnection().getHost(), "Node_log.txt");
        WriteFile.logger("Time Failure: " + waitingTime, "Node_log.txt");
        WriteFile.logger(new Date().toString(), "Node_log.txt");
        } else {
        // // Sleep again until the CC stops
        }
    String statusDown = this.getState().toString();
    System.out.println(statusDown);
    while (!this.getTimer().isExpired()){
        WriteFile.logger("Down - "+ new Date().toString(), "Node_log.txt");
        Thread.sleep(8000);
    }
        break;
```

**Figura 18:** Função Failed

4. Por fim, temos o *TIMER_REPAIR* que injeta um novo comando para reparar a falha e fazer o sistema voltar a funcionar.

```
case TIMER_REPAIR:
    if (this.getTimer().isExpired()) {
        this.startCC();
        int waitingTimeR = this.generateRandomRepairTime();
        this.setTimer(new MyTimer(waitingTimeR));
        WriteFile.logger("Time Failure: " + waitingTimeR, "Node_log.txt");
        WriteFile.logger("Repaired. Node: " + this.getSshConnection().getHost(), "Node_log.txt");
        WriteFile.logger(new Date().toString(), "Node_log.txt");
        Thread.sleep(30000);
        this.setState(StateMachineEnum.RUNNING);
    }
```

**Figura 19:** Função Timer Repair

O *this.startCC()* chama a função com o código para iniciar novamente o serviço do Nó, é nesta função que o usuário deve colocar o código do sistema que ele deseja reparar a falha.

A variável waitingTimeR é responsável por receber o tempo gerado pela distribuição de números aleatórios, é com base neste valor que o injetor deve esperar até iniciar o Nó novamente, o tipo de distribuição pode ser escolhido pelo usuário no início do processo na interface do próprio Injetor. Neste exemplo foi utilizado a distribuição exponencial.

# 4 Estudo de Caso

Neste capítulo, serão apresentados os estudos e validações que foram realizados através de Máquinas com VMs instanciadas por um servidor software Eucalyptus instalado.

Posteriormente, foi instalado o Sistema Operacional Centos OS versão 6.5 nas duas máquinas disponíveis; foram instanciadas duas VMs uma de cada máquina para aumentar a estrutura de teste da rede. A estrutura da rede foi configurada da seguinte maneira: Máquina Front com IP 192.168.0.148, Máquina Node com IP 192.168.0.150 seguindo a VM1 instanciada através do Front ficou com o IP 192.168.0.149. E por fim, a VM2 instanciada pelo Node com IP 192.168.0.151. Essas informações foram passadas para o Injetor para dar início ao processo de injeção de falhas.

## 4.1 Ambiente de Teste – Eucalyptus

A validação do Injetor foi realizada uma rede doméstica com o sistema Eucalyptus utilizado como ferramenta de teste, por se tratar de uma ferramenta que suporta comando de falha e reparo e que foi desenvolvida com esse intuito.

O comando utilizado para inserir a falha no Nó foi “service eucalyptus-nc stop” esse comando simula uma parada no componente injetado evitando que usuário tenha que esperar que uma falha real do sistema ocorra, o que tornaria o teste inviável em relação ao tempo que seria necessário para validar o Injetor.

O mesmo ocorre quando precisamos reparar o Nó, para isso inserimos o comando “service eucalyptus-nc start” que tem como resultado trazer de volta a ativa o componente que estava parado.

