Pós-Graduação em Ciência da Computação

Breno José Ribeiro de Vasconcelos

MODELOS PARA AVALIAÇÃO DE DISPONIBILIDADE E CÁLCULO DE CAPACIDADE DE UM SOFTWARE DE COMPRESSÃO DE VÍDEO DISTRIBUÍDO EM NUVEM OPENSTACK

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2019

Breno José Ribeiro de Vasconcelos

MODELOS PARA AVALIAÇÃO DE DISPONIBILIDADE E CÁLCULO DE CAPACIDADE DE UM SOFTWARE DE COMPRESSÃO DE VÍDEO DISTRIBUÍDO EM NUVEM OPENSTACK

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Paulo Romero Martins Maciel
Co-orientador: Jamilson Ramalho Dantas

Recife
2019

Dissertação de Mestrado apresentada por **Breno José Ribeiro de Vasconcelos** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**MODELOS PARA AVALIAÇÃO DE DISPONIBILIDADE E CÁLCULO DE CAPACIDADE DE UM SOFTWARE DE COMPRESSÃO DE VÍDEO DISTRIBUÍDO EM NUVEM OPENSTACK**" **Orientador: Prof. Dr. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Dr. Divanilson Rodrigo de Sousa Campelo
Centro de Informática / UFPE

Prof. Dr. Gustavo Rau de Almeida Callou
Departamento de Computação / UFRPE

Prof. Orientador: Prof. Dr. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 01 de agosto de 2019.

Prof. Dr. Ricardo Bastos Cavalcante Prudêncio
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Dedico este trabalho à minha família, em especial a minha filha Maria Priscilla.

Agradecimentos

Primeiramente, agradeço à minha família sempre ao meu lado, ao meu orientador Paulo Maciel, ao meu coorientador Jamilson Dantas, a todos os colegas do grupo de pesquisa e meus amigos.

"Our family are an alternate stratification of poetry and mathematics."
(Ada Lovelace)

RESUMO

Alta disponibilidade é um atributo indispensável para garantir o aprovisionamento contínuo de serviços. Por isto, se faz necessária a aplicação de uma ou mais técnicas para o aumento desta métrica, como por exemplo, os mecanismos de redundância de nuvem investigados por este trabalho. Um problema atual é a compressão de vídeos em tempo real, que é essencial para empresas onde o *time-to-market* é vital. Esta dissertação propõe um estudo sobre a disponibilidade de uma arquitetura que utiliza a nuvem privada OpenStack como infraestrutura, e ainda o software Morph para realizar a compressão de vídeo distribuída. Primeiro, propomos modelos hierárquicos RBD (*reliability block diagram*) e SPN (stochastic Petri net) para representar uma arquitetura básica, sem redundâncias. Realizamos o cálculo de sua disponibilidade e, em seguida, validamos o modelo proposto. A validação foi assistida por uma ferramenta de injeção de falhas e reparo criada para este trabalho. Aplicamos uma análise de sensibilidade para identificar o gargalo na disponibilidade e propusemos novos modelos. Através destes novos modelos, representamos a redundância dos nós físicos e efetuamos a análise das disponibilidades. Ao final, apresentamos uma comparação entre as arquiteturas propostas. Os resultados alcançados apresentam ganhos na disponibilidade, devido ao fato da adição de redundância dos nós.

Keywords: Rede de Petri estocástica. Desempenho. Disponibilidade.

ABSTRACT

High availability is an indispensable attribute to guarantee continuous services provisioning. Thus, it becomes necessary to apply one or more techniques to increase this metric, for instance, the cloud redundancy mechanism investigated in this dissertation. A current problem is real time video compression, which is essential for companies where time-to-market is vital. This dissertation proposes a study about the availability of a OpenStack private cloud architecture. We adopted the Morph software to accomplish distributed video compression. First, we proposed hierarchical RBD (reliability block diagram) and SPN (stochastic Petri net) models to represent the baseline architecture without redundancies. We calculated its availability and then validated the model. The validation was aided by a failure and repair injection tool we created for this purpose. A sensitivity analysis was performed to identify the availability bottlenecks in this system. Taking that finding into consideration, we proposed other models to achieve higher availability. Through these models we represented the redundancy of the physical nodes and performed the availability analysis. Finally, we present a comparison between the proposed architectures in terms of their availability. The results accomplished in this dissertation show a gain in availability, due to the addition of node redundancy. Downtime was reduced over 90.0575%, going from 88.8292 hours to 8.8318 hours per year.

Key-words: Stochastic Petri net. OpenStack. Availability.

LISTA DE FIGURAS

Figura 1 – Rede de Petri 30
Figura 2 – Componentes das redes de Petri 32
Figura 3 – Rede de Petri Estocástica 32
Figura 4 – Análise de sensibilidade 34
Figura 5 – Arquitetura Morph 37
Figura 6 – Metodologia 44
Figura 7 – Arquitetura OpenStack-Morph com um nó 46
Figura 8 – Fluxograma do Modcs Injector 47
Figura 9 – RBD da arquitetura básica OpenStack-Morph 51
Figura 10 – Modelo RBD da Estrutura Geral 52
Figura 11 – Modelo RBD do subsistema FrontEnd 52
Figura 12 – Modelo RBD subsistema MVM 53
Figura 13 – Modelo RBD do subsistema MN 54
Figura 14 – Rede de Petri com um nó 55
Figura 15 – Arquitetura OpenStack-Morph com dois nós 57
Figura 16 – Rede de Petri com dois nós 58
Figura 17 – Rede de Petri com três nós 62
Figura 18 – Gráfico de distribuição de probabilidade 69
Figura 19 – MTTF do nó 72
Figura 20 – MTTR do nó 72
Figura 21 – MTTF da VM 72
Figura 22 – MTTR da VM 72
Figura 23 – MTTF do nó 75
Figura 24 – MTTR do nó 75
Figura 25 – MTTF da VM 75
Figura 26 – MTTR da VM 75
Figura 27 – Disponibilidade das arquiteturas 77
Figura 28 – COA das arquiteturas 78
Figura 29 – Disponibilidade da capacidade total 79

LISTA DE TABELAS

Tabela 1 – Comparativo entre trabalhos 25
Tabela 2 – Parâmetros de entrada para o injetor 49
Tabela 3 – Expressões de guarda do modelo SPN um nó 55
Tabela 4 – Expressões de guarda do modelo SPN com dois nós 59
Tabela 5 – Expressões de Guarda do modelo com três nós 65
Tabela 6 – Valores dos parâmetros 68
Tabela 7 – Alterações de estados 70
Tabela 8 – Valores da distribuição 70
Tabela 9 – Intervalos de confiança para γ e A 70
Tabela 10 – Parâmetros de entrada do modelo SPN com um nó 71
Tabela 11 – Resultados do modelo SPN de um nó 71
Tabela 12 – Ranking de sensibilidade baseado em derivadas parciais da arquitetura de um nó 73
Tabela 13 – Resultados da capacidade do modelo de um nó 73
Tabela 14 – Parâmetros de entrada da SPN com 2 nós 74
Tabela 15 – Resultados do modelo SPN com dois nós 74
Tabela 16 – Análise de sensibilidade da arquitetura com dois nós 75
Tabela 17 – Resultados da capacidade do modelo de dois nós 76
Tabela 18 – Parâmetros de entrada para modelo SPN com três nós 76
Tabela 19 – Resultados do modelo SPN com três nós 76
Tabela 20 – Disponibilidade dos nós 77
Tabela 21 – Resultados da capacidade do modelo de três nós 78

Lista de abreviaturas e siglas

COA	*Capacity-Oriented Availability*
VPN	*VPN*
Apache Web Server	*Apache Web Server*
CaaS	*Communication as a Service*
CTMC	*continuos-time Markov chains*
DaaS	*Desktop as a Service*
DBaaS	*Database as a Service*
GSPN	*Generalized Stochastic Petri Nets*
IaaS	*Infrastructure as a Service*
MC	*Motion Compensation*
MCP	*Motion-compensated prediction*
ME	*Motion Estimation*
MRM	*Markov reward model*
MTTF	*mean time to failure*
MTTR	*mean time to repair*
MVs	*Movie Vectors*
PaaS	*Platform as a Service*
PN	*Petri Nets*
RBD	*reliability block diagram*
SaaS	*Software as a Service*
SPN	*Stochastic Petri Nets*
VoD	*Video on Demand*
XaaS	*X as a Service*

SUMÁRIO

1 INTRODUÇÃO 13
1.1 MOTIVAÇÃO E JUSTIFICATIVA 15
1.2 OBJETIVOS 16
1.3 ORGANIZAÇÃO DA DISSERTAÇÃO 16

2 TRABALHOS RELACIONADOS 18
2.1 CONSIDERAÇÕES FINAIS 24

3 REFERENCIAL TEÓRICO 26
3.1 COMPUTAÇÃO EM NUVEM E OPENSTACK 26
3.1.1 Neutron 27
3.1.2 Nova 27
3.2 DEPENDABILIDADE 27
3.2.1 Reliability Block Diagram 28
3.2.2 Redes de Petri 29
3.2.3 Redes de Petri Estocásticas 29
3.3 ANÁLISE DE SENSIBILIDADE 33
3.4 MÉTODO KEESEE PARA CÁLCULO DO INTERVALO DE CONFIANÇA . 34
3.5 MORPH 36
3.6 INJEÇÃO DE FALHA E REPARO 37
3.7 COMPRESSÃO OU TRANSCODIFICAÇÃO DE VÍDEO 40
3.8 CONSIDERAÇÕES 42

4 METODOLOGIA, ARQUITETURAS DE NUVEM E INJETOR DE FALHAS 43
4.1 METODOLOGIA 43
4.2 ARQUITETURA OPENSTACK-MORPH 45
4.3 MODCS INJECTOR 46

5 MODELOS DAS ARQUITETURAS 50
5.1 RBD 50
5.2 MODELO DA ARQUITETURA 1 54
5.3 MODELO DA ARQUITETURA 2 57
5.4 MODELO DA ARQUITETURA 3 61
5.5 CONSIDERAÇÕES 65

6 ESTUDOS DE CASO 67

6.1 ESTUDO DE CASO I: VALIDAÇÃO 67
6.2 ESTUDO DE CASO II - ARQUITETURA BÁSICA 71
6.2.1 Disponibilidade da arquitetura I 71
6.2.2 Análise de sensibilidade da arquitetura I 71
6.2.3 Disponibilidade orientada à capacidade da arquitetura I 73
6.3 ESTUDO DE CASO III 73
6.3.1 Disponibilidade da arquitetura II 74
6.3.2 Análise de sensibilidade da arquitetura II 74
6.3.3 Disponibilidade orientada à capacidade da arquitetura II 75
6.4 ESTUDO DE CASO IV 76
6.4.1 Disponibilidade da arquitetura III 76
6.4.2 Disponibilidade orientada à capacidade da arquitetura III 78
6.5 CONSIDERAÇÕES 79

7 CONCLUSÃO 81
7.1 CONTRIBUIÇÕES 82
7.2 TRABALHOS FUTUROS 82

REFERÊNCIAS 84

8 APÊNDICE 90
8.1 MERCURY SCRIPT 90
8.2 CÓDIGO MODCS INJECTOR 110

1

Introdução

Conteúdos de mídia digital já fazem parte do nosso dia a dia, seja com o propósito de entretenimento - como conteúdos produzidos pela *Netflix* ou por usuários do *Youtube* -, informativos - documentários -, e até educativos - como no caso de videoaulas. O fato é que cada vez mais pessoas e instituições dependem de conteúdos de mídia digital. Qualquer um pode produzir e distribuir vídeos na Internet, o que fez com que, em 2017, 75% de todo o tráfego IP da internet tenha sido de transferência de arquivos de vídeo, sendo que a estimativa para 2022 é que este número suba para 82% (CISCO, 2018).

Apesar de termos um excelente poder computacional, seu crescimento é paralelo ao crescimento da nossa necessidade por processamento (CHATZOPOULOS et al., 2016). Uma das áreas que requer maior capacidade de processamento é a compressão de vídeo de alta qualidade (ZHAO et al., 2006). Para o processamento de imagens, um computador pessoal ainda é considerado lento, já que é incapaz de realizar, em tempo real, a transcodificação de um arquivo de vídeo de alta resolução. Uma estratégia que costuma ser adotada para acelerar a transcodificação, consiste em dividir o vídeo em pedaços e delegar a compressão de cada pedaço a um computador conectado em uma rede local (CHATZOPOULOS et al., 2016; PEREIRA et al., 2010).

Existem empresas que trabalham no segmento de transmissão de vídeo ao vivo pela internet, tais como *Youtube live*, *Facebook live*, Globo.com, etc. Em algumas destas empresas, o *time-to-market* é vital, como por exemplo, agências de notícias, retransmissores em tempo real de partidas de futebol ou transmissões feitas ao vivo através das redes sociais.

É necessária uma grande largura de banda - também muito custosa - para que os vídeos sejam transmitidos com alta qualidade pela internet, em seu tamanho original. Por requererem, tanto de seus transmissores quanto de seus receptores, uma internet com alta velocidade, deve-se fazer a compressão dos vídeos para posterior transmissão, diminuindo o tamanho do arquivo a ser transferido, assim como diminuindo a necessidade de velocidade de Internet. Esta abordagem pode trazer alguns problemas, como por exemplo, quando uma das máquinas responsáveis pela compressão falha, ou quando há um corte de energia, fazendo com que a compressão em tempo real seja inviável. Algumas técnicas têm sido desenvolvidas, uma destas no trabalho de (PEREIRA et al., 2010), onde os autores descrevem uma solução utilizada pela empresa Globo.com. Esta técnica faz uso de máquinas virtuais dinamicamente alocadas na nuvem Amazon, com o objetivo de alcançar a compressão de vídeo em tempo real, por meio da distribuição da carga de compressão entre estas máquinas.

De acordo com o *NIST* (*National Institute of Standards and Technology*) (MELL; GRANCE et al., 2011), a computação em nuvem é um modelo computacional onde é possível requerer a alocação de mais ou menos recursos computacionais (máquinas virtuais, armazenamento, aplicações), e que estes sejam obtidos de forma célere, e ainda, com o mínimo de esforço ou interação com o provedor de serviços. Os principais modelos de serviços oferecidos pelos provedores de nuvem são o *Software as a Service* (SaaS), *Platform as a Service* (PaaS) e *Infrastructure as a Service* (IaaS) (MELL; GRANCE et al., 2011). Além destes modelos, surgiram outros, entre eles, *Communication as a Service* (CaaS), *Database as a Service* (DBaaS), *Desktop as a Service* (DaaS) e por último, *X as a Service* (XaaS) indicando a infinidade de modelos que podem ser implementados com uma infraestrutura de nuvem (SCHAFFER, 2009).

As nuvens podem ser classificadas em quatro tipos: 1. a nuvem privada é aquela cuja infraestrutura é de uso exclusivo de uma entidade privada; 2. a nuvem comunitária é construída para o uso de uma comunidade que divide preocupações semelhantes; 3. a nuvem pública, na qual a infraestrutura é provida por uma empresa e qualquer pessoa pode contratar seus serviços; 4. a nuvem híbrida, que consiste na mistura de duas ou mais infraestruturas de nuvens mencionadas distintamente. (MELL; GRANCE et al., 2011).

Existem algumas soluções de nuvem *open source* disponíveis para que pessoas possam implementar sua própria infraestrutura. Uma destas opções é o *OpenStack* (SEFRAOUI; AISSAOUI; ELEULDJ, 2012b), que consiste em ser uma plataforma de nuvem popular, baseada em *Linux*.

A técnica de compressão de vídeos de alta resolução em tempo real mencionada, funciona através da utilização de um *software* que tem como característica sua organização em nós mestre e escravos. O nó mestre é responsável por segmentar o arquivo de vídeo e delegar seus pedaços aos nós escravos. Estes, são responsáveis por realizar a compressão e retornar o resultado desta ao nó mestre. Por fim, o nó mestre recebe todos os

pedaços e junta-os, obtendo o vídeo de alta qualidade comprimido em um único arquivo. Este paradigma de mapeamento e redução, onde o sistema automaticamente paraleliza o processo computacional entre clusters de máquinas chama-se MapReduce (DEAN; GHEMAWAT, 2008).

O trabalho apresentado por (GAO; WEN, 2016) propõe a implementação de um software para realização da compressão de vídeo distribuída. A proposta se utiliza da escalabilidade da nuvem pública Amazon para realização da compressão de vídeo distribuída. Além do custo crescente para tal distribuição, relativo ao aluguel das máquinas virtuais, eleva-se ainda o custo com banda do interessado, já que um arquivo de vídeo de alta resolução sem compressão ultrapassa 20Mb por segundo.

A indisponibilidade é um possível estado do provimento de serviço de compressão dos vídeos, trazendo cortes à transmissão e possível inviabilidade técnica para a continuação do processo, o que pode ocorrer por diversos motivos, tais como falhas de hardware ou de rede. Para se obter uma alta disponibilidade, se faz necessária a análise da infraestrutura, proposição de melhorias, como por exemplo mecanismos de tolerância a falhas e testes para homologação das melhorias.

Esta dissertação propõe modelos para analisar a disponibilidade de infraestruturas de nuvem privada *OpenStack* (SEFRAOUI; AISSAOUI; ELEULDJ, 2012b), utilizadas para compressão de video.

O estudo faz uma investigação de três arquiteturas propostas, considerando um, dois e três nós. Para representar as arquiteturas, modelos de diagramas de bloco de confiabilidade (*reliability block diagram* (RBD)) (TRIVEDI et al., 1996), e redes de Petri estocásticas (*Stochastic Petri Nets* (SPN)) (MOLLOY, 1981), (MARSAN; CONTE; BALBO, 1984) são utilizados. Utilizando os modelos, é possível calcular a disponibilidade das arquiteturas.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Como pode ser visto no relatório feito pela Cisco, em 2017, o tráfego IP de vídeos na Internet correspondeu a 75% de toda a banda. A projeção para 2022 é que este consumo corresponda a 82% do total(CISCO, 2018), enquanto que deste percentual, os vídeos ao vivo correspondam a 13% (CISCO, 2019).

Neste contexto, é factível e contributiva a criação de ferramentas que auxiliem no aumento da disponibilidade dos serviços de transmissão de vídeo ao vivo. Isto é possível através da sugestão de uma arquitetura *baseline*, e da criação de modelos matemáticos para o auxilio do cálculo da disponibilidade dos componentes da arquitetura, e a proposição de novas arquiteturas e seus modelos, os quais são usados para o estudo da implantação de dispositivos, como redundância ou tolerância à falhas, a fim de que haja um aumento na disponibilidade.

O resultado desta pesquisa habilita o profissional responsável pela arquitetura a identificação de pontos mais relevantes em termos de disponibilidade, bem como produz modelos

matemáticos para prova estatística.

1.2 OBJETIVOS

O objetivo do trabalho é construir modelos estocásticos, que auxiliem o planejamento de infraestruturas de transcodificação de vídeo distribuída com alta disponibilidade, utilizando uma nuvem privada *OpenStack*.

A metodologia da dissertação envolve etapas para a construção de um modelo hierárquico, sua validação e análise de sensibilidade para identificação da influência dos componentes na disponibilidade, bem como a proposição de novos modelos com mecanismos de tolerância a falhas, de forma que a disponibilidade seja aumentada.

De forma mais direta, os objetivos do trabalho são:

- Criar modelos de disponibilidade da infraestrutura, utilizando formalismos *Reliability Block Diagram* e *Stochastic Petri Net* para obtenção de métricas de disponibilidade e aplicação de uma análise de sensibilidade;
- Desenvolver ferramenta de injeção de falha e reparo, para auxílio na validação dos modelos.

1.3 ORGANIZAÇÃO DA DISSERTAÇÃO

A dissertação está dividida em 8 capítulos, os quais serão brevemente destacados a seguir.

No Capítulo 2, serão discutidos os trabalhos relacionados. Esta dissertação aborda trabalhos referentes à avaliação de desempenho, disponibilidade, técnicas de modelagem e ambientes de experimentação.

O Capítulo 3 apresenta a fundamentação teórica do trabalho proposto. Uma visão geral sobre computação em nuvem *OpenStack*, apresentada juntamente com conceitos básicos sobre avaliação de disponibilidade, além dos principais métodos de avaliação. Também são abordadas duas técnicas de modelagem de sistemas, RBD e SPN. Análise de sensibilidade, aplicação *Morph* e uma visão geral a respeito de transcodificação de vídeo também são outros tópicos abordados pelo capítulo.

O Capítulo 4 fala a respeito da metodologia utilizada para o desenvolvimento dos modelos do sistema em questão, além da apresentação de possíveis arquiteturas.

O Capítulo 5 descreve a construção dos modelos e suas etapas, partindo da coleta de tempos de falha e reparo na literatura, montagem de modelos RBD hierárquicos e a construção de um modelo final em SPN com um nó, e uma máquina virtual com o intuito de se extrair a disponibilidade do sistema.

O Capítulo 6 trata a respeito da validação dos modelos através da injeção de falha e reparo na representação física da infraestrutura modelada. Um programa foi criado para

efetuar as injeções e monitorar as entidades descritas no modelo. Também se demonstra o resultado da variação do modelo de *Petri Nets* (PN) com a adição de mais nós.

As conclusões e os trabalhos futuros são apresentados no Capítulo 7, enquanto no Capítulo 8, estão os apêndices.

2

Trabalhos Relacionados

Este capítulo apresenta alguns dos trabalhos na área de compressão de vídeo distribuída, análise de disponibilidade, *Capacity-Oriented Availability* (*COA*) (HEIMANN; MITTAL; TRIVEDI, 1990) e custo. Para isto, utiliza modelos baseados em *Reliability Block Diagram*, *Continuous Time Markov Chain* (KLEINROCK, 1975) (BOLCH et al., 2006), *MRM* (CLOTH et al., 2005) e SPN. A maior parte dos trabalhos examinados aqui dizem respeito à nuvem privada *Eucalyptus* (NURMI et al., 2009). No entanto, nós realizamos a análise de uma arquitetura de compressão de vídeo baseada em nuvem privada *OpenStack* (SEFRAOUI; AISSAOUI; ELEULDJ, 2012b), o que distingue este trabalho dos demais, no âmbito da pesquisa científica.

No trabalho de Dantas et al. (2016), os autores propõem modelos analíticos e análise de sensibilidade para um serviço de *Video on Demand* (VoD), em uma nuvem privada *Eucalyptus*. A análise de sensibilidade conduzida tem por foco máquinas virtuais e os componentes de VoD *streaming*, visando identificar gargalos para propor melhorias na infraestrutura, através de uma combinação hierárquica de dois tipos de modelos: RBD e *continuos-time Markov chains* (CTMC). Apesar de similaridades com esta dissertação, existem duas diferenças básicas. A primeira é que o modelo aqui proposto é relativo à disponibilidade de uma infraestrutura de compressão distribuída de vídeo. A está no uso da nuvem privada *OpenStack*. O trabalho tem modelos que são utilizados para gerar outros modelos. Como exemplo de novos modelos, pode-se mencionar o estudo de Dantas et al. (2016), que utiliza nuvem privada *Eucalyptus*, enquanto nossa arquitetura se utiliza de nuvem privada *OpenStack*.

Dantas et *al.* demonstram, por meio de análise de sensibilidade, que a variação de falha e reparo do *Apache Web Server* (Apache Web Server) (Apache Software Foundation, 1995) causa o maior impacto na disponibilidade do serviço. É feita ainda uma análise de sensibilidade semelhante, com o intuito de identificar os pontos de maior impacto de nossa infraestrutura, no que diz respeito à disponibilidade.

Um destes pontos de impacto pode ser percebido no trabalho de Gao et al. (2016), que compreende um sistema de compressão de vídeo paralelizada (*Morph*). Por utilizar os recursos de uma nuvem, ele é capaz de aprovisionar automaticamente (*autoscaling*) mais máquinas virtuais a fim de obter mais recursos para suprir diferentes cargas de trabalho. A arquitetura do *Morph* (GAO; WEN, 2016) conta com um nó mestre, responsável por gerenciar todo o trabalho de compressão, sendo responsável por prover a interface pela qual o usuário ou o sistema interagem, bem como gerenciar filas, sendo encarregado de selecionar em ordem os pedaços de vídeo a serem comprimidos. Além disto, também tem a capacidade de realizar *autoscaling*, onde pode alocar ou desalocar máquinas virtuais ou contêineres, com o objetivo de manter um tempo fixo de compressão de vídeo, independente do tamanho do arquivo. Nosso trabalho faz uma análise de sua implementação em uma nuvem privada *OpenStack*, desenvolvendo modelos de disponibilidade. Também propõe uma redundância em seu nó mestre, que representa um ponto de falha único.

Os autores Melo et al. (2016) preocupam-se em desenvolver modelos de disponibilidade referentes à infraestrutura de um servidor de sincronização baseado em *SyncML framework*. O motivo é assessorar partes interessadas em prover tal serviço em plataforma de nuvem. O intuito é dimensionar incrementos na arquitetura, bem como auxiliar previsões para elaboração de um *service level agreement* (*SLA*), prevenindo perdas financeiras. Ao longo deste trabalho também é construída uma representação do sistema, que foi usada como base para o desenvolvimento da representação dos servidores deste estudo. Tal representação foi modelada em RBD, com a intenção de se construir os modelos de disponibilidade. A escolha de tal diagrama, conforme exposta em Melo et al. (2016) se deve ao fato da ausência de prioridade entre os componentes da arquitetura: a falha de qualquer um dos componentes acarreta a indisponibilidade do serviço. Para a modelagem do serviço, neste trabalho de Melo et al. optou-se por usar uma CTMC, utilizando como estratégia de reparo de determinado serviço um *reboot* da máquina virtual. Nosso trabalho também faz uso de alguns dos valores de *mean time to failure* (MTTF) e *mean time to repair* (MTTR) obtidos por Melo et al. (2016).

Já o trabalho de Bezerra et al. (2015) propõe modelos de disponibilidade para um serviço de *streaming on demand* em uma nuvem privada *Eucalyptus*. Este serviço é baseado em uma máquina virtual com o sistema operacional *linux*. Existem dois componentes nesta máquina virtual que proveem o serviço: Apache Web Server, responsável pelo serviço *web* e o *software VLC*, responsável pelo serviço de *streaming* dos arquivos de vídeo. Esta customização da máquina virtual foi feita para que o reparo de possíveis falhas pudesse

ser automatizado. Através de uma análise de sensibilidade, os pontos críticos e possíveis gargalos referentes à disponibilidade são identificados, e uma redundância de componentes da infraestrutura é proposta, com a intenção de aumentar a disponibilidade do sistema como um todo. É feita aqui uma comparação entre os resultados da infraestrutura com e sem redundância, demonstrando uma diminuição de mais de 50% no downtime anual. O trabalho também faz uso de um modelo de alto nível RBD. Alguns dos blocos deste modelo representam outro modelo RBD ou uma CTMC do bloco de serviços, modelado desta forma para representar comportamentos que não podem ser totalmente representados por meio de modelos RBD. A disponibilidade da arquitetura redundante proposta é calculada através de uma equação fechada, que é obtida através do modelo CTMC, e utilizada em outra equação, que mensura a disponibilidade do sistema como um todo.

Os autores Melo et al. (2017) propõem modelos RBD e SPN para a avaliação da disponibilidade orientada à capacidade de nós em uma nuvem privada *Eucalyptus*. Estes cálculos apontam quantas máquinas virtuais estarão disponíveis em um nó. Também descrevem a quantidade de recursos que serão perdidos devido às rotinas de falhas e reparos. No mesmo estudo, Melo et al. (2017) demonstram que os modelos RBD são utilizados para cálculo de métricas como disponibilidade estacionária e *downtime* anual. Já o modelo SPN é utilizado para calcular a disponibilidade orientada à capacidade. Esta dissertação utiliza algumas métricas do trabalho de Melo et al. (2017), assim como alguns de seus MTTF e MTTR de componentes de hardware e componentes da nuvem. Existem similaridades entre os modelos descritos em Melo et al. (2017) e este trabalho, sendo que a principal diferença está na plataforma que provê o serviço de nuvem. Enquanto o *Eucalyptus* é avaliado no trabalho de Melo et al. (2017), neste optou-se pela utilização da plataforma *OpenStack*, em razão de uma maior disponibilidade documental para ser consultada, além de maior controle da nuvem por meio de APIS escritas em *python*.

Por sua vez, trabalho de Pereira et al. (2010) discorre sobre a implementação de uma arquitetura *Split&Merge* (LIAO; LI, 1997), que oferece a funcionalidade de dividir determinado arquivo de vídeo em partes menores, e delegar, de forma distribuída, a tarefa de compressão de cada arquivo, em diferentes máquinas virtuais. Graças a este artifício, juntamente com a elasticidade promovida pela nuvem pública Amazon, é possível ter um tempo de compressão de vídeo fixo, independente do tamanho do vídeo. Esta arquitetura proposta no trabalho de Pereira et al. (2010), faz uso de nuvem pública para a implementação do serviço elástico, diferindo desta dissertação neste ponto. Nosso trabalho foi realizado com a utilização de um *software* de compressão distribuído, baseado em uma nuvem privada. Esta arquitetura *Split&Merge* propõe resolver o problema do ponto único de falha, dividindo o controle entre 2 nós mestres, acarretando num aumento da disponibilidade.

Dantas et al. (2012) fazem análise de diversas arquiteturas possíveis para construção de uma nuvem *Eucalyptus* redundante, comparando a disponibilidade e custo de cada

uma delas. Entre estas arquiteturas, são avaliadas as disponibilidades para arquiteturas com 1, 2 ou 3 *clusters*, com uma estratégia de *warm-standby*. Modelos RBD são utilizados para analisar a dependabilidade das arquiteturas em razão do mecanismo de redundância. Foi adotado um modelo heterogêneo e hierárquico, composto por modelos RBD e *MRM* para representar estas arquiteturas. Enquanto o RBD descreve os componentes de mais alto nível, a *MRM* representa os componentes envolvidos no mecanismo de redundância e provê uma equação fechada para o cálculo da disponibilidade dos subsistemas redundantes. Os modelos são aplicados em 3 cenários diferentes, com variações no número de *clusters*, resultando em 9 arquiteturas.

No estudo de Melo et al. (2014), foram desenvolvidos modelos analíticos de disponibilidade para um serviço de *streaming* de VoD baseado em nuvem privada *Eucalyptus*. A estratégia de modelagem utilizou uma abordagem combinada entre RBD e CTMC. Uma análise de sensibilidade do modelo também foi proposta, para a identificação de possíveis gargalos, provendo um guia para implementação das melhorias do serviço. Tal arquitetura assemelha-se com uma versão mais simples da arquitetura utilizada nesta dissertação, considerando um servidor *FrontEnd* e um servidor nó, contendo os serviços.

A combinação dos modelos CTMC e RBD forma um modelo hierárquico, utilizado para realização da análise. A modelagem em CTMC permite a obtenção de uma equação fechada para análise da disponibilidade estacionária, sendo útil, também, na análise de sensibilidade e na obtenção das métricas desejadas, sem a necessidade de recorrer a uma solução numérica do modelo. Esta dissertação faz uso de alguns MTTF utilizados pelo trabalho de Melo et al. (2014), além de usar, também, um modelo hierárquico similar ao proposto, com SPN e RBD para a modelagem do hardware e dos serviços propostos. Ainda Melo et al. (2015), propõem um modelo de disponibilidade de um serviço VoD baseado em nuvem privada *Eucalyptus*, posto que são modelos que fazem uma avaliação de sensibilidade de seus componentes, para guiar melhorias de implementações nos sistemas. Neste caso, uma análise de sensibilidade paramétrica foi utilizada para a identificação de gargalos no serviço de VoD. A arquitetura deste sistema difere daquela proposta por Melo et al. (2014) por ter um nó a mais na parte do serviço, utilizado como estratégia de redundância com modo de *warm-standby*. O trabalho avalia 3 estudos de caso: o primeiro, com uma máquina dedicada ao *Front-end* e duas máquinas para os nós. Neste primeiro caso, a disponibilidade do sistema de *streaming* de vídeo sem redundância é de 0.994401, evidenciando a importância na busca para soluções efetivas na melhoria do sistema. Para o segundo caso, uma análise de sensibilidade paramétrica foi realizada e demonstra que a taxa de reparo do *Front-end* é o parâmetro mais importante relacionado à disponibilidade. Sua segunda taxa mais importante é a taxa de reparo do Front-end. Neste trabalho, constata-se o ponto crítico no tangente à disponibilidade, priorizando o componente nas melhorias consideradas. Já o terceiro caso faz uma análise de sensibilidade utilizando outra técnica, a de *"reliability importance"*, concluindo, também, que a parte

mais importante da arquitetura, no que se diz respeito a disponibilidade, é o *Front-end*.

Bezerra et al. (2014) propõem modelos de avaliação de um serviço de VoD numa nuvem privada *Eucalyptus*, com algumas diferenças do proposto em Melo et al. (2015). A arquitetura difere no fato de contar com um nó como *cloud manager* e um nó *controller*. São propostos modelos RBD para cálculo de disponibilidade desta arquitetura. O bloco de serviços do RBD é modelado por um modelo *Markov reward model* (MRM), que considera os comportamentos interdependentes dos componentes Apache Web Server, *VLC* e máquina virtual. Este modelo permite a extração de equações fechadas que auxiliam no cálculo da disponibilidade. Para a validação dos modelos propostos em Bezerra et al. (2014), utilizou-se uma técnica de injeção de falha, onde um *script shell* inseriu falhas periódicas no sistema e usou uma redução de fator de 1000 no MTTF. A análise de sensibilidade paramétrica, também considerada no trabalho, sugere que o nó *controller* é o componente mais crítico desta arquitetura.

O trabalho de Costa et al. (2016) apresenta uma análise de disponibilidade do *framework OpenMobster*. Utiliza modelos CTMC e RBD e realiza análise de sensibilidade paramétrica. Seu estudo de caso faz uso de injeção de falhas e reparos, que é realizada através de *shell scripts* e programas escritos em linguagem R.

Já o estudo de Souza et al. (2013a) refere-se ao estudo de caso da ferramenta de injeção de falhas e reparos Eucabomber, desenvolvida com o objetivo de auxiliar estudos de confiabilidade e disponibilidade na plataforma *Eucalyptus*. Esta ferramenta representa os tempos entre falhas através de uma distribuição exponencial escolhida pelo usuário. A ferramenta é específica para avaliar as métricas em nuvens *Eucalyptus*, e utiliza o protocolo *SSH2* para simulação de injeções de falha e reparo.

A forma como a injeção de falhas e reparos é conduzida no Eucabomber (SOUZA et al., 2013a) é a mesma utilizada no presente estudo: o *software* gera um número aleatório, dorme este período de tempo e injeta uma simulação de falha, fazendo com que o sistema fique indisponível. Novamente, o software gera um número aleatório, dorme este período de tempo e injeta uma simulação de reparo (um comando para ligar o servidor pela rede, por exemplo). O ciclo continua até que o software receba um comando de parada.

O trabalho de Brilhante et al. (2014) propõe uma extensão do *software* Eucabomber (SOUZA et al., 2013a), que também é utilizado para injeção de falhas e reparos nos componentes da nuvem *Eucalyptus*. O software faz uso de números aleatórios, exponencialmente distribuídos utilizando taxas, injeta a falha e/ou reparo, além de monitorar a disponibilidade dos serviços. A diferença entre esse trabalho e esta dissertação consiste no fato de que o estudo de Brilhante et al. (2014) é específico para nuvens *Eucalyptus*, enquanto nosso trabalho é agnóstico a sistemas, posto que utilizamos uma função *Python* para executar os comandos através do sistema operacional vigente.

Por sua vez, a pesquisa de Garcia et al. (2011) faz um estudo a respeito de uma infraestrutura de nuvem que utiliza o *Hadoop* (SHVACHKO et al., 2010), para transcodifi-

cação de vídeo, com o objetivo de disponibilizá-lo para *streaming*, utilizando tecnologia HLS (FECHEYR-LIPPENS, 2010). Nos experimentos descritos no trabalho de Garcia et al.(2011), o conteúdo de um DVD é processado em três contextos: o primeiro considera os capítulos do DVD como partes a serem processadas em paralelo; e o segundo, com o conteúdo do DVD dividido em segmentos de dez segundos; e o terceiro considera a divisão do conteúdo do DVD igual ao número de nós disponíveis na nuvem.

O trabalho discorre a respeito das vantagens do uso da computação em nuvem no processamento de vídeos. Aponta a vantagem da redução do espaço necessário para armazenamento, bem como habilita diferentes tipos de usuários a consumir o vídeo transcodificado, e finaliza salientando que o uso desta infraestrutura pode ir além do HLS.

O estudo de Kafhali et al. (2017) consiste numa análise de performance em *cloud data centers*. Para isto, propõe um modelo estocástico que utiliza o formalismo CTMC baseado em teoria de filas. Exemplos numéricos são apresentados para estimar o número de VMS necessárias para satisfazer determinado QoS. Utiliza um ambiente de simulação para realização da análise e validação de um modelo analítico, que pode estimar com precisão o número de máquinas virtuais necessárias para satisfazer determinada métrica de QoS.