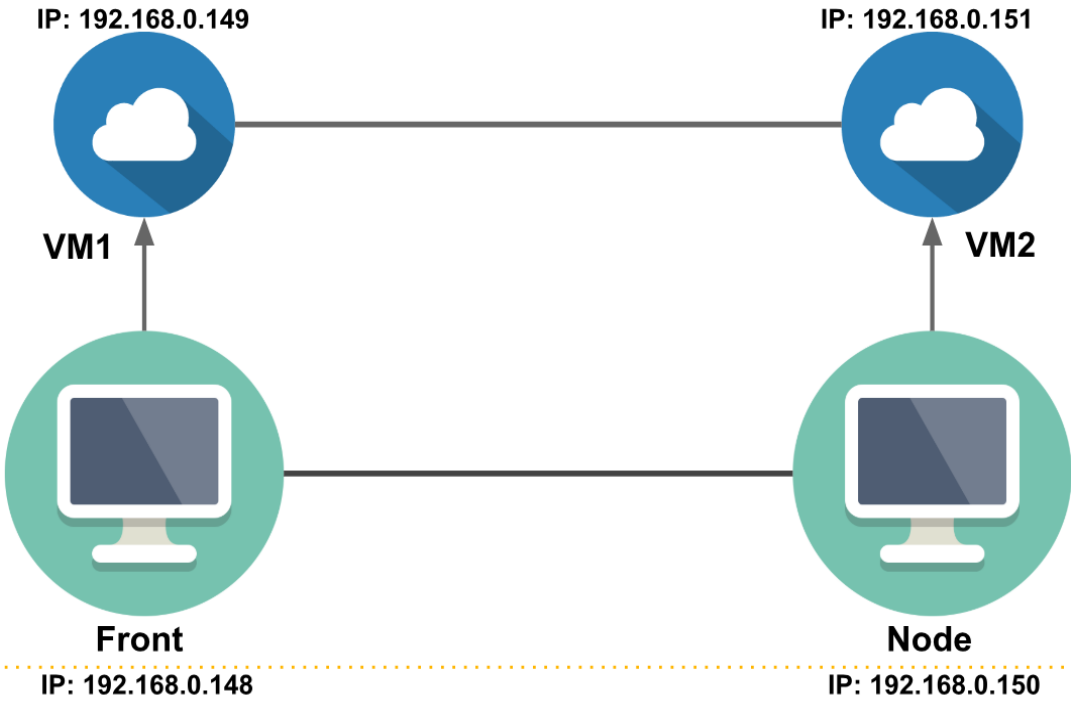


**Figura 20:** Estrutura da rede de teste

Após testes contínuos de no mínimo 2 dias cada, foi constatado que de fato as informações passadas pelo injetor seguem todos os métodos científicos deste trabalho e que o Framework pode ser utilizado para testes não só no Eucalyptus como foi mostrado no cenário de teste, ele pode ser utilizado em qualquer outro sistema via IP, com a condição de que será necessário pesquisar os comandos necessários anteriormente de início e parada, para serem implementados no código do injetor.

## 4.2 Resultados

Neste tópico serão explanados os resultados obtidos após o último teste realizado com o Injetor de falhas desenvolvido neste trabalho no ambiente com a ferramenta de teste Eucalyptus.

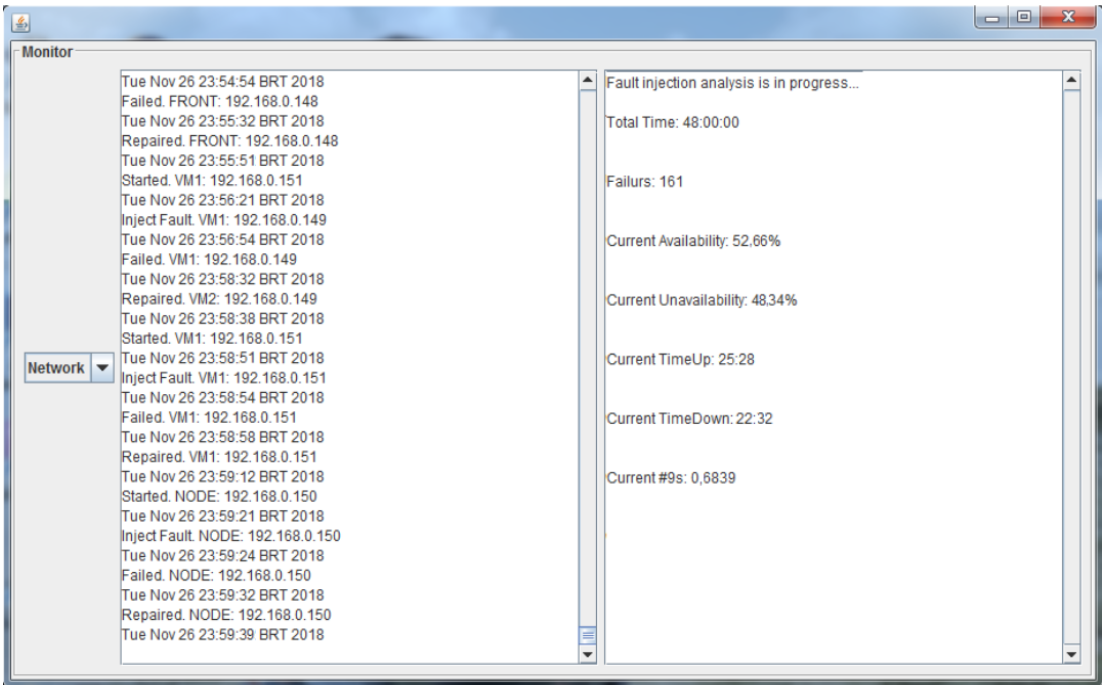


**Figura 21:** Resultado obtidos no ambiente de teste

O teste teve duração de 48 horas, neste período podemos constatar através do Monitor que ocorreram 161 falhas, valor que representa a quantidade de vezes que o sistema parou de funcionar, seguindo pode ver que mesmo a disponibilidade tendo seu resultado maior que a indisponibilidade a rede não se comportou de forma satisfatória, apresentando apenas 52,66% de disponibilidade. A gravidade disso pode ser compreendida de melhor forma nas informações abaixo, que diz respeito ao tempo total que o sistema passou disponível e indisponível, que neste cenário foi de 1528 minutos disponível, o que significa que no período de 48 horas que representa 2880 minutos, o sistema passou 1352 minutos inoperante. Os resultados podem ser melhor visualizados na tabela abaixo:

| | |
|---|---|
| **Tempo de Injeção** | 48 Horas |
| **Disponibilidade** | 52,66% |
| **Indisponibilidade** | 48,34% |
| **Tempo UP** | 25:28 |
| **Tempo Down** | 22:32 |
| **#9s** | 0,6839 |

# 5 Considerações Finais

Presentemente, existem diversas ferramentas que auxiliam o usuário em processos de injeção de falhas, mas nenhuma desenvolvida com o intuito de funcionar em vários ambientes de teste e que forneça os resultados apresentados neste trabalho. Desde o princípio, a proposta deste trabalho foi desenvolver um injetor de falhas que permitisse a diminuição do retrabalho dos pesquisadores e desenvolvedores. Tendo isso em vista, o injetor foi criado a partir de um Kernel que foi desenvolvido com o intuito de auxiliar na criação de novas ferramentas de injeção de falhas e avaliação de desempenho, tendo como base classes prontas de algoritmos de distribuição aleatória e comunicação com sistemas de teste através do protocolo SSH2. Além disso ela conta