Por sua vez, a pesquisa de Karolewicz et al. (2017) propõe a redução dos custos de provedores de VoD que oferecem *adaptive video streaming*. A proposta apresenta, por meio de métodos numéricos, um comparativo entre o custo de se hospedar todas as versões do vídeo, ou apenas o maior arquivo, e realizar a transcodificação de acordo com a demanda. O estudo faz uma análise de performance, e tem como objetivo manter a disponibilidade do sistema em um número igual ou superior a 99%. A validação dos métodos propostos é realizada através de simulações, e utiliza um simulador discreto de eventos na infraestrutura para analisar a disponibilidade do conteúdo.

Esse trabalho propõe métodos para provisionamento de um *adaptive video streaming service*, oferecido em um ambiente de nuvem. Estes métodos também auxiliam na decisão sobre um determinado vídeo ser armazenado ou transcodificado.

Já o estudo realizado por Martin et al. (2004) propõe a utilização de modelos CTMC e MRM, para obtenção de medidas de QoS de um serviço de VoD adaptativo. Estes modelos também são úteis na definição de *service level agreements*, já que são capazes de computar diversos parâmetros discutidos neste estudo de Martin et al. (2004). A validação dos modelos se dá através de um *testbed* experimental de um serviço de VoD.

Esses modelos fazem parte de uma nova metodologia apresentada para auxiliar no *design* e avaliação de serviços multimídia ponto a ponto. A metodologia facilita a obtenção de métricas de performance e análise por modelos markovianos, podendo ser aplicada para qualquer serviço multimídia que sejam possíveis de serem representáveis por modelos CTMC.

2.1 CONSIDERAÇÕES FINAIS

Este capítulo apresentou trabalhos relacionados ao tema presente disponíveis na literatura. São estudos que discorrem a respeito da análise de atributos do termo "dependabilidade", como desempenho, disponibilidade, etc. Análise de sensibilidade, modelos estocásticos e técnicas de validação através de injeções de falha e reparo também são apresentados diferentes pesquisas que embasaram este capítulo apresentados neste capítulo.

Neste trabalho são utilizadas diversas técnicas descritas nas obras apresentadas neste capítulo. As quais focam em serviços utilizando nuvem privada *Eucalyptus*, enquanto nosso trabalho utiliza *OpenStack* como plataforma de nuvem privada. Por sua vez, a literatura relacionada a injeções de falha e reparo, discutida neste capítulo é específica para plataformas de nuvem *Eucalyptus*, enquanto o injetor de falha e reparo desenvolvido por este trabalho foi programado na intenção de suportar a plataforma de nuvem *OpenStack*.

A Tabela 1 sumariza todas as obras abordadas por esta dissertação, assim como pelos trabalhos mencionados neste capítulo. Dentre a lista de atributos estudados pelos autores de outros trabalhos, essa dissertação não faz análise de custo, nem utiliza modelagem *MRM*.

Tabela 1 – Comparativo entre trabalhos

Trabalho	Contexto	Análise de sensibilidade	Disponibilidade	COA	Custo	RBD	CTMC	SPN	MRM	Injeção de Falhas	Cloud Computing	Transcodificação distribuída
Este trabalho	Compressão de vídeo distribuída	✓	✓	✓		✓		✓		✓	✓	✓
(DANTAS et al., 2016)	Serviço de streaming on demand	✓	✓			✓	✓				✓	
(MELO et al., 2016)	OwnCloud	✓	✓			✓	✓				✓	
(BEZERRA et al., 2015)	Streaming on demand em cloud	✓	✓			✓	✓				✓	
(MELO et al., 2017)	Estimativa de recursos em nuvem privada	✓	✓	✓		✓		✓			✓	
(DANTAS et al., 2012)	Nuvem privada Eucalyptus		✓		✓	✓			✓		✓	
(MELO et al., 2014)	Streaming on demand com Eucalyptus	✓	✓			✓	✓				✓	
(BEZERRA et al., 2014)	Streaming on demand com Eucalyptus	✓	✓			✓			✓	✓	✓	
(MELO et al., 2015)	Streaming on demand com nuvens privadas	✓	✓			✓	✓				✓	✓
(SOUZA et al., 2013a)	Software de injeção de falhas e reparos Eucabomber	✓	✓			✓				✓		
(BRILHANTE et al., 2014)	Software de injeção de falhas e reparos Eucabomber 2.0		✓				✓		✓	✓		
(COSTA et al., 2016)	Mobile Backend-as-a-service platform	✓	✓			✓	✓			✓		
(GARCIA; KALVA, 2011)	Cloud de transcodificação de vídeo		✓								✓	✓
(GAO; WEN, 2016)	Software para transcodificação de vídeo distribuída										✓	✓
(PEREIRA et al., 2010)	Implementação de software de compressão em nuvem pública										✓	✓
(KAFHALI; SALAH, 2017)	Estudo de performance em CDC		✓				✓				✓	
(KAROLEWICZ; BĘBEN, 2017)	Estudo de redução de custo em provedores *VoD*		✓		✓						✓	✓
(MARTIN et al., 2004)	Modelagem de serviço de *streaming* adaptável						✓		✓			

3

Referencial Teórico

Este capítulo introduz o leitor à plataforma de nuvem de código aberto *OpenStack*, e seus principais componentes, na perspectiva desta pesquisa. Entre os componentes, discorremos a respeito do componente *Neutron*, responsável pela gerência das redes utilizadas pela plataforma, e do componente *Nova*, responsável pelo aprovisionamento de instâncias computacionais. Conceitos de dependabilidade são apresentados, e em seguida, há a introdução a respeito de modelagem hierárquica para obtenção de métricas de dependabilidade. As técnicas de modelagem utilizadas por nosso trabalho são *Reliability Block Diagram* e *Stochastic Petri Net*.

Ainda neste capítulo, apresentamos um panorama sobre análise de sensibilidade e ainda, sobre a aplicação Morph, responsável por realizar a compressão de vídeos distribuída, e que é levado em consideração por este trabalho, na construção da infraestrutura escolhida.

Técnicas de injeção de falha e reparo também são abordadas, bem como a análise de alguns softwares que automatizam esta tarefa. Por fim, apresentamos conceitos relativos à compressão de vídeos, também importantes para esta pesquisa, que busca auxiliar a construção de uma infraestrutura de compressão de vídeo distribuída.

3.1 COMPUTAÇÃO EM NUVEM E OPENSTACK

Existem diversas soluções de nuvem privada de código aberto, como por exemplo *OpenStack*, *OpenNebula* e *Eucalyptus* (SEFRAOUI; AISSAOUI; ELEULDJ, 2012b). Este tra-

balho optou pela nuvem *OpenStack* por ser inteiramente *opensource*, desde sua concepção, e por dispor de um maior suporte a novos padrões (SEFRAOUI; AISSAOUI; ELEULDJ, 2012a).

3.1.1 Neutron

O controlador de rede *Neutron* gerencia todas as questões relacionadas a rede para a infraestrutura virtual de rede e aspectos da infraestrutura física de rede no ambiente *OpenStack*. Graças a este controlador, é possível habilitar que projetos criem topologias de redes virtuais avançadas, que podem incluir um balanceador de carga ou uma *VPN*, por exemplo (COMMUNITY, 2019).

3.1.2 Nova

Nova é a parte do OpenStack responsável por prover instâncias de computação (servidores virtuais). Dá suporte à criação de máquinas, servidores *bare metal* e suporte limitado a containers, ou seja, é o controlador responsável por gerenciar as instâncias computacionais criadas pelos usuários da nuvem privada.

3.2 DEPENDABILIDADE

O termo "dependabilidade"(*dependability*), criado por Laprie (AVIZIENIS et al., 2004), engloba alguns conceitos como confiabilidade, disponibilidade, segurança, confidencialidade, manutenabilidade, segurança, integridade, etc. A "dependabilidade"de um sistema é a sua capacidade de entregar, de forma confiável, um conjunto de serviços que são observados por agentes externos. Um serviço é confiável quando implementa a funcionalidade exata especificada pelo sistema. Uma falha de sistema ocorre quando este sistema falha em prover sua respectiva funcionalidade (AVIZIENIS et al., 2004).

Um dos atributos da "dependabilidade"é a disponibilidade. Ela é relacionada ao tempo em que o sistema está disponível para o usuário, ou seja, quando inexiste qualquer erro no sistema que impossibilite prover determinado serviço (falha). Para o cálculo da disponibilidade, usamos a Equação 3.1, ou pela Equação 3.2 (MACIEL et al., 2012), onde A é a disponibilidade, *UPTIME* é o tempo em que o sistema esteve disponível e *DOWNTIME* é o tempo em que o sistema esteve indisponível.

Para o cálculo da confiabilidade de um sistema com N elementos em série, onde os seus tempos de falha são exponencialmente distribuídos usamos a Equação 3.3, onde R é a confiabilidade (*reliability*) e R_t é a confiabilidade no tempo t, isto se as taxas de falha e reparo forem exponencialmente distribuídas ($e^{-\lambda_{i \times t}}$).

$$A = \frac{E\left\lfloor uptime \right\rfloor}{E\left[uptime\right] + E\left\lfloor downtime \right\rfloor} \tag{3.1}$$

ou

$$A = \frac{MTTF}{MTTF + MTTR} \tag{3.2}$$

$$R_i = \prod_{i=1}^{n} R_i(t) \tag{3.3}$$

se

$$R_i(t) = e^{-\lambda_{i \times t}}$$

$$\lambda_{eq} = \sum_{n=i}^{n} \lambda_i$$

$$MTTF_s = \frac{1}{\sum_{n=i}^{n} \lambda_i}$$

onde

$$MTTF = \int_0^\infty R(t)dt = \int_0^\infty exp^{(-\lambda)t} = \frac{1}{\lambda}$$

Através dos meios para o cálculo de disponibilidade, também é possível calcular a disponibilidade orientada à capacidade (COA), que é uma métrica relativa a capacidade de provimento computacional oferecida pelo sistema, ou seja, considera não apenas a disponibilidade ou indisponibilidade, mas a quantidade de serviço ofertado pelo sistema (HEIMANN; MITTAL; TRIVEDI, 1990). O cálculo da COA considera o número máximo de recursos disponíveis em qualquer estado S_i (no caso deste trabalho, o número de máquinas virtuais), ou NMVMS, a taxa de recompensa, equivalente ao número de recursos disponíveis em determinado estado S_i, ou seja, π_i, o conjunto de todos os estados disponíveis, ED e a capacidade máxima, CM, como mostra a Equação 3.4:

$$\frac{\sum_{Si \in ED}^{NMVMS} pc_i \times \pi_i}{CM} \tag{3.4}$$

Através do cálculo, conseguimos determinar a real quantidade de recursos que são entregues ao usuário.

3.2.1 Reliability Block Diagram

RBD, ou *reliability block diagram*, é um tipo de formalismo que representa determinado sistema, através de um modelo que consiste em blocos conectados serialmente ou em paralelo. Cada bloco representa um componente do sistema, indicando suas respectivas taxas de falha e reparo. Os caminhos paralelos indicam redundância no sistema, onde todo bloco de uma determinada linha paralela precisa falhar para que o sistema falhe, enquanto na representação serial, basta que um dos componentes falhem, para que o sistema inteiro falhe (MACIEL et al., 2012). Existe um tipo de estrutura representada em RBD como blocos K-out-of-N (K de N), nos quais, por exemplo, um sistema com quatro blocos idênticos e sem dependência entre eles (a, b, c, d), pode ser operacional se pelo menos 3 dos 4 componentes estão funcionando corretamente. Além disto, também podem

ter seus blocos agrupados, reduzindo o número de blocos e considerando-os como um único bloco (MACIEL et al., 2012). A Equação 3.5 refere-se ao cálculo da disponibilidade (A) para um modelo RBD com os blocos em série (As), enquanto a Equação 3.6, refere-se ao cálculo da disponibilidade para um modelo RBD com os blocos em paralelo. Já a Equação 3.7, refere-se ao cálculo da disponibilidade de um modelo RBD com blocos *K-out-of-n*. O cálculo expressado na Equação 3.7 só é válido quando o tempo para falha é exponencialmente distribuído.

$$As = \prod_{i=1}^{n} A_i \tag{3.5}$$

$$Ap = 1 - \prod_{i=1}^{n} (1 - A_i) \tag{3.6}$$

$$MTTF_{KooN} = \frac{1}{\lambda} \sum_{i=k}^{n} \frac{1}{i} \tag{3.7}$$

3.2.2 Redes de Petri

Redes de Petri, ou rede de lugar e transição, é um dos muitos modelos matemáticos de modelagem utilizados para descrever sistemas. Modela eventos discretos, sendo representado por um gráfico com lugares e transições. Os eventos são representados pelas transições e os lugares representam as condições. Os lugares e transições são conectados por arcos. A estrutura de uma rede de Petri em matrizes é a quíntupla R = (P,T,I,O,M_0), onde P é um conjunto de lugares finitos, T é um conjunto finito de transições, I : P $\times$ T $\rightarrow \mathbb{N}$ é a matriz das pré-condições, O : P $\times$ T $\rightarrow \mathbb{N}$ é a matriz de pós-condição e M_0 é a marcação inicial. Além destes, redes de Petri também possuem outro atributo que é a marcação (*token*). Os *tokens* são informações dos lugares. A forma como os *tokens* são distribuídos nos lugares representam o estado do sistema em um determinado momento (MURATA, 1989). Um exemplo da rede de Petri é apresentado na Figura 1, que descreve uma infraestrutura com servidores. Existem dois estados, o primeiro (SERVIDORES UP) que descreve a quantidade de servidores que estão em funcionamento. O segundo (SERVIDORES DOWN), descreve a quantidade de servidores que não estão em funcionamento. As duas transições apresentadas no trabalho (FALHA e REPARO), representam as possibilidades de ocorrerem falhas e reparos na infraestrutura.

3.2.3 Redes de Petri Estocásticas

O avanço nos estudos de redes de Petri trouxe novas extensões, derivadas do modelo original. Redes de Petri temporizadas, estocásticas, alto nível e coloridas, são algumas das variações existentes da técnica de modelagem. Estas mudanças ocorreram pela necessidade de adaptabilidade, bem como por causa de diferentes características e demandas não possíveis de serem representadas pelas redes de Petri.

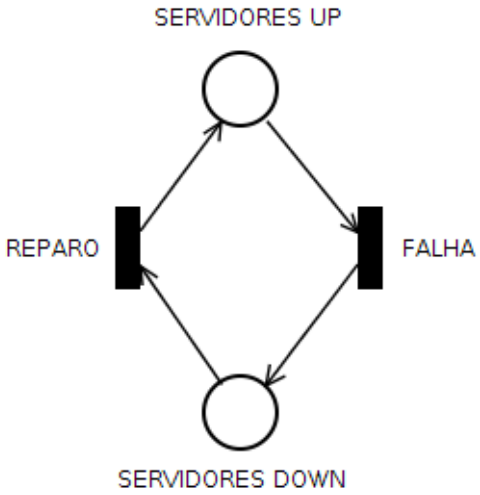

Figura 1 – Rede de Petri

Uma das extensões do estudo de redes de Petri, a rede de Petri estocástica (GERMAN, 2000), acrescenta a possibilidade de representar transições com tempo exponencialmente distribuído, permitindo uma análise probabilística dos sistemas. Além disto, acrescenta a representação de transições instantâneas, ou transições imediatas, que têm tempo de disparo de zero. Diferentes níveis de prioridade podem ser definidos para transições imediatas concorrentes, resolvendo uma situação de confusão (MARSAN; CONTE; BALBO, 1984).

A necessidade de representar transições temporizadas e imediatas em um mesmo modelo trouxe outra extensão das redes de Petri estocásticas, a *Generalized Stochastic Petri Nets* (GSPN). Esta extensão permite associações entre transições temporizadas e imediatas (MARSAN; CONTE; BALBO, 1984). O acrônimo SPN tem sido utilizado para se referir às extensões derivadas de SPN, como as GSPN (GERMAN, 2000).

Uma SPN pode ser representada por uma 9-tupla SPN = {P,T,I,O,H,$\prod$,G,M_0,Atts}:

P = {p1, p2, ..., pn} representa o conjunto dos lugares;

T = {t1, t2, ..., tn} representa o conjunto das transições;

I $\in (\mathbb{N}^n \rightarrow \mathbb{N})^{n \times m}$ representa a matriz dos arcos de entrada (podem ser dependentes de marcação);

O $\in (\mathbb{N}^n \rightarrow \mathbf{N})^{n \times m}$ representa a matriz dos arcos de saída (podem ser dependentes de marcação);

H $\in (\mathbb{N}^n \rightarrow \mathbf{N})^{n \times m}$ representa a matriz dos arcos inibidores (podem ser dependentes de marcação);

$\prod \in \mathbb{N}^m$ representa o vetor de associação entre os níveis de prioridade e as transições;

G $\in (\mathbb{N}^n \rightarrow \{\boldsymbol{true,false}\})^m$ representa o vetor de associação entre as expressões de guarda relacionados às marcações de lugares a cada transição;

$\mathbf{M}_0 \in \mathbb{N}^n$ representa o vetor de marcação inicial (estado inicial);

Atts = *(Dist, Markdep, Policy, Concurrency, W)*^{m} representa o conjunto de atribuições das transições onde:

Dist $\in (N)^m \rightarrow \mathcal{F}$ é a função de distribuição de probabilidade associada ao tempo de disparo de uma transição (o domínio de F é [0,∞]), e pode ser dependente de marcação;

Markdep $\in$ {constant, enabdep}, onde a distribuição de probabilidade referente ao tempo de uma transição pode ser independente (constant) ou dependente de marcação (enabdep), onde a distribuição depende do atual grau de habilitação;

Policy $\in$ {prd, prs}, corresponde à política de memória utilizada pela transição. Prd, ou *preemptive repeat different*, corresponde à política *reset memory* ou *race enabling policy*, enquanto prs, ou *preemptive resume*, que possui significado correspondente à *age memory policy*;

Concurrency $\in$ {ss, is} representa a semântica de concorrência das transições, onde ss representa *single server*, enquanto is representa *infinite server*.

W : T $\rightarrow IR\cup$ é a função que representa o peso das transições imediatas (W_t) e a taxa das transições temporizadas (λ_t), onde:

$$\pi(t) = \begin{cases} \geq 1, & \text{para transições imediatas} \\ 0, & \text{caso contrário.} \end{cases} \tag{3.8}$$

O estado de um sistema modelado é representado pela quantidade de tokens nos lugares. As transições modelam as ações do sistema e seu disparo altera o estado atual do sistema. O tempo associado às transições é descrito como exponencialmente distribuído (MARSAN; CONTE; BALBO, 1984).

Arcos inibidores, representados pela Figura 2a fazem parte desta extensão de redes de Petri. São utilizados para inibir um disparo de transição quando determinado lugar não tem tokens.

As transições temporizadas estão representadas pela Figura 2b, e as transições imediatas, Figura 2c.