com a opção de receber códigos de falha e reparo diretamente na interface, o que a torna adaptável a outros ambientes de teste que suportem está metodologia. Em consequência desses fatos, a ferramenta poderá ser utilizada por um longo período sem a necessidade de mudanças em sua estrutura. Há grande expectativa de que o Injetor seja usado pela comunidade científica, por esse motivo, o código se tornou *Open Source* e está disponível para todos os membros do *GitHub* para que possam utilizar e contribuir com o seu crescimento. Para o futuro, pretendo lançar a versão 3.0 da ferramenta com uma interface gráfica aprimorada e com novos recursos que serão adicionados de acordo com a necessidade dos usuários.

A eficiência da ferramenta foi constatada dentro de um cenário no qual as falhas e reparos foram injetadas em uma estrutura de nuvem configurada através do sistema Eucalyptus, deixando claro que a ferramenta não limita-se apenas a este cenário. Os resultados dos testes foram analisados ao final de cada execução, seguindo de forma exata as fórmulas citadas em artigos científicos indicados pelo professor Paulo Maciel, demonstrando que o processo de injeção de falhas comportou-se como o esperado no cenário estudado. Os resultados obtidos através do Monitor mostram que de fato injetor pode ajudar os pesquisadores e desenvolvedores a avaliarem a disponibilidade, indisponibilidade de seus sistemas e dessa forma agilizar a obtenção dos resultados.

# REFERÊNCIAS


ARAUJO, Jean et al. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In: **Proceedings of the Middleware 2011 Industry Track Workshop**. ACM, 2011. p. 4. Disponível em: https://sci-hub.tw/10.1145/2090181.2090185 . Acesso em: 08 dez. 2018

ESTANISLAU SOARES GALINDO, Hugo. Geração de carga sintética para o planejamento de capacidade em ambientes de servidores virtuais. Mar. 2010. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/2420/1/arquivo3248_1.pdf. Acesso em: 02 dez. 2018.

JAIN, Raj. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. New Jersey: John Wiley & Sons,1990. Disponível em: https://sci-hub.tw/10.1093/comjnl/35.3.207 Acesso em: 08 dez. 2018

MACIEL, Paulo RM et al. **Dependability modeling**. In: Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions. IGI Global, 2012. p. 53-97. Disponível em: https://bit.ly/2Ge129U. Acesso em: 12 dez. 2018.

MANFREDINI, Ricardo Augusto. **Condução de experimentos de injeção de falhas em banco de dados distribuídos**. jul. 2001. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/2474/000320304.pdf?sequence=1&locale=pt_BR. Acesso em: 02 dez. 2018.