A Figura 3 apresentada refere-se à uma modelagem através de redes de Petri. Esta modelagem representa um servidor físico funcionando com algumas máquinas virtuais. O lugar VMS UP representa o número de máquinas virtuais em funcionamento em determinado instante, enquanto o lugar VMS down representa o número de máquinas virtuais que não estão em funcionamento. A transição temporizada FALHA VM representa a falha de

(a) Arco Inibidor

(b) Transição temporizada

(c) Transição imediata

Figura 2 – Componentes das redes de Petri

uma máquina virtual que a torna indisponível, transferindo um token para o lugar VMS DOWN. A transição temporizada REPARO VM, representa o reparo de uma máquina virtual que a torna disponível, transferindo um token para o lugar VMS UP.

O lugar SERVIDORES UP representa o número de servidores ligados, enquanto o lugar SERVIDORES DOWN, representa os servidores indisponíveis. A transição temporizada FALHA SERVIDOR representa a falha de um servidor físico, sendo que seu disparo faz com que um token seja transferido para o lugar SERVIDORES DOWN. A transição temporizada REPARO SERVIDOR representa o reparo de um servidor físico, sendo que seu disparo faz com que um token seja transferido ao lugar SERVIDORES UP.

A transição imediata SERVIDOR DOWN FALHA VM tem um arco inibidor atrelado ao lugar SERVIDORES UP, indicando que esta transição só pode ser disparada quando não existirem tokens no lugar SERVIDORES UP, o que significaria que não existem máquinas físicas operando. Sem servidores em funcionamento, a transição imediata é disparada, transferindo todos os tokens de VMS UP para VMS DOWN, já que as máquinas virtuais não podem existir sem um servidor físico.

Outro arco inibidor é observado entre o lugar SERVIDORES DOWN e a transição REPARO VM, indicando que só pode haver o reparo das máquinas virtuais quando há um token no lugar SERVIDORES UP, que no caso do modelo apresentado também significa que o número de tokens em SERVIDORES DOWN é zero.

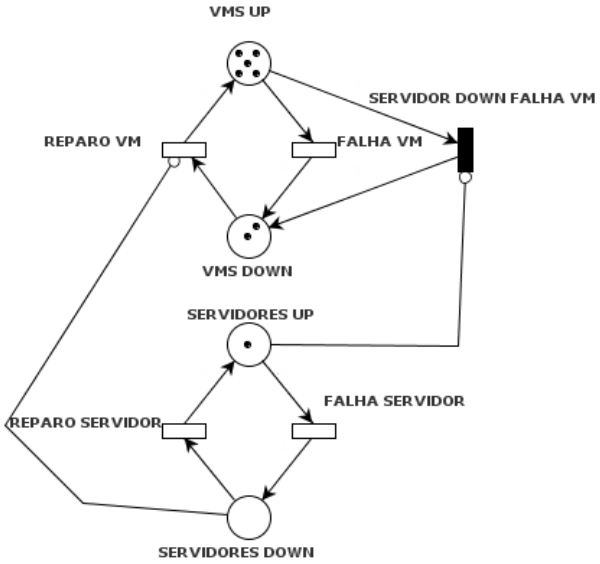

Figura 3 – Rede de Petri Estocástica

3.3 ANÁLISE DE SENSIBILIDADE

Análise de sensibilidade é composta por um conjunto de técnicas utilizadas para descobrir os parâmetros mais relevantes nas métricas de determinado modelo. Estas técnicas permitem a identificação de gargalos no sistema. Uma análise de sensibilidade pode ajudar a determinar (HAMBY, 1994):

- Quais parâmetros são insignificantes e podem ser eliminados do modelo final;
- Quais entradas têm mais influência em determinado resultado de um modelo;
- Quais parâmetros são mais correlacionados com o resultado.

Entre as várias técnicas que existem, estão a análise diferencial, *design* fatorial, derivação de sensibilidade, indíces de importância e etc.

A análise de sensibilidade paramétrica, ou índice de sensibilidade, como também é conhecida, é uma das técnicas simples, que funciona quando é calculada a diferença percentual da saída, ou quando existe variação de um dos parâmetros, com seu valor mínimo ao máximo (HAMBY, 1994), (HOFFMAN; MILLER, 1983). Os autores Hoffman; Miller (1983) defendem a utilização do alcance completo dos possíveis valores para a descoberta real dos parâmetros de sensibilidade. O índice de sensibilidade Sθ (x), que indica o impacto da medida x a respeito do parâmetro θ. A Equação 3.9 demonstra o cálculo de diferencial percentual para uma métrica X(θ), onde o valor máximo X (θ) e mínimo X (θ) são os valores máximo e mínimo, computados variando-se o parâmetro θ em um espectro de n possíveis valores de interesse.

$$S\theta(X) = \frac{max\{X(\theta)\} - min\{X(\theta)\}}{maxX(\theta)} \tag{3.9}$$

A Figura 4 representa um exemplo da análise de sensibilidade de um determinado parâmetro (MTTF do nó), sendo sua variância e o seu impacto na disponibilidade do sistema, representados pelo modelo. Percebe-se que, com um aumento no tempo médio de falha (MTTF) do componente analisado, ocorre um impacto direto na disponibilidade do sistema. Uma posterior análise correlacional com outros índices demonstra quais os componentes que mais afetam a disponibilidade, revelando o parâmetro de maior interesse na busca de uma maior disponibilidade do sistema.

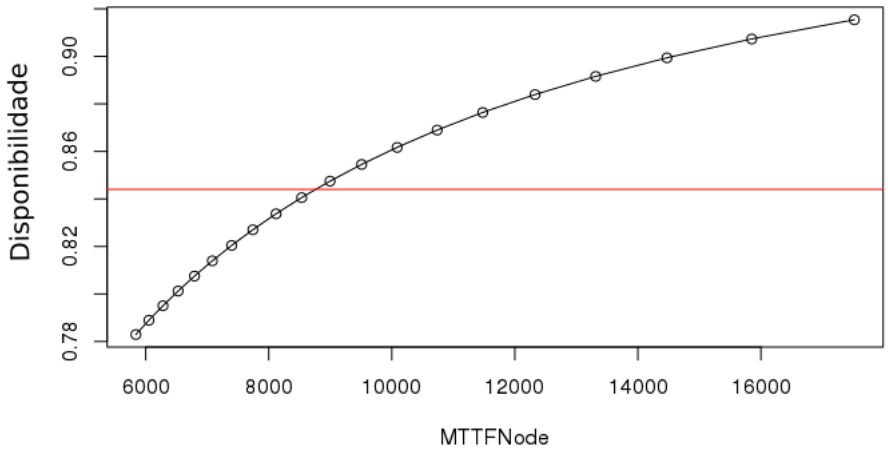

Figura 4 – Análise de sensibilidade

3.4 MÉTODO KEESEE PARA CÁLCULO DO INTERVALO DE CONFIANÇA

O método de Keesee (KEESEE, 1965) é utilizado por este trabalho para determinar o intervalo de confiança dos testes de injeção de falha e reparo. Este método tem como objetivo a determinação de um intervalo de confiança para disponibilidade, quando os tempos de MTTF e MTTR são estimados.

Por sua vez, quando a taxa de falha e reparo de um sistema são constantes independentes de tempo, fica demonstrado que as estimativas de MTTF e MTTR podem ser combinadas para que se obtenha a função de densidade de relação de variância. A disponibilidade estimada tem uma densidade que é a função da densidade de relação de variância. Esta relação pode ser utilizada para que se obtenha as equações para um intervalo de confiança da disponibilidade.

O método tem como primeiro passo a determinação da densidade de probabilidade de alguma função para estimar $\hat{A}$ (disponibilidade estimada). A derivação da densidade da função do MTTF assume que a taxa de falha de um sistema $\frac{1}{\hat{A}}$, que é uma constante independente de tempo, então a função de densidade de t_i :

$$f(t_i) = \frac{1}{\theta} exp(-\frac{t_i}{\theta})$$

por definição

$$\theta = \frac{1}{n} \sum_{i=1}^{n} t_i$$

onde n é o número de períodos de tempo.

A derivação de função de densidade de t_j é similar a de t_i, onde apenas substituímos o MTTF por MTTR e o N (número de falhas) por M (número de reparos).

Na derivação das funções de densidade, assumiu-se que a cada período de tempo T terminou em uma falha ou reparo. A soma das ocorrências de falha e reparo resultam no grau de liberdade da validação. Durante um período de testes, podemos fazer uso de um de dois casos para utilizar o grau de liberdade. O primeiro caso ocorre quando um teste é terminado por um número predeterminado de falhas, e com isso, determina-se que o número de graus de liberdade é igual a duas vezes o número de falhas. Para o segundo caso, onde o teste termina depois de um período de tempo predeterminado, o número de graus de liberdade é igual ao número de falhas mais 2.

O método continua obtendo os limites de confiança superiores e inferiores do intervalo de confiança onde, para o primeiro caso, temos as Equações 3.10 e 3.11:

$$(\frac{\phi}{\theta})_U = \frac{\widehat{\phi}}{\widehat{\theta}} F_{\frac{1-a}{2};2n,2m} \tag{3.10}$$

e

$$(\frac{\phi}{\theta})_L = \frac{\widehat{\phi}}{\widehat{\theta}} F_{\frac{a}{2};2n,2m} \tag{3.11}$$

ou, como no segundo caso, temos as Equações 3.12 e 3.13:

$$(\frac{\phi}{\theta})_U = \frac{n}{n-1} \frac{\widehat{\phi}}{\widehat{\theta}} F_{\frac{1-a}{2};2n,2m} \tag{3.12}$$

e

$$(\frac{\phi}{\theta})_L = \frac{n}{n-1} \frac{\widehat{\phi}}{\widehat{\theta}} F_{\frac{a}{2};2n,2m} \tag{3.13}$$

onde:

n = número de falhas no caso 1 (Equações 3.10 e 3.11);

n = número de falhas 1 no caso 2 (Equações 3.12 e 3.13);

m = número de reparos;

1-a = nível de confiança.

$F_{\frac{1-a}{2}}; 2n, 2m$= a parte superior da função de densidade F com 2n e 2m graus de liberdade.

e

$F_{\frac{\alpha}{2}}; 2n, 2m=$ a parte inferior da função de densidade F com 2n e 2m graus de liberdade

Se a disponibilidade é expressada como apresentado na Equação 3.14:

$$A = \frac{1}{1 + (\frac{\phi}{\theta})} \tag{3.14}$$

então os limites inferiores e superiores do intervalo de confiança de Â pode ser obtido das relações apresentadas nas Equações 3.15 e 3.16:

$$A_U = \frac{1}{1 + (\frac{\phi}{\theta})_L} \tag{3.15}$$

e

$$A_L = \frac{1}{1 + (\frac{\phi}{\theta})_U} \tag{3.16}$$

3.5 MORPH

Sistemas de distribuição de compressão de vídeo permitem que haja uma transcodificação de vídeo eficiente, já que diminui de forma gradual o tempo necessário para a finalização da transcodificação (PEREIRA et al., 2010). Os videos são transcodificados para que o tempo de envio aos servidores de distribuição caiam, assim como o tempo de download também caia pelas partes interessadas através do servidor de distribuição, para que o vídeo em questão possa ser assistido enquando o download das partes seguintes continue em paralelo.

Nesta dissertação, a compressão é realizada pelo Morph, um software de transcodificação de vídeo distribuída *opensource*, que pode ser utilizado em nuvem, permitindo a escalabilidade dos recursos para codificação e decodificação de arquivos, numa velocidade relativa à quantidade de recursos alocados. É uma implementação escalável desenvolvida em *Python*, que configura uma arquitetura nós mestre-escravos, onde o nó mestre tem um banco de dados *MySQL* e sua aplicação atua como um servidor, recebendo a requisição de transcodificação de vídeo, dividindo o arquivo recebido em pedaços, realiza a concentração, a designação e operações de agendamento. Os nós escravos, por sua vez, são responsáveis por se conectar ao servidor mestre e receber pedaços de vídeo para efetuar a sua devida transcodificação. O Morph pode transcodificar os blocos de vídeo de forma paralela, através dos seus diversos escravos, para obter uma velocidade mais rápida, além de gerenciar as transferências de dados e comunicações entre os nós mestre e escravos. Na implementação desta arquitetura, representada na Figura 5, o nó escravo pode sair ou entrar do cluster a qualquer momento, graças a alocação dinâmica de recursos (GAO; WEN, 2016).

A escolha desta solução se deve ao fato de ser de código aberto, amplamente documentada e compatível com a nuvem *OpenStack*, utilizada por este trabalho.

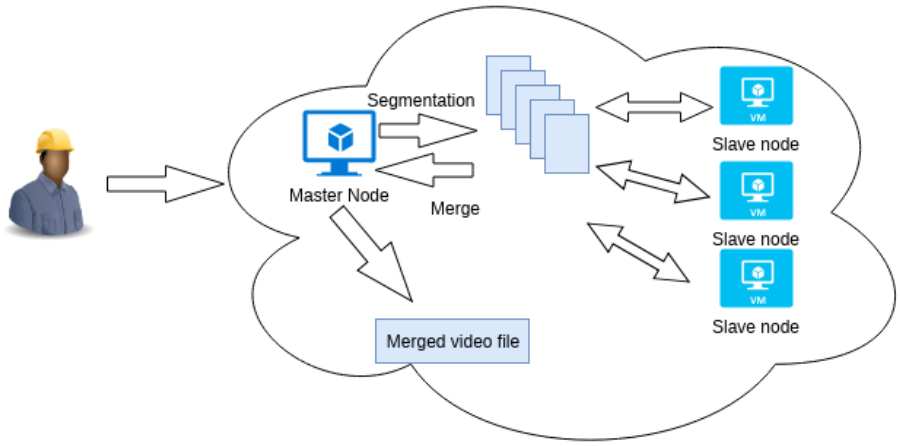

Figura 5 – Arquitetura Morph

3.6 INJEÇÃO DE FALHA E REPARO

De acordo com Avizienis et al. (2004), o termo dependabilidade diz respeito à capacidade de um determinado sistema ou subsistema de prover um conjunto de serviços confiáveis, que podem ser observados por agentes externos. Um defeito pode ser definido como uma falha de um componente de um sistema, de um subsistema ou de outro sistema que interage com o sistema sendo observado (AVIZIENIS et al., 2004). Como consequência, o sistema em questão pode falhar e não cumprir o papel que lhe é designado no determinado instante. É possível ocorrer a inutilização do sistema até seu posterior reparo. Vários esforços foram feitos no sentido de desenvolver sistemas com tolerância a falhas (RANDELL, 1975), (GRAY; CHERITON, 1989), (SCHOTT, 1995),(REIS et al., 2005) assim como no desenvolvimento de técnicas de injeção de falhas e reparo com diversas aplicações. Já Duraes e Madeira (2006) pontuam três usos mais comuns da simulação de falha:

Validação de mecanismos de tolerância a falhas : consiste em acelerar a ocorrência de falha através da simulação de falhas de sofware;

Predição de cenários de pior caso e auxilio a risco experimental : diz respeito à quantificação de erros e análise de risco de impacto para cenários de pior caso. Também identifica os componentes que oferecem maior risco de falha e maior impacto no sistema;

Benchmarking de dependabilidade : é utilizado no cálculo de métricas relacionadas ao comportamento de um determinado sistema ou componente, no caso da ocorrência de falhas.

Existem diferentes tipos de injeção de falhas, como por exemplo em *hardware*, em simulações de sistema, ou através de *software*. Esta última opção é possível quando se simula uma falha no *hardware* através de um software de implementação de injeção de falha (*Software Implemented Fault Injection*, ou *SWIFI*). Por terem um baixo custo de desenvolvimento e baixa complexidade, são bastante populares (DURAES; MADEIRA, 2006).

Algumas ferramentas *SWIFI* podem ser citadas, como por exemplo:

1. FERRARI (KANAWATI; KANAWATI; ABRAHAM, 1992);
2. FTAPE (TSAI; IYER, 1995);
3. Xception (CARREIRA et al., 1998);
4. Goofi (AIDEMARK et al., 2001);
5. Eucabomber (SOUZA et al., 2013b);
6. Eucabomber 2.0 (BRILHANTE et al., 2014).

FERRARI - **F**ault and **ERR**or **A**utomatic **R**eal-time **I**njector é um injetor de falhas em software com o intuito de simular situações de falhas em *hardware*. Funciona com arquivos de software executáveis injetando falhas que alteram o fluxo de execução normal de determinado software. Por ser uma ferramenta de injeção de falha em softwares em tempo de execução, é dependente de arquitura específica de processador (KANAWATI; KANAWATI; ABRAHAM, 1992).

FTAPE - Fault Tolerance And Performance Evaluator - é uma ferramenta utilizada para comparar computadores tolerantes a falhas. Existem três tipos de injeção de falha nesse software:

1. ***location-based stress-based injection LSBI*** - ou injeção baseada em local e quantidade de carga;
2. ***time-based stress-based injection TSBI*** - injeção baseada em tempo e quantidade de carga;
3. ***randomly*** onde o tempo é aleatoriamente selecionado, baseado numa distribuição específica.

Estas falhas funcionam estressando determinado sistema através de um módulo de geração de carga em determinado local (CPU, memória ou disco), e ainda, injetando falhas em momentos específicos, relativos à quantidade de carga de trabalho sendo processada, através de determinada quantidade de tempo e carga de trabalho, ou com o tempo beaseado numa distribuição específica (TSAI; IYER, 1995).

Xception é uma ferramenta de injeção de falha e monitoramento que funciona através de *breakpoints* de *hardware*, que também dispõe de um módulo de controle temporal de injeção de falha, utilizado através do *timer* interno disponível nos processadores modernos (CARREIRA et al., 1998). O *Xception* é feito para sistemas baseados na família de processadores *PowerPC*.

GOOFI, ou *Generic Object-Oriented Fault Injection Tool*, é uma ferramenta de injeção de falha e monitoramento escrita em *Java*, utilizando banco de dados compatível com *SQL*, que suporta injeção de falhas através de técnicas *SWIFI* e *SCIFI* (*Scan-Chain Implemented Fault Injection*). Esta ferramenta foi desenvolvida com o intuito de generalizar a injeção de falhas e reparos, permitindo que os usuários possam escrever novas técnicas de injeção de falha, minimizando o esforço relacionado à programação na criação de novos testes (AIDEMARK et al., 2001).

EucaBomber (SOUZA et al., 2013b) é uma ferramenta para realização de testes de disponibilidade em nuvens privadas *Eucalyptus* (NURMI et al., 2009), escrita em *Java*, que utiliza um *framework*, o *FlexLoadGenerator*, para assistência na criação de ferramentas, com o intuito de executar testes de *performance* e disponibilidade. Tem como dois principais módulos, um de gerência de conexão e um de geração aleatória de variáveis. Este software tem como função a injeção de falhas e/ou reparos de *software* e *hardware*, através da simulação via software. Os tempos de falhas e/ou reparos seguem alguma distribuição exponencial das disponíveis no software (Erlang, Exponencial, Pareto, etc.), e seleciona algum dos serviços disponíveis na plataforma *Eucalyptus* (*Cloud Controller*, *Cluster Controller*, *Storage Controller*, *Walrus* ou *Node Controller*).

O sucessor do EucaBomber, EucaBomber 2.0, tem como principal diferença da sua versão anterior a refatoração de sua arquitetura, oferecendo novas métricas que consideram o ponto de vista do usuário (BRILHANTE et al., 2014).

Apesar de todo o trabalho referente ao desenvolvimento de ferramentas SWIFI, elas ainda apresentam diversos problemas e limitações para determinados tipo de uso no que tange à injeção de falhas de forma genérica, nos diversos tipos diferentes de sistema. Seja pela complexidade envolvida na implementação (TSAI; IYER, 1995), pela dependência de arquitetura de processadores (CARREIRA et al., 1998), pela dependência de sistema operacional ou até de impacto no comportamento do sistema, como no caso de execução do *software* em *trace mode*. No caso do Eucabomber (SOUZA et al., 2013b) e Eucabomber 2.0 (BRILHANTE et al., 2014), tal limitação se dá pelo fato do uso de ssh como protocolo de comunicação (sistemas operacionais *Linux* de forma geral), e ainda, pela necessidade

de programação em *Java* para adaptação do software para injeção de falhas em outras arquiteturas de nuvem.

3.7 COMPRESSÃO OU TRANSCODIFICAÇÃO DE VÍDEO

O volume de dados na Internet cresce de forma exponencial (CISCO, 2018). O modo como passamos a produzir, disponibilizar e consumir conteúdos de mídia mudou drásticamente, passando a fazer parte de nossas rotinas de uma forma cada vez mais frequente. Isto é possível graças ao avanço de algumas tecnologias envolvidas na difusão de mídias digitais, em especial, os vídeos. Podemos citar, como exemplo o aumento de conexões mais rápidas e o desenvolvimento e melhora dos algoritmos de compressão de vídeo, que possibilita diminuir o tamanho do vídeo de forma considerável, viabilizando sua transmissão através da internet.

Compressão de vídeo é uma área de pesquisa que busca resolver o problema de transporte dados-fonte de vídeo, com a maior fidelidade possível em determinado *bit rate*, ou, transporte de dados-fonte de vídeo, utilizando o menor *bit rate* possível, e, ao mesmo tempo, mantendo o mínimo de qualidade aceitável (SHANNON, 1959). Em ambos, há um *tradeoff* entre o tamanho do arquivo de vídeo e sua qualidade. Defende-se a existência de eficiência no código de compressão de vídeos, quando este consegue executar o *tradeoff* de forma bem feita. O sistema completo da compressão de vídeo, com a capacidade de realizar a compressão e a descompressão é chamado de *codec* (SULLIVAN; WIEGAND, 2005). Existem diversas técnicas para compressão de vídeo, como por exemplo:

1. **Predição**: um processo onde um conjunto de valores de predição é criado e usado para prever os valores de entrada das amostras de vídeo, de forma que os valores que precisam, necessariamente serem, representados são apenas os valores residuais, ou seja, as diferenças dos valores previstos;

2. **Transformação**: esta técnica consiste em formar um novo conjunto de amostras, a partir de uma combinação de amostras de entrada, geralmente através de combinação linear;

3. **Quantização**: Consiste em reduzir a precisão de um grupo de amostras com o intuito de diminuir a quantidade espaço requerido pela ação de compressão.

Uma forma de comprimir vídeos é comprimir cada *frame* individualmente. A esquemática do formato JPEG (COMMITTEE et al., 1990) consiste em obter amostras com blocos de igual dimensão, que são transformadas por uma transformação *Discrete Cosine Transform* (DCT) (AHMED; NATARAJAN; RAO, 1974) sendo que seus coeficientes são quantificados e transmissíveis utilizando-se códigos de tamanho variado. Este esquema de codificação é chamado de *intra-picture* ou *intra-coding*. A codificação de cada um dos quadros independe da codificação dos antecessores e sucessores. Para um melhor desempenho de

compressão, aproveita-se da redundância temporal dos vídeos, que consiste, basicamente, na repetição de determinada cena e seus quadros posteriores sinalizam apenas as mudanças efetivamente ocorridas nos frames, contrariamente à recompressão de todas as partes. A combinação de duas técnicas de predição e transformação compõem o chamado *codec* híbrido (HABIBI, 1974).

Em *codecs* híbridos modernos, as regiões podem ser previstas, utilizando-se da predição inter-picture. *Motion-compensated prediction* (MCP), é uma técnica que se aproveita do fato de que a maioria das mudanças em um vídeo é causada pelo movimento dos objetos em determinada cena, e uma pequena quantidade de movimento pode resultar em uma grande diferença nos valores das amostras em uma imagem, em especial nas bordas.

Prever uma área da imagem atual, baseando-se na imagem anterior com pouca alteração, pode reduzir, significativamente, a necessidade de refinamento. O uso do deslocamento espacial de vetores de movimento *Movie Vectors* (MVs) para predição é conhecido como *Motion Compensation* (MC), ou compensação de movimento, enquanto a busca dos melhores vetores de movimento é conhecida como *Motion Estimation* (ME), ou estimativa de movimento. A codificação da diferença resultante do sinal de refinamento é conhecida como MCP residual coding, ou codificação residual de MCP (SULLIVAN; WIEGAND, 2005).

A melhoria das técnicas de MCP foram a maior razão das melhorias na codificação de vídeo atingidas por padrões modernos, quando comparados de geração em geração (SULLIVAN; WIEGAND, 2005). O codec H.264/AVC, utilizado pelo software *Morph* (GAO; WEN, 2016), e utilizado nesta dissertação, faz uso de diversas técnicas evolutivas de MCP. Em suma, o algoritmo funciona através da execução de alguns passos (SULLIVAN; WIEGAND, 2005), são eles:

1. Cada imagem é dividida em blocos;
2. A primeira imagem de cada sequência de vídeo é tipicamente codificada como *Intra mode*;
3. Para as imagens subsequentes, ou entre espaços aleatórios de tempo, o modo de codificação *inter-picture* é utilizado na maior parte dos blocos.

A utilização, pelo *Morph*, e por consequência, por esta dissertação, do algoritmo H.264 se dá pelo fato de ser uma tecnologia de compressão amplamente difundida e utilizada, já que é um dos *codecs* com alta taxa de compressão eficiente.

3.8 CONSIDERAÇÕES

Este capítulo introduziu o leitor a conceitos de nuvem, explicando conceitos referentes à nuvem privada *OpenStack* e seus principais componentes utilizados por este trabalho.

Referimos também a aplicação *Morph*, que utilizamos hospedado em uma infraestrutura que utiliza *OpenStack*.

Discorremos a respeito de conceitos de "dependabilidade", mas, em especial, a disponibilidade. Apresentamos técnicas de modelagem estocástica, utilizando modelos *SPN* e *RBD*. Os modelos desenvolvidos por este trabalho utilizam os modelos propostos para extração das métricas de disponibilidade, conforme o interesse deste estudo. Fizemos também uma sumarização a respeito dos conceitos de análise de sensibilidade. Com relação à validação do modelo proposto por este trabalho, fizemos uso de técnicas de injeção de falhas e reparos, conceito também explicado ao leitor por este capítulo.

Por fim, discorremos a respeito de compressão de vídeo, produto final produzido pela infraestrutura proposta ao longo desta pesquisa.

4

Metodologia, arquiteturas de nuvem e injetor de falhas

Este capítulo apresenta a metodologia utilizada para avaliar a disponibilidade da plataforma *OpenStack-Morph*. Além disto, também demonstramos o fluxo de funcionamento do injetor de falhas e reparos desenvolvido por este trabalho. O injetor foi utilizado para auxiliar na validação do modelo, utilizando geração de número aleatórios exponencialmente distribuídos, injeção de falhas e reparos, além de um constante monitoramento da disponibilidade dos sistemas observados. Por fim, apresentamos os comandos de entrada do injetor de falhas e reparos, com suas respectivas descrições e taxas.

4.1 METODOLOGIA

O primeiro item da metodologia (Figura 6) refere-se ao entendimento do sistema, seus principais componentes e interações, pra que seja possível modelar seus principais componentes, com o objetivo de obter as métricas de interesse. Com as métricas de interesse definidas, representada pelo item 2, inicia-se então o processo de modelagem do sistema, demonstrado pelo item 3. Este processo utiliza representações *RBD* e *SPN* na construção dos modelos utilizados por este trabalho, tendo como objetivo o cálculo das métricas de interesse, *MTTFs*, *MTTRs* e, consequentemente, a disponibilidade de todo o sistema. O trabalho propõe três modelos construídos para o sistema proposto, variando a quantidade de nós físicos, buscando o aumento da disponibilidade através de redundância. O item 4 se refere à obtenção dos resultados, sendo realizado através do uso do *software* Mercury

(SILVA et al., 2013), sendo utilizados os modelos desenvolvidos. Em seguida, no item 5, ocorre a validação dos resultados referentes à disponibilidade obtida. Esta validação é efetuada através de experimentos de injeção de falhas e reparos. No caso de os modelos serem validados (representado pelo item 6), efetua-se uma análise de sensibilidade, representada pelo item 7, variando todos os parâmetros do modelo e obtendo um índice de sensibilidade que indica o componente que mais influencia na disponibilidade. Em seguida, propomos novos modelos (item 8) que contemplam o acréscimo de um novo nó, e, consequentemente, mecanismos de redundância na busca por um aumento na disponibilidade. A análise de resultados indicada pelo item 9 se refere aos resultados calculados de disponibilidade, disponibilidade orientada à capacidade, capacidade real e à diferença da variação entre estas métricas.

Além da metodologia, este capítulo também apresenta a arquitetura implantada em laboratório, para experimentos de validação e objeto de estudo deste trabalho.

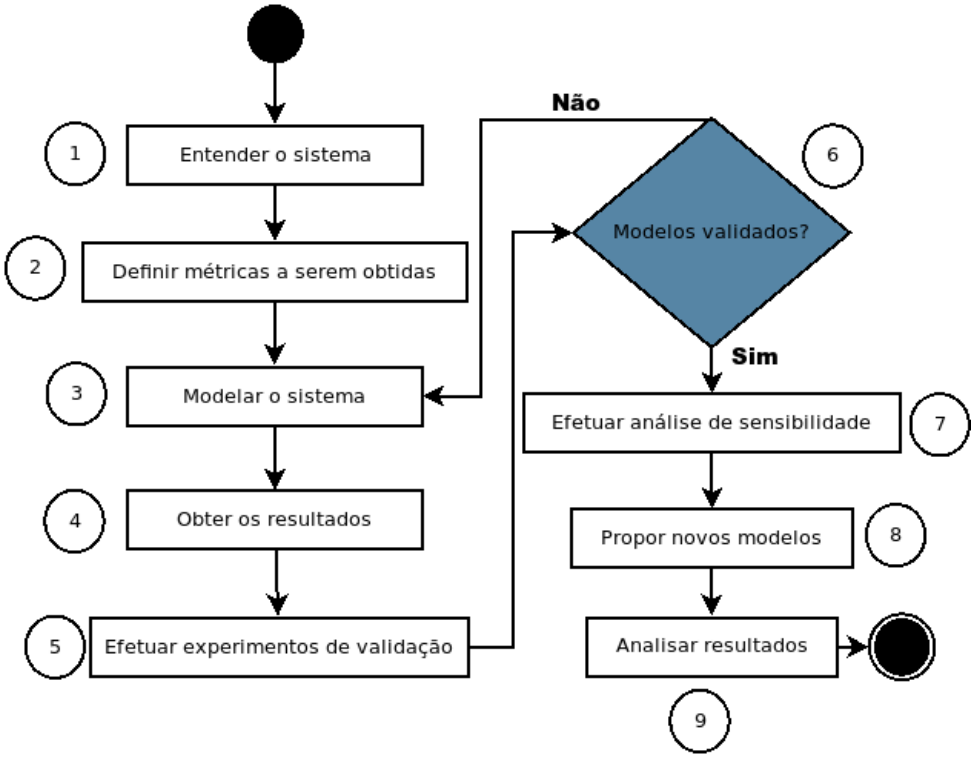

Figura 6 – Metodologia

4.2 ARQUITETURA OPENSTACK-MORPH

O software responsável por realizar a codificação paralela é o *Morph* (GAO; WEN, 2016), que utiliza uma arquitetura de nós mestre e escravos. A informação principal pertinente a todos os nós é gravada em um banco de dados *MySQL*(MYSQL, 2004), localizado na mesma máquina do nó mestre.

Além do *Morph*, para este estudo, uma arquitetura baseada em nuvem foi implantada, especificamente, o *OpenStack*. A razão por trás disso se deve ao fato de que a alocação das máquinas virtuais pode ser feita de forma dinâmica, o que implica numa maior elasticidade do sistema de compressão, permitindo que, em qualquer momento, novas máquinas virtuais sejam adicionadas como escravas ao cluster do *Morph*, aumentando seu desempenho e diminuindo o tempo necessário para a compressão (PEREIRA et al., 2010; GAO; WEN, 2016).

Além disto, a facilidade trazida pelo *network file system* permite que o contexto do *software Morph* seja persistido. Esta persistência ganha redundância quando há mais de um nó *OpenStack*, permitindo que, quando alguma falha de *hardware* ocorra, seja possível instanciar uma nova máquina virtual *Morph* mestre, com o mesmo contexto da máquina que falhou. Por causa disto, o *downtime* é menor, já que a nova máquina virtual *Morph* assume a responsabilidade de continuar a gerência do serviço de compressão.

A Figura 7 ilustra a posição do *Morph* na arquitetura *baseline* proposta. Existem cinco principais elementos: a câmera, o capturador - um computador para receber, segmentar e enviar ao sistema de compressão-, uma nuvem *OpenStack* com um único nó hospedando a aplicação *Morph* e o servidor de *streaming*.

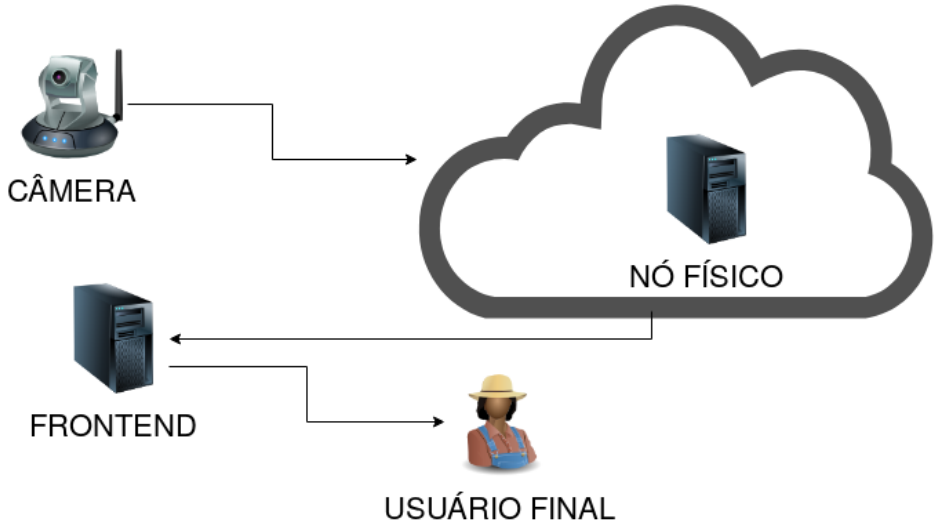

Figura 7 – Arquitetura OpenStack-Morph com um nó

Para que o *Morph* funcione, primeiro uma máquina virtual foi criada com uma imagem *Linux Ubuntu*. Isto se fez necessário por causa da dependência gerada pelo software responsável pela compressão, bem como pela necessidade do banco de dados e do interpretador *Python*. A aplicação *Morph* utiliza *FFMPEG* (BELLARD; NIEDERMAYER et al., 2012), compilado com os módulos x264 (MERRITT; VANAM, 2006), utilizado para a compressão de vídeo.

A câmera de entrada gera o vídeo no servidor, que depois é enviado ao *Morph*. Este, por sua vez, é responsável por dividir o vídeo em tamanhos iguais, de forma pré-configurada, e distribuí-los aos nós escravos, residentes em máquinas virtuais na mesma

nuvem. As máquinas no *cluster* recebem seus respectivos pedaços, comprimindo-os e devolvendo-os ao nó mestre. Depois de receber todos os pedaços dos vídeos de volta, o Morph os reúne, completando sua tarefa.

Além desta arquitetura de nuvem privada, onde a compressão acontece no perímetro, é necessário que os vídeos comprimidos sejam enviados ao servidor de distribuição, que contém uma versão customizada do servidor *web Nginx* (REESE, 2008), com um módulo *RTMP*(BOSE; STIRPE, 2002), um protocolo de *streaming* amplamente utilizado. Por causa desta arquitetura, o usuário apenas sinaliza seu desejo de receber o vídeo, onde o servidor é responsável pelo resto. Já que a intenção é avaliar apenas a disponibilidade de nossa plataforma *Morph-Openstack*, este arranjo foi feito para provar a viabilidade geral de um *streaming* ao vivo, e por isto, o servidor de *streaming* não é considerado neste estudo.

4.3 MODCS INJECTOR

A ferramenta de injeção de falha e reparo desenvolvida no presente estudo tem a missão de injetar falhas que correspondam ao não provisionamento de certos serviços na nuvem, sendo importante validar os resultados obtidos a partir destes modelos. A ferramenta foi escrita utilizando a linguagem de programação *Python* e tem o seguinte comportamento, descrido pelo fluxograma na Figura 8.