SILVA, Bruno. **A framework for availability, performance and survivability evaluation of disaster tolerant cloud computing systems**. 2016. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/18035/1/Bruno_Silva_Doutorado_Ciencia_da_Computacao_2016.pdf . Acesso em: 02 dez. 2018.

SOUZA, Débora Stefani Lima de. **FlexLoadGenerator-Um Framework para Apoiar o Desenvolvimento de Ferramentas Voltadas a Estudos de Avaliação de Desempenho e Dependabilidade**. 2013. Disponível em: http://www.modcs.org/wp-content/uploads/wmodcs/2016-1/Debora-workshop-2016.1.pdf . Acesso em: 02 dez. 2018.

WANNER, Paulo César Herrmann**. Ferramenta de injeção de falhas para avaliação de segurança**. jul. 2003. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/2752/000376074.pdf?sequence=1&isAllowed=y. Acesso em: 02 dez. 2018.

OLIVEIRA, Gustavo Menezes. Injeção de falhas de comunicação em ambientes distribuídos. 2011. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/32864/000783416.pdf?sequence=1&isAllowed=y. Acesso em: 12 dez. 2018

ESTANISLAU SOARES GALINDO, Hugo. Geração de carga sintética para o planejamento de capacidade em ambientes de servidores virtuais. 2010. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/2420/1/arquivo3248_1.pdf. Acesso em: 12 dez. 2018.

VACARO, Juliano Cardoso; WEBER, Taisy Silva. FIRMI: Um Injetor de Falhas para a Avaliação de Aplicações Distribuídas baseadas em RMI. In: **Workshop de Testes e Tolerância a Falhas (2006 maio: Curitiba). Anais.[Porto Alegre]: Sociedade Brasileira de Computação, 2006.** 2006. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/6382/000529168.pdf?sequence=1&isAllowed=y. Acesso em: 12 dez. 2018.

SEGALL, Zary et al. Fiat-fault injection based automated testing environment. In: **Fault-Tolerant Computing, 1995, Highlights from Twenty-Five Years., Twenty-Fifth International Symposium on**. IEEE, 1995. p. 394. Disponível em: https://sci-hub.tw/10.1109/ftcs.1988.5306. Acesso em: 12 dez. 2018.

TSAI, Timothy; IYER, RaviShankar. FTAPE-A fault injection tool to measure fault tolerance. In: **10th Computing in Aerospace Conference**. 1995. p. 1041. Disponível em: https://sci-hub.tw/10.2514/6.1995-1041. Acesso em: 12 dez. 2018.

HAN, Seungjae; SHIN, Kang G.; ROSENBERG, Harold A. Doctor: An integrated software fault injection environment for distributed real-time systems. In: **Computer Performance and Dependability Symposium, 1995. Proceedings., International**. IEEE, 1995. p. 204-213. Disponível em: https://sci-hub.tw/10.1109/ipds.1995.395831. Acesso em: 12 dez. 2018.

LOVRIC, T.; ECHTLE, K. ProFI: Processor fault injection for dependability validation. In: **International Workshop on Fault and Error Injection for Dependability Validation of Computer Systems', Göteborg, Sweden**. 1993. Disponível em: https://sci-hub.tw/10.1109/FTCS.1989.105591. Acesso em: 12 dez. 2018.

DREBES, Roberto Jung. Firmament: um módulo de injeção de falhas de comunicação para linux. 2005. Disponível em: https://lume.ufrgs.br/bitstream/handle/10183/7196/000496578.pdf?sequence=1&isAllowed=y. Acesso em: 12 dez. 2018.

DAWSON, Scott; JAHANIAN, Farnam; MITTON, Todd. ORCHESTRA: A probing and fault injection environment for testing protocol implementations. In: **Computer Performance and Dependability Symposium, 1996., Proceedings of IEEE International**. IEEE, 1996. p. 56. Disponível em: https://web.eecs.umich.edu/~farnam/pubs/1996-djm-cpds.pdf. Acesso em: 12 dez. 2018.

MURAYAMA, Yukio; YAMAGUCHI, Suguru. DBS: a powerful tool for TCP performance evaluations. In: **Performance and Control of Network Systems**. International Society for Optics and Photonics, 1997. p. 570-582. Disponível em: https://pdfs.semanticscholar.org/8d5c/6b43d78f65104c8d7aa560fe66013940fb4e.pdf . Acesso em: 12 dez. 2018.