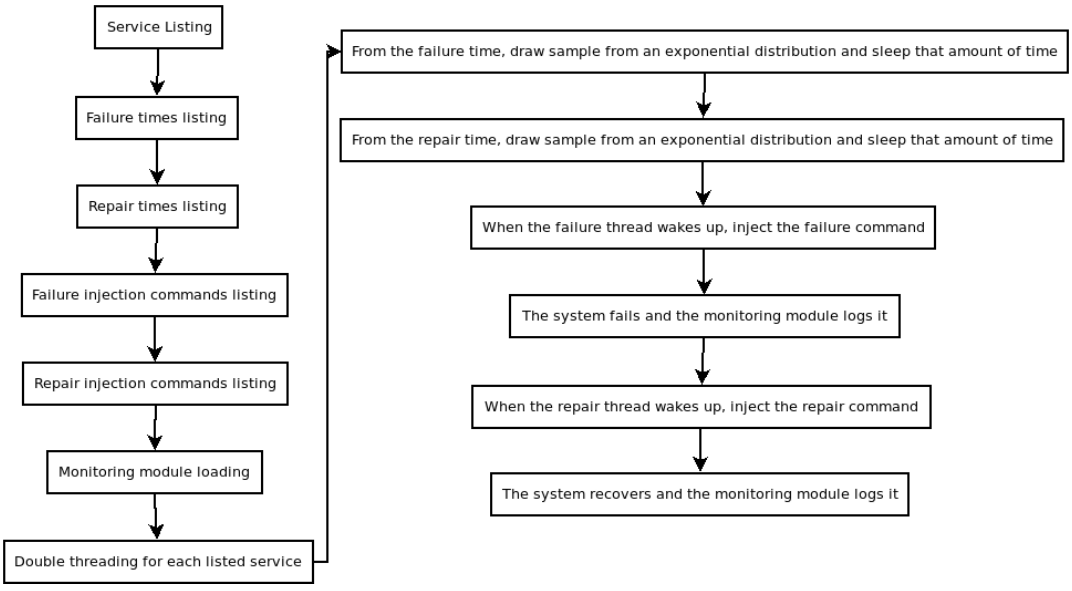

Figura 8 – Fluxograma do Modcs Injector

Os passos necessários para a configuração do injetor são descritos a seguir:

1. Listar todos os serviços nos quais se deseja injetar/monitorar;
2. Listar taxas de falha para cada serviço;
3. Listar o tempo de reparo para cada serviço;

4. Prover *string* equivalente ao comando de falha a ser injetado para simular a falha;
5. Prover *string* equivalente ao comando de reparo a ser injetado para cada um dos serviços.

O usuário precisa listar cada comando necessário para se verificação do status do serviço, com o objetivo de ser utilizado pelo módulo de monitoramento. Esta *string* representando um comando *bash* será executada e deve retornar 0 para falha ou 1 para sucesso.

Após esta etapa, a ferramenta cria duas *threads* para cada serviço: uma para a injeção de falhas e reparos, e outra para o módulo de monitoramento.

A primeira *thread* tem como primeira função obter um número aleatório, exponencialmente distribuído através do uso de uma função *Python*. A *thread* dorme esta quantidade de segundos, e quando acorda, faz a injeção de falha no dispositivo. Novamente um número aleatório exponencialmente distribuído é adquirido e a *thread* dorme aquela quantidade de tempo. Depois deste último passo, a *thread* acorda, injeta o reparo e recomeça o seu ciclo.

A segunda *thread* tem como objetivo o monitoramento constante dos estados de cada componente. Quando o módulo responsável pelo monitoramento carrega, este se mantém num *loop* infinito, verificando o estado dos serviços listados, verificação esta que deve ser efetuada pela utilização do comando de verificação provido pelo usuário. Quando há uma mudança entre os estados disponível e não disponível; esta *thread* grava em um arquivo a data e hora exatas em que a mudança de estado ocorreu, indicando o estado do sistema atual.

O *software* executa as *threads* até que um comando de saída seja emitido, ou o software finalizado. Os comandos para injeção de falha, injeção de reparo, comando para verificação do serviço e as taxas de falha e reparo que utilizamos neste trabalho, estão demonstradas na Tabela 2, cujos valores das taxas utilizadas representam as inversas dos MTTFs e MTTRs.

Ao fim dos experimentos, o arquivo gerado com registros de mudanças de estado dos componentes trouxe todas as informações necessárias para o cálculo da sua disponibilidade. Também possibilitou cálculos relativos à validação do modelo, a exemplo da validação efetuada por este trabalho.

Tabela 2 – Parâmetros de entrada para o injetor

Serviço	Comando p/ verificação	Comando p/ falha	Comando p/ reparo	Taxa de falha (h^{-1})	Taxa de reparo (h^{-1})
Hardware	ping -W 1 -c 1 192.168.0.200 &> /dev/null && echo "1| echo "0";	ssh root@192.168.0.200 "echo o >/proc/sysrq-trigger"	'wakeonlan -i 192.168.0.0 2c:59:e5:9a:97:f0'	0.000012816	0.000138889
VM	ping -W 1 -c 1 192.168.0.236 &> /dev/null && echo "1| echo "0";	ssh ubuntu@192.168.0.236 -i /root/chavecloudnova.pem sudo service networking stop	ssh ubuntu@192.168.0.236 -i /root/chavecloudnova.pem && source /root/keystonerc_admin; /usr/bin/nova reboot b538798d-3af5-46a5-997d-4e4fce01b9b3	0.000020763	0.003333333

5

Modelos das arquiteturas

O sistema foi abstraído utilizando-se os modelos hierárquicos, para que as métricas de interesse pudessem ser adquiridas. Uma posterior análise de sensibilidade nos modelos fundamenta a proposta de novos modelos. Os modelos feitos em *RBD* foram utilizados para obtenção das métricas de confiabilidade do sistema, enquanto o modelo *SPN* proposto é utilizado em combinação com os modelos *RBD*, por causa das limitações presentes no formalismo *RBD*.

5.1 RBD

A modelagem através de RBD foi escolhida por causa da interdependência de seus componentes, para que o sistema continue funcionando, além do fato de não existir prioridade entre os componentes.

A Figura 9 representa a arquitetura básica do sistema, sem nenhum tipo de redundância. A estrutura geral é representada pelos componentes FrontEnd e Nó físico, que é subdividido em duas subcategorias: Nó físico (MN) e *Morph Virtual Machine* MVM. O FrontEnd (FE) é a parte responsável pela transmissão do conteúdo de vídeo transcodificado, que age como um servidor de *VoD streaming*, recebendo o conteúdo transcodificado e retransmitindo sob demanda para o usuário final. O nó físico representa o equipamento físico utilizado na implantação do *OpenStack*. Dividimos o nó físico em duas camadas. A primeira camada, representa a parte física e o sistema operacional hospedeiro, ou *Main Node* (MN), enquanto a segunda parte representa a máquina virtual, o componente MVM. Este componente é responsável por gerenciar um *cluster* de transcodificação de vídeo em

nuvem, utilizando o *Morph*. Todos os tempos de MTTF e MTTR utilizados pelos modelos RBD deste trabalho são exponencialmente distribuídos, tendo sido adquiridos nos trabalhos de Mel et al. (MELO et al., 2017) e Dantas et al. (DANTAS et al., 2012).

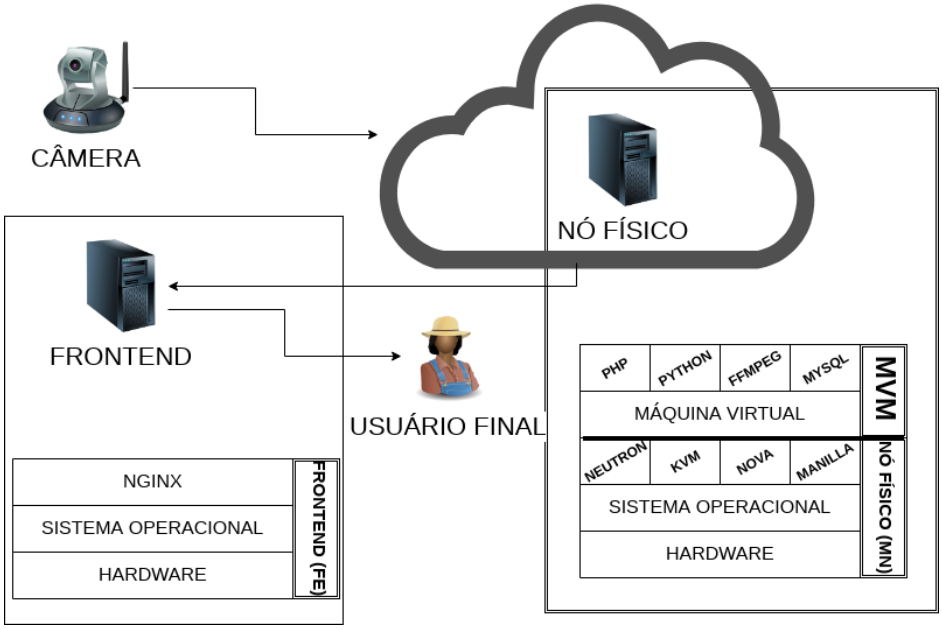

Figura 9 – RBD da arquitetura básica OpenStack-Morph

O modelo RBD ilustrado na Figura 10 é uma representação hierárquica da estrutura geral com os três blocos: FE, MVM e MN. Cada um dos blocos representa o subsistema com seu próprio modelo RBD. A avaliação de disponibilidade da arquitetura geral é apresentada na Equação 5.1.

Figura 10 – Modelo RBD da Estrutura Geral

$$R_{GS} = R_{FE} \times R_{MVM} \times R_{MN} \tag{5.1}$$

O modelo *RBD* apresentado na Figura 11 representa o *FrontEnd*. É um modelo RBD em série, com seus componentes sendo representados pelos blocos HW (*hardware*), OS (*operating system*) e NGINX (*web server*) (NEDELCU, 2010). A confiabilidade deste modelo *RBD* pode ser representada pela Equação 5.2.

$$R_{FRONTEND} = R_{HW} \times R_{OS} \times R_{NGINX} \tag{5.2}$$

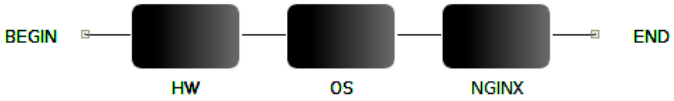

Figura 11 – Modelo RBD do subsistema FrontEnd

O modelo RBD que representa a máquina virtual mestre da aplicação *Morph* ilustrado pela Figura 12 tem os componentes VM (*virtual machine*), OS (*operating system*), PYTHON (SANNER et al., 1999), MYSQL (MYSQL, 2001), FFMPEG (BELLARD, 2001) e PHP (GROUP et al., 2007), como blocos independentes e a expressão de sua confiabilidade é apresentada na Equação 5.3. É importante ressaltar que a falha de qualquer um dos componentes em série acarreta a falha do sistema e ao não provimento do serviço.

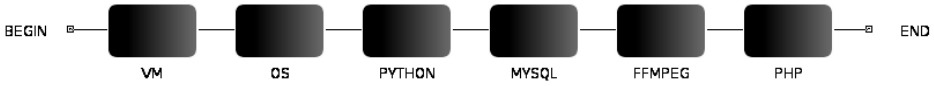

Figura 12 – Modelo RBD subsistema MVM

$$R_{MVM} = R_{VM} \times R_{OS} \times R_{PYTHON} \times R_{MYSQL} \times R_{FFMPEG} \times R_{PHP} \tag{5.3}$$

O subsistema MN, apresentado na Figura 13 é composto pelos blocos HW (*hardware*), OS (*operating system*), NEUTRON, KVM (*Kernel-based virtual machine*) (DREPPER, 2013), NOVA e MANILA (KHEDHER; CHOWDHURY, 2017). A disponibilidade deste subsistema é computada de acordo com a Equação 5.4.

$$A_{MN} = A_{HW} \times A_{OS} \times A_{NEUTRON} \times A_{KVM} \times A_{NOVA} \times A_{MANILLA} \tag{5.4}$$

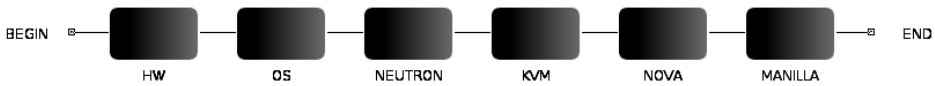

Figura 13 – Modelo RBD do subsistema MN

5.2 MODELO DA ARQUITETURA 1

A arquitetura *baseline* é composta por um simples nó com uma nuvem *OpenStack* e uma máquina virtual hospedada neste nó. Esta arquitetura é utilizada para uma comparação entre diferentes arquiteturas e modelos descritos por este trabalho.

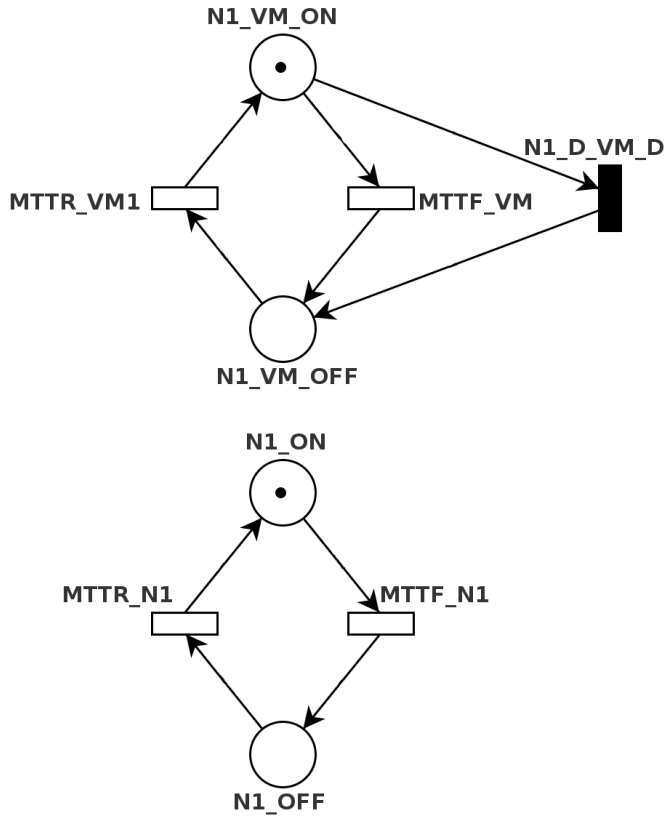

Figura 14 – Rede de Petri com um nó

Tabela 3 – Expressões de guarda do modelo SPN um nó

Transição	Expressão de guarda
N1_D_VM_D	N1_ON==0
MTTR_VM1	N1_OFF==0

A Figura 14 representa nosso primeiro modelo *SPN*. O lugar N1_ON representa o número de nós físicos disponíveis, o lugar N1_VM_ON representa o número de máquinas virtuais disponíveis, o lugar N1_OFF representa o número de nós físicos indisponíveis e o lugar N1_VM_OFF representa o número de máquinas virtuais indisponíveis. É importante salientar que a parte superior do modelo SPN, que compreende os lugares N1_VM_ON e N1_VM_OFF, e as transições MTTR_VM1, MTTF_VM1 e N1_D_VM_D representam o serviço completo descrito no RBD MN (Figura 12). No caso de falha de qualquer um dos componentes em série descritos no RBD, o sistema fica indisponível. A transição MTTF_VM1 representa o MTTF da máquina virtual, enquanto a transição MTTR_VM1 representa o MTTR da máquina virtual. A transição N1_D_VM_D é uma transição imediata, que tem uma expressão de guarda que dispara quando há uma falha no nó, colocando todos os tokens do lugar N1_VM_ON no lugar N1_VM_OFF. A transição MTTR_VM1 tem uma expressão de guarda que impede que ela seja disparada, caso não existam tokens no lugar N1_ON, já que máquinas virtuais

não podem estar rodando, caso o nó esteja no estado DOWN. Todas as transições imediatas dos modelos deste trabalho utilizam a semântica *single-server*, onde um intervalo de tempo é marcado quando a transição é habilitada pela primeira vez, e um novo intervalo é marcado quando há o disparo de uma transição que continua habilitada na nova marcação. Isto significa que os conjuntos de tokens que habilitam a transição são processados serialmente, e que a especificação temporal associada à transição é independente do grau de habilitação. Também é importante ressaltar que não há concorrência entre todas as transições imediatas dos modelos propostos por este trabalho, onde consideramos que o peso da prioridade de cada uma destas transições é de 1. As expressões de guarda deste modelo estão descritas na Tabela 3.

As métricas de disponibilidade, *uptime*, *downtime*, COA e disponibilidade orientada à capacidade total (TCA) foram extraídas através do Mercury, utilizando, respectivamente, as expressões 5.5, 5.6, 5.7, 5.10 e 5.11. Para o cálculo do *uptime* e *downtime* em medida de tempo, multiplicamos o resultado da disponibilidade por 8760 horas, referentes a um ano. No caso de downtime, este valor resultante da multiplicação é posteriormente subtraído de 8760 horas. As Equações matemáticas da COA (Equação 5.8) e da TCA_{BS} (Equação 5.9) são apresentadas. O i indica a quantidade de recursos disponíveis, que é multiplicado pela recompensa do estado S_i (πi) e, depois, o somatório é dividido pelo número máximo de recursos, cujo valor, neste modelo, é sete.

$$(P\{(\#N1\_VM\_ON = 1)\}) \tag{5.5}$$

$$(P\{((\#N1\_VM\_ON = 1))\} \times 8760) \tag{5.6}$$

$$(8760 - (P\{(\#N1\_VM\_ON = 1))\} * 8760) \tag{5.7}$$

$$COA_{OneNode} = \frac{\sum_{i=1}^{7} i \times (\pi_i)}{7} \tag{5.8}$$

$$TCA_{BS} = \sum_{i=1}^{7} i \times (\pi_i) \tag{5.9}$$

$$\begin{aligned} &(P\{(\#N1\_VM\_ON = 1)\}+ \\ &P\{(\#N1\_VM\_ON = 2) * 2\}+ \\ &P\{(\#N1\_VM\_ON = 3) * 3\}+ \\ &P\{(\#N1\_VM\_ON = 4) * 4\}+ \\ &P\{(\#N1\_VM\_ON = 5) * 5\}+ \\ &P\{(\#N1\_VM\_ON = 6) * 6\}+ \\ &P\{(\#N1\_VM\_ON = 7) * 7\}) \div 7 \end{aligned} \tag{5.10}$$

$$
\begin{aligned}
(P\{((\#N1\_VM\_ON = 1))\}+ \\
P\{((\#N1\_VM\_ON = 2) * 2)\}+ \\
P\{((\#N1\_VM\_ON = 3) * 3)\}+ \\
P\{((\#N1\_VM\_ON = 4) * 4)\}+ \\
P\{((\#N1\_VM\_ON = 5) * 5)\}+ \\
P\{((\#N1\_VM\_ON = 6) * 6)\}+ \\
P\{((\#N1\_VM\_ON = 7) * 7)\})
\end{aligned}
\tag{5.11}
$$

5.3 MODELO DA ARQUITETURA 2

A Figura 16 ilustra o modelo *SPN* para a arquitetura com dois nós, que pode ser vista na Figura 15.

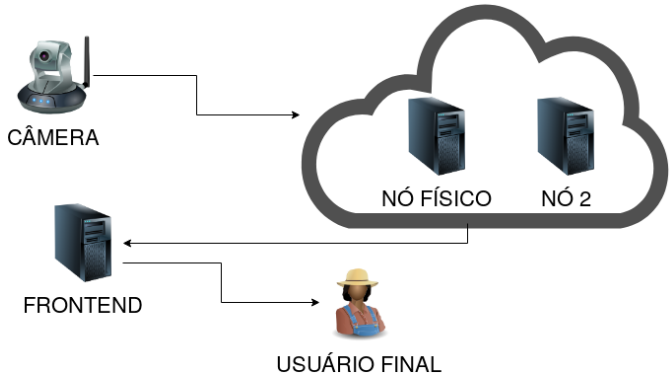

Figura 15 – Arquitetura OpenStack-Morph com dois nós

Esta arquitetura utiliza uma nuvem *OpenStack* implantada com dois nós físicos, e uma replicação total de todos os seus componentes. O objetivo desta escolha mira atingir uma redundância total dos componentes de nuvem, fazendo com que os seus componentes não sejam interdependentes, e ainda, garantindo o funcionamento de um nó de forma independente, não importando o estado em que o outro nó se encontra.

Os lugares N1_VM_ON, N1_VM_OFF, N1_ON e N1_OFF do segundo modelo, apresentado na Figura 16 representam, respectivamente, o número de máquinas virtuais disponíveis no primeiro nó, o número de máquinas virtuais indisponíveis no primeiro nó, o número de máquinas físicas disponíveis no primeiro nó e o número de máquinas físicas indisponiveis no primeiro nó.

A transição MTTF_VM_1, refere-se ao MTTF das máquinas virtuais no primeiro nó, enquanto a transição MTTR_VM_1 refere-se ao MTTR das máquinas virtuais no primeiro nó. A transição imediata N1_D_VM_D possui uma expressão de guarda, restringindo seu disparo para somente quando não houverem tokens no lugar N1_ON. Esta

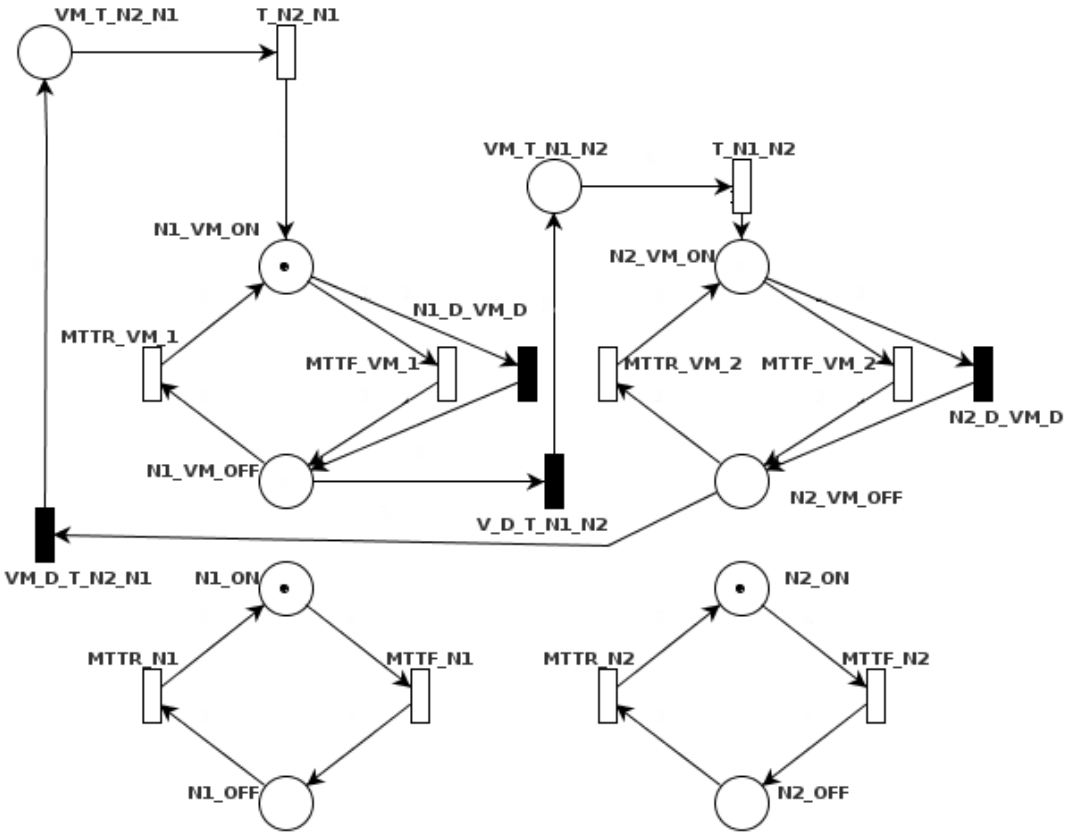

Figura 16 – Rede de Petri com dois nós

transição imediata representa um disparo ao estado do sistema onde a máquina virtual está em seu estado OFF (N1_VM_OFF), pelo fato de seu nó físico estar desligado.

A transição imediata V_D_T_N1_N2, representa um disparo para o estado do sistema no qual a máquina virtual do primeiro nó está indisponível, enquanto o segundo nó está disponível para instanciar uma nova máquina virtual, idêntica à primeira. A máquina virtual que é instanciada no segundo nó se inicia com o mesmo contexto da máquina virtual onde ocorreu a falha. O contexto da compressão transcodificada é armazenado em um *network file system* (NFS), e montado na máquina virtual durante sua inicialização. Esta transição tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N1_ON for igual a zero, e a quantidade de tokens no lugar N2_ON for maior que zero, indicando que o primeiro nó está no estado DOWN.

A transição MTTF_N1 representa o MTTF do primeiro nó, enquanto a transição MTTR_N1 é relativa ao MTTR do primeiro nó. O lugar VM_T_N1_N2 representa a instanciação da máquina virtual do primeiro nó ao lugar N2_VM_ON, que representa o número de máquinas virtuais disponíveis no segundo nó através da transição T_N1_N2.

Os lugares N2_VM_OFF, N2_ON e N2_OFF representam, respectivamente, o número de máquinas virtuais indisponíveis no segundo nó, o número de máquinas físicas

Tabela 4 – Expressões de guarda do modelo SPN com dois nós

Transição	Expressão de guarda
MTTR_VM_1	#N1_ON=1
MTTR_VM_2	#N2_ON=1
N1_D_VM_D	#N1_ON=0 AND #N2_ON=0
N2_D_VM_D	#N1_ON=0 AND #N2_ON=0
VM_D_T_N1_N2	#N1_ON=0 AND #N2_OFF=0
VM_D_T_N2_N1	#N2_ON=0 AND #N1_OFF=0

disponíveis no segundo nó e o número de máquinas físicas indisponíveis no segundo nó. A transição MTTF_VM_2 se refere ao MTTF das máquinas virtuais no segundo nó, enquanto a transição MTTR_VM_2 representa o MTTR das máquinas virtuais no segundo nó, enquanto o MTTF e MTTR do segundo nó são representados, respectivamente, por MTTF_N2 e MTTR_N2. A transição imediata N2_D_VM_D tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N2_ON for igual a zero e a quantidade de tokens no lugar N1_ON for maior que zero, indicando que o segundo nó está no estado DOWN, e que o primeiro nó tem uma máquina física disponível. O lugar VM_T_N2_N1 representa a instanciação da máquina virtual do segundo nó onde ocorreu uma falha, ao lugar N1_VM_ON, através da transição T_N2_N1. As expressões de guarda referentes a este modelo são apresentadas na Tabela 4.

As métricas de disponibilidade, *uptime*, *downtime*, COA e TCA foram extraída através do Mercury, utilizando, respectivamente, as expressões 5.12, 5.13, 5.14, 5.17 e 5.18. As Equações 5.15 e 5.16 referem-se à COA e TCA do segundo modelo (Figura 16).

$$(P\{(\#N1\_VM\_ON > 0)OR(\#N2\_VM\_ON > 0)\}) \tag{5.12}$$

$$(P\{(\#N1\_VM\_ON > 0)OR(\#N2\_VM\_ON > 0)\} * 8760) \tag{5.13}$$

$$(8760 - (P\{(\#N1\_VM\_ON > 0)OR(\#N2\_VM\_ON > 0)\} * 8760)) \tag{5.14}$$

$$COA_{TwoNodes} = \frac{\sum_{i=1}^{14} i \times (\pi_i)}{14} \tag{5.15}$$

$$TCA = \sum_{i=1}^{14} i \times (\pi_i) \tag{5.16}$$

$$
\begin{aligned}
&(P\{((\#N1\_VM\_ON = 1)OR(\#N2\_VM\_ON = 1))\}+\\
&P\{((\#N1\_VM\_ON = 2)OR(\#N2\_VM\_ON = 2) * 2)\}+\\
&P\{((\#N1\_VM\_ON = 3)OR(\#N2\_VM\_ON = 3) * 3)\}+\\
&P\{((\#N1\_VM\_ON = 4)OR(\#N2\_VM\_ON = 4) * 4)\}+\\
&P\{((\#N1\_VM\_ON = 5)OR(\#N2\_VM\_ON = 5) * 5)\}+\\
&P\{((\#N1\_VM\_ON = 6)OR(\#N2\_VM\_ON = 6) * 6)\}+\\
&P\{((\#N1\_VM\_ON = 7)OR(\#N2\_VM\_ON = 7) * 7)\}+\\
&P\{((\#N1\_VM\_ON = 8)OR(\#N2\_VM\_ON = 8) * 8)\}+\\
&P\{((\#N1\_VM\_ON = 9)OR(\#N2\_VM\_ON = 9) * 9)\}+\\
&P\{((\#N1\_VM\_ON = 10)OR(\#N2\_VM\_ON = 10) * 10)\}+\\
&P\{((\#N1\_VM\_ON = 11)OR(\#N2\_VM\_ON = 11) * 11)\}+\\
&P\{((\#N1\_VM\_ON = 12)OR(\#N2\_VM\_ON = 12) * 12)\}+\\
&P\{((\#N1\_VM\_ON = 13)OR(\#N2\_VM\_ON = 13) * 13)\}+\\
&P\{((\#N1\_VM\_ON = 14)OR(\#N2\_VM\_ON = 14) * 14)\}\\
&) \div 14
\end{aligned}
\tag{5.17}
$$

$$
\begin{aligned}
&(P\{((\#N1\_VM\_ON = 1)OR(\#N2\_VM\_ON = 1))\}+\\
&P\{((\#N1\_VM\_ON = 2)OR(\#N2\_VM\_ON = 2) * 2)\}+\\
&P\{((\#N1\_VM\_ON = 3)OR(\#N2\_VM\_ON = 3) * 3)\}+\\
&P\{((\#N1\_VM\_ON = 4)OR(\#N2\_VM\_ON = 4) * 4)\}+\\
&P\{((\#N1\_VM\_ON = 5)OR(\#N2\_VM\_ON = 5) * 5)\}+\\
&P\{((\#N1\_VM\_ON = 6)OR(\#N2\_VM\_ON = 6) * 6)\}+\\
&P\{((\#N1\_VM\_ON = 7)OR(\#N2\_VM\_ON = 7) * 7)\}+\\
&P\{((\#N1\_VM\_ON = 8)OR(\#N2\_VM\_ON = 8) * 8)\}+\\
&P\{((\#N1\_VM\_ON = 9)OR(\#N2\_VM\_ON = 9) * 9)\}+\\
&P\{((\#N1\_VM\_ON = 10)OR(\#N2\_VM\_ON = 10) * 10)\}+\\
&P\{((\#N1\_VM\_ON = 11)OR(\#N2\_VM\_ON = 11) * 11)\}+\\
&P\{((\#N1\_VM\_ON = 12)OR(\#N2\_VM\_ON = 12) * 12)\}+\\
&P\{((\#N1\_VM\_ON = 13)OR(\#N2\_VM\_ON = 13) * 13)\}+\\
&P\{((\#N1\_VM\_ON = 14)OR(\#N2\_VM\_ON = 14) * 14)\})
\end{aligned}
\tag{5.18}
$$

5.4 MODELO DA ARQUITETURA 3

O terceiro modelo *SPN* com três nós, ilustrado na Figura 17 é similar ao segundo modelo ilustrado na Figura 16, com a principal diferença sendo a adição de um nó, possibilitando a instanciação de máquinas virtuais no segundo ou terceiro nó, sendo que todo

o comportamento posterior permanece o mesmo, como será descrito a seguir. Novamente, optamos por usar três nós controller, replicando todos os componentes do *OpenStack*, objetivando a redundância total dos componentes para uma maior disponibilidade.

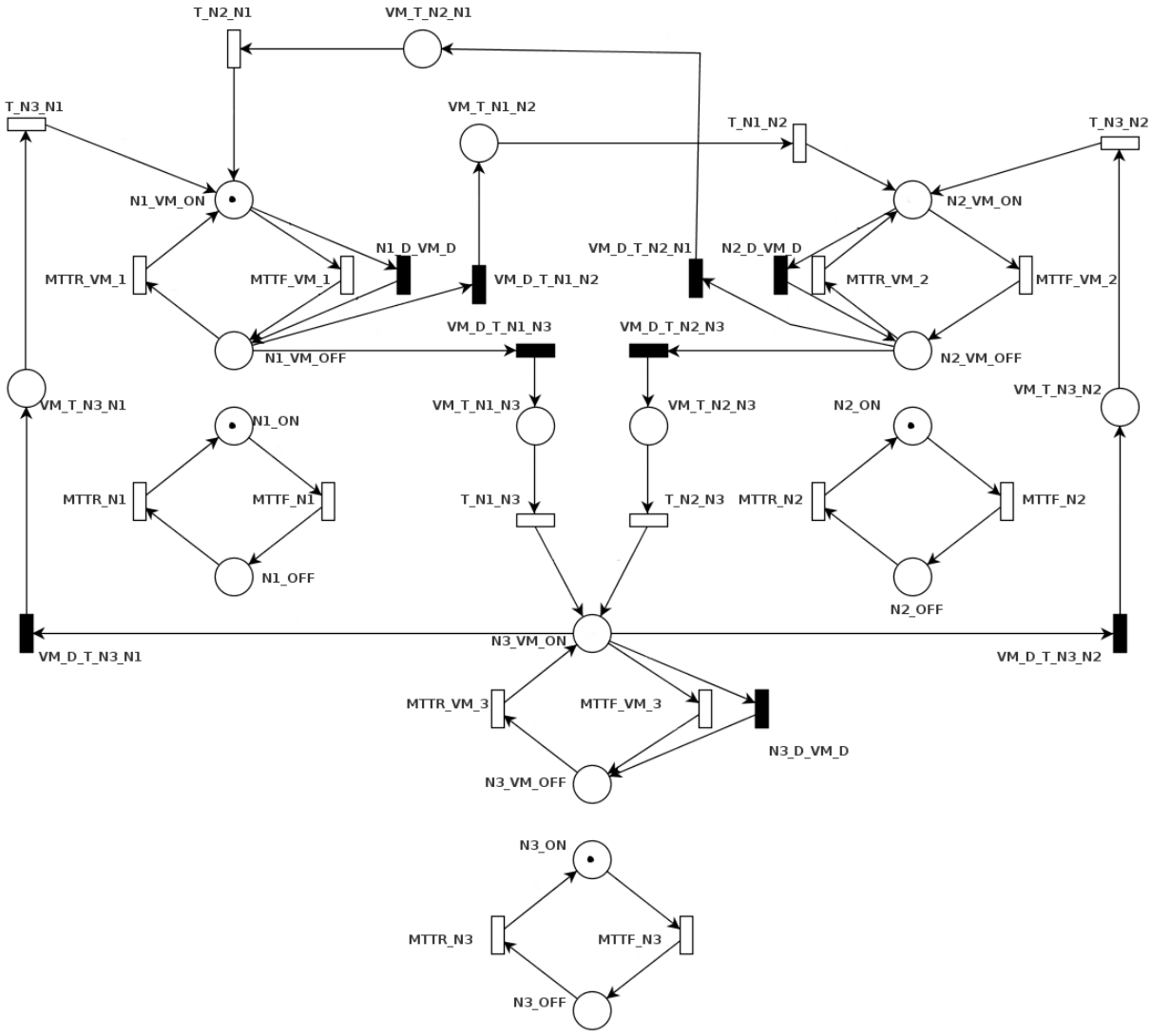

Figura 17 – Rede de Petri com três nós

Os lugares N1_VM_ON, N1_VM_OFF, N1_ON e N1_OFF do terceiro modelo (Figura 17) respectivamente, representam o número de máquinas virtuais disponíveis no primeiro nó, o número de máquina virtuais indisponíveis no primeiro nó, o número de máquinas físicas disponíveis no primeiro nó e o número de máquinas físicas indisponiveis no primeiro nó.

Existem ainda a transição MTTF_VM_1, que se refere ao MTTF das máquinas virtuais no primeiro nó, MTTR_VM_1, representando o MTTR das máquinas virtuais no primeiro nó, a transição imediata N1_D_VM_D, que tem uma expressão de guarda que só permite que ela seja disparada quando não houver tokens no lugar N1_ON. Esta transição imediata representa o estado do sistema no qual a máquina virtual é forçada a ir para o seu estado OFF (N1_VM_OFF) pelo fato do seu nó físico estar desligado.

Os lugares N2_VM_ON, N2_VM_OFF, N2_ON e N2_OFF respectivamente, representam o número de máquinas virtuais disponíveis no segundo nó, o número de máquinas virtuais indisponíveis no segundo nó, o número de máquinas físicas disponíveis no segundo nó e o número de máquinas físicas indisponíveis no segundo nó.

Existem as transições MTTF_VM_2, que se refere ao MTTF das máquinas virtuais no segundo nó, MTTR_VM_2 representando o MTTR das máquinas virtuais no segundo nó, a transição imediata N2_D_VM_D , que tem uma expressão de guarda que só permite que ela seja disparada quando não houver tokens no lugar N2_ON. A transição imediata VM_D_T_N2_N1 tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N2_ON for igual a zero e a quantidade de tokens no lugar N1_OFF for igual que zero, indicando que o segundo nó está no estado DOWN, e que o primeiro nó tem uma máquina física disponível. A transição MTTF_N2 representa o MTTF do segundo nó enquanto a transição MTTR_N2 é relativa ao MTTR do segundo nó.

A transição imediata VM_D_T_N1_N2 tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N1_ON for igual a zero e a quantidade de tokens no lugar N2_OFF for igual a zero, indicando que o primeiro nó está no estado DOWN, e que o segundo nó tem uma máquina física disponível. O lugar VM_T_N2_N1 representa a instanciação da máquina virtual onde ocorreu uma falha no lugar N1_VM_ON, que representa o número de máquinas virtuais disponíveis no segundo nó através da transição T_N2_N1. A transição imediata VM_D_T_N2_N3 tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N2_ON for igual a zero e a quantidade de tokens no lugar N3_OFF for igual a zero, indicando que o segundo nó está no estado DOWN, e que o terceiro nó tem uma máquina física disponível. O lugar VM_T_N2_N3 representa a instanciação de uma cópia da máquina virtual onde ocorreu a falha, ao lugar N3_VM_ON, que representa o número de máquinas virtuais disponíveis no terceiro nó através da transição T_N2_N3.

Os lugares N3_VM_ON, N3_VM_OFF, N3_ON e N3_OFF respectivamente, representam o número de máquinas virtuais disponíveis no terceiro nó, o número de máquinaS virtuais indisponíveis no terceiro nó, o número de máquinas físicas disponíveis no terceiro nó, e o número de máquinas físicas indisponiveis no terceiro nó.

Existem as transições MTTF_VM_3, que se refere ao MTTF das máquinas virtuais no terceiro nó, MTTR_VM_3 representando o MTTR das máquinas virtuais no terceiro nó, a transição imediata N3_D_VM_D , que tem uma expressão de guarda que só permite que ela seja disparada quando não houver tokens no lugar N3_ON. A transição imediata VM_D_T_N3_N1 tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N3_ON for igual a zero e a quantidade de tokens no lugar N1_OFF for igual que zero, indicando que o terceiro nó está no estado DOWN, e que o primeiro nó tem uma máquina física disponível. A transição MTTF_N3 representa o

MTTF do terceiro nó, enquanto a transição MTTR_N3 é relativa ao MTTR do terceiro nó.

A transição imediata VM_D_T_N3_N2 tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N3_ON for igual a zero e a quantidade de tokens no lugar N2_OFF for igual a zero, indicando que o terceiro nó está no estado DOWN, e que o segundo nó tem uma máquina física disponível. O lugar VM_T_N3_N1 representa a instanciação de uma cópia da máquina virtual do terceiro nó, onde ocorreu a falha, ao lugar N1_VM_ON, que representa o número de máquinas virtuais disponíveis no primeiro nó através da transição T_N3_N1. A transição imediata VM_D_T_N3_N2 tem uma expressão de guarda que só permite seu disparo quando a quantidade de tokens no lugar N3_ON for igual a zero e a quantidade de tokens no lugar N2_OFF for igual a zero, indicando que o terceiro nó está no estado DOWN, e que o segundo nó tem uma máquina física disponível. O lugar VM_T_N3_N2 representa a instanciação de uma cópia da máquina virtual do terceiro nó, onde ocorreu a falha, ao lugar N2_VM_ON, que representa o número de máquinas virtuais disponíveis no segundo nó através da transição T_N3_N2.

As métricas de disponibilidade, *uptime*, *downtime*, COA e TCA foram extraídas através do Mercury, utilizando, respectivamente, as expressões 5.19, 5.20, 5.21, 5.24 e 5.25. As Equações 5.22 e 5.23 são referentes ao cálculo da COA e da TCA.

$$(P\{(\#N1\_VM\_ON > 0)OR(\#N2\_VM\_ON > 0)OR(\#N3\_VM\_ON > 0)\}) \tag{5.19}$$

$$(P\{(\#N1\_VM\_ON > 0)OR(\#N2\_VM\_ON > 0)OR(\#N3\_VM\_ON > 0)\} * 8760) \tag{5.20}$$

$$(8760 - (P\{(\#N1\_VM\_ON > 0)OR(\#N2\_VM\_ON > 0)OR(\#N3\_VM\_ON > 0)\} * 8760)) \tag{5.21}$$

$$COA_{TrêsNós} = \frac{\sum_{i=1}^{21} i \times (\pi_i)}{21} \tag{5.22}$$

$$TCA = \sum_{i=1}^{21} i \times (\pi_i) \tag{5.23}$$

$$
\begin{aligned}
&(P\{((\#N1\_VM\_ON = 1)OR(\#N2\_VM\_ON = 1)OR(\#N3\_VM\_ON = 1))\}+\\
&P\{(((\#N1\_VM\_ON = 2)OR(\#N2\_VM\_ON = 2)OR(\#N3\_VM\_ON = 2)) * 2)\}+\\
&P\{(((\#N1\_VM\_ON = 3)OR(\#N2\_VM\_ON = 3)OR(\#N3\_VM\_ON = 3)) * 3)\}+\\
&P\{(((\#N1\_VM\_ON = 4)OR(\#N2\_VM\_ON = 4)OR(\#N3\_VM\_ON = 4)) * 4)\}+\\
&P\{(((\#N1\_VM\_ON = 5)OR(\#N2\_VM\_ON = 5)OR(\#N3\_VM\_ON = 5)) * 5)\}+\\
&P\{(((\#N1\_VM\_ON = 6)OR(\#N2\_VM\_ON = 6)OR(\#N3\_VM\_ON = 6)) * 6)\}+\\
&P\{(((\#N1\_VM\_ON = 7)OR(\#N2\_VM\_ON = 7)OR(\#N3\_VM\_ON = 7)) * 7)\}+\\
&P\{(((\#N1\_VM\_ON = 8)OR(\#N2\_VM\_ON = 8)OR(\#N3\_VM\_ON = 8)) * 8)\}+\\
&P\{(((\#N1\_VM\_ON = 9)OR(\#N2\_VM\_ON = 9)OR(\#N3\_VM\_ON = 9)) * 9)\}+\\
&P\{(((\#N1\_VM\_ON = 10)OR(\#N2\_VM\_ON = 10)OR(\#N3\_VM\_ON = 10)) * 10)\}+\\
&P\{(((\#N1\_VM\_ON = 11)OR(\#N2\_VM\_ON = 11)OR(\#N3\_VM\_ON = 11)) * 11)\}+\\
&P\{(((\#N1\_VM\_ON = 12)OR(\#N2\_VM\_ON = 12)OR(\#N3\_VM\_ON = 12)) * 12)\}+\\
&P\{(((\#N1\_VM\_ON = 13)OR(\#N2\_VM\_ON = 13)OR(\#N3\_VM\_ON = 13)) * 13)\}+\\
&P\{(((\#N1\_VM\_ON = 14)OR(\#N2\_VM\_ON = 14)OR(\#N3\_VM\_ON = 14)) * 14)\}+\\
&P\{(((\#N1\_VM\_ON = 15)OR(\#N2\_VM\_ON = 15)OR(\#N3\_VM\_ON = 15)) * 15)\}+\\
&P\{(((\#N1\_VM\_ON = 16)OR(\#N2\_VM\_ON = 16)OR(\#N3\_VM\_ON = 16)) * 16)\}+\\
&P\{(((\#N1\_VM\_ON = 17)OR(\#N2\_VM\_ON = 17)OR(\#N3\_VM\_ON = 17)) * 17)\}+\\
&P\{(((\#N1\_VM\_ON = 18)OR(\#N2\_VM\_ON = 18)OR(\#N3\_VM\_ON = 18)) * 18)\}+\\
&P\{(((\#N1\_VM\_ON = 29)OR(\#N2\_VM\_ON = 19)OR(\#N3\_VM\_ON = 19)) * 19)\}+\\
&P\{(((\#N1\_VM\_ON = 20)OR(\#N2\_VM\_ON = 20)OR(\#N3\_VM\_ON = 20)) * 20)\}+\\
&P\{(((\#N1\_VM\_ON = 21)OR(\#N2\_VM\_ON = 21)OR(\#N3\_VM\_ON = 21)) * 21)\}) \div 21
\end{aligned}
\tag{5.24}
$$

$$
\begin{aligned}
&(P\{((\#N1\_VM\_ON = 1)OR(\#N2\_VM\_ON = 1)OR(\#N3\_VM\_ON = 1))\}+\\
&P\{(((\#N1\_VM\_ON = 2)OR(\#N2\_VM\_ON = 2)OR(\#N3\_VM\_ON = 2)) * 2)\}+\\
&P\{(((\#N1\_VM\_ON = 3)OR(\#N2\_VM\_ON = 3)OR(\#N3\_VM\_ON = 3)) * 3)\}+\\
&P\{(((\#N1\_VM\_ON = 4)OR(\#N2\_VM\_ON = 4)OR(\#N3\_VM\_ON = 4)) * 4)\}+\\
&P\{(((\#N1\_VM\_ON = 5)OR(\#N2\_VM\_ON = 5)OR(\#N3\_VM\_ON = 5)) * 5)\}+\\
&P\{(((\#N1\_VM\_ON = 6)OR(\#N2\_VM\_ON = 6)OR(\#N3\_VM\_ON = 6)) * 6)\}+\\
&P\{(((\#N1\_VM\_ON = 7)OR(\#N2\_VM\_ON = 7)OR(\#N3\_VM\_ON = 7)) * 7)\}+\\
&P\{(((\#N1\_VM\_ON = 8)OR(\#N2\_VM\_ON = 8)OR(\#N3\_VM\_ON = 8)) * 8)\}+\\
&P\{(((\#N1\_VM\_ON = 9)OR(\#N2\_VM\_ON = 9)OR(\#N3\_VM\_ON = 9)) * 9)\}+\\
&P\{(((\#N1\_VM\_ON = 10)OR(\#N2\_VM\_ON = 10)OR(\#N3\_VM\_ON = 10)) * 10)\}+\\
&P\{(((\#N1\_VM\_ON = 11)OR(\#N2\_VM\_ON = 11)OR(\#N3\_VM\_ON = 11)) * 11)\}+\\
&P\{(((\#N1\_VM\_ON = 12)OR(\#N2\_VM\_ON = 12)OR(\#N3\_VM\_ON = 12)) * 12)\}+\\
&P\{(((\#N1\_VM\_ON = 13)OR(\#N2\_VM\_ON = 13)OR(\#N3\_VM\_ON = 13)) * 13)\}+\\
&P\{(((\#N1\_VM\_ON = 14)OR(\#N2\_VM\_ON = 14)OR(\#N3\_VM\_ON = 14)) * 14)\}+\\
&P\{(((\#N1\_VM\_ON = 15)OR(\#N2\_VM\_ON = 15)OR(\#N3\_VM\_ON = 15)) * 15)\}+\\
&P\{(((\#N1\_VM\_ON = 16)OR(\#N2\_VM\_ON = 16)OR(\#N3\_VM\_ON = 16)) * 16)\}+\\
&P\{(((\#N1\_VM\_ON = 17)OR(\#N2\_VM\_ON = 17)OR(\#N3\_VM\_ON = 17)) * 17)\}+\\
&P\{(((\#N1\_VM\_ON = 18)OR(\#N2\_VM\_ON = 18)OR(\#N3\_VM\_ON = 18)) * 18)\}+\\
&P\{(((\#N1\_VM\_ON = 29)OR(\#N2\_VM\_ON = 19)OR(\#N3\_VM\_ON = 19)) * 19)\}+\\
&P\{(((\#N1\_VM\_ON = 20)OR(\#N2\_VM\_ON = 20)OR(\#N3\_VM\_ON = 20)) * 20)\}+\\
&P\{(((\#N1\_VM\_ON = 21)OR(\#N2\_VM\_ON = 21)OR(\#N3\_VM\_ON = 21)) * 21)\})
\end{aligned}
\tag{5.25}
$$

Tabela 5 – Expressões de Guarda do modelo com três nós

Transição	Expressão de Guarda
MTTR_VM_1	#N1_ON=1
MTTR_VM_2	#N2_ON=1
MTTR_VM_3	#N3_ON=1
N1_D_VM_D	#N1_ON=0 AND #N2_ON=0 AND #N3_ON=0
N2_D_VM_D	#N1_ON=0 AND #N2_ON=0 AND #N3_ON=0
N3_D_VM_D	#N1_ON=0 AND #N2_ON=0 AND #N3_ON=0
VM_D_T_N1_N2	#N1_ON=0 AND #N2_OFF=0
VM_D_T_N1_N3	#N1_ON=0 AND #N3_OFF=0
VM_D_T_N2_N1	#N2_ON=0 AND #N1_OFF=0
VM_D_T_N2_N3	#N2_ON=0 AND #N2_OFF=0
VM_D_T_N3_N1	#N3_ON=0 AND #N1_OFF=0
VM_D_T_N3_N2	#N3_ON=0 AND #N2_OFF=0

5.5 CONSIDERAÇÕES

Os modelos propostos por este capítulo têm como objetivo identificar possíveis ganhos na disponibilidade da infraestrutura estudada. O primeiro modelo é proposto com base em uma análise de sensibilidade, indicando que aquele componente modelado pela primeira arquitetura é o de maior interesse no estudo de ganho de disponibilidade. Os modelos subsequentes são consequência do mesmo estudo de sensibilidade feito com o modelo imediatamente anterior proposto.

6

Estudos de caso

Este capítulo apresenta resultados de disponibilidade, análise de sensibilidade e cálculo de COA. As métricas são obtidas através de modelos SPN propostos por este trabalho. Apresentamos quatro seções relativas à validação e modelos.

O propósito do primeiro estudo de caso é a validação do primeiro modelo. Além da validação, propomos outros modelos, com o objetivo de aumentar a disponibilidade do sistema. O segundo estudo de caso propõe um modelo SPN com um nó, obtém métrica de disponibilidade, efetua uma análise de sensibilidade no modelo, e calcula a disponibilidade orientada à capacidade. O terceiro estudo de caso, realiza o cálculo da disponibilidade através do modelo SPN, que contempla a existência de dois nós, além de realizar uma análise de sensibilidade e o cálculo de sua *COA*. O quarto estudo de caso, com três nós, calcula a disponibilidade através do modelo SPN, calcula sua *COA* e finaliza com um comparativo entre as três arquiteturas representadas pelos modelos.

6.1 ESTUDO DE CASO I: VALIDAÇÃO

É importante verificar se o modelo representa de forma real o comportamento do sistema. Para isto, se faz necessária a utilização de algum método de validação, como os que descrevemos em seções anteriores.

Para este trabalho, construímos uma ferramenta que realiza injeções de falhas e reparos na infraestrutura, simulando uma falha ou um reparo real, e auxiliando no processo de validação do modelo proposto. A validação do modelo, no caso deste trabalho, consiste em colher evidências que nos permitam afirmar com 95% de certeza que o modelo corresponde

ao sistema real. Os tempos de falha dos componentes foram reduzidos em um fator de 10 para que a injeção de falha fosse efetuada de forma mais célere, com o objetivo de acelerar os testes de injeção de falhas e reparos.

A infraestrutura foi implementada em um ambiente real e o experimento de injeção de falhas e reparos foi conduzido por quase oitenta e nove dias. Em paralelo, o módulo de monitoramento da mudança de estado do sistema gravou as alterações em um arquivo de log.

Esta validação aqui descrita é específica para nosso caso, onde utilizamos um servidor HP ProLiant DL320e Gen 8 Server, com um processador Intel(R) Xeon(R) CPU E3-1240 V2 @ 3.40GHz e 24GB de memória RAM. O *OpenStack* utilizado foi a versão 3.14.3, o *OpenStack Queens*.

Primeiramente, calculamos a disponibilidade do sistema utilizando o método de (KEESEE, 1965), que nos permite calcular o intervalo de confiança com tempos exponencialmente distribuídos. As Equações 6.1, 6.2 e 6.3 apresentam o cálculo realizado para obtenção desta métrica, e os valores das variáveis são apresentados na Tabela 6. O tempo total disponível é relativo ao tempo em que o serviço estava disponível, o tempo total indisponível é o tempo em que o sistema não estava disponível, seja por falha da máquina virtual ou do nó físico. A quantidade total de falhas se refere à soma do total de falhas da máquina virtual e falhas da máquina física, enquanto a quantidade total de reparos, também. O tempo total do sistema disponível foi de pouco mais de 79 dias, enquanto o tempo total onde o sistema estava indisponível foi de pouco mais de 9 dias, como pode ser observado na Tabela 6.

Tabela 6 – Valores dos parâmetros

Parâmetro	Descrição	Valor
ε	Tempo total disponível (s)	6899316s
η	Tempo total indisponível (s)	823434 s
ω	Quantidade total de falhas	214
κ	Quantidade total de reparos	214

$$\rho = \frac{\varepsilon}{\omega} \tag{6.1}$$

$$\alpha = \frac{\eta}{\kappa} \tag{6.2}$$

$$\hat{\mathrm{A}} = \frac{\rho}{\rho + \alpha} \tag{6.3}$$

Seguindo nosso trabalho de validação, calculamos os limites mínimos e máximos da distribuição da Função F utilizando o Minitab (MINITAB, 1991), e definindo como 95%

nosso intervalo de confiança. Definimos como limite máximo e mínimo o grau de liberdade obtido como sendo o total de falhas e reparos. Esta função utiliza a quantidade de falhas e reparos apresentados na Tabela 7 como seu grau de liberdade, como apresentado na Figura 18. O grau de liberdade utilizado por este trabalho é o somatório da ocorrência das falhas e reparos observadas pelo experimento.

A injeção das falhas e reparos ocorreu nos dois componentes da infraestrutura: o nó físico (MN) e a máquina virtual (MVM) onde ocorre o provimento do serviço. Quando há uma falha na máquina virtual, o serviço é considerado indisponível, e o mesmo ocorre quando há falha no nó físico, já que a máquina virtual não está em execução quando seu *host* está indisponível.

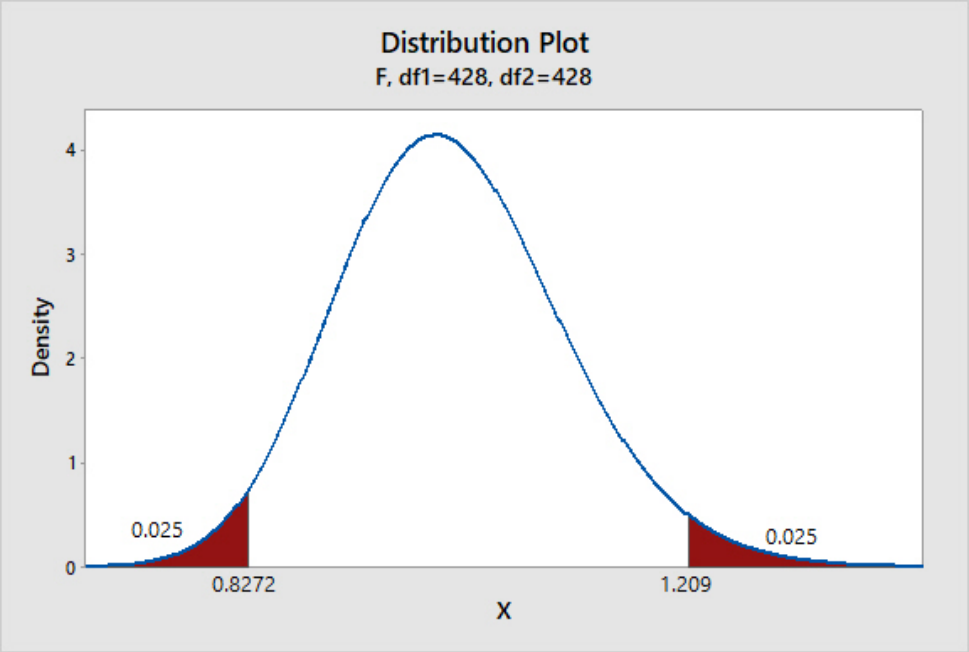

Figura 18 – Gráfico de distribuição de probabilidade

Tabela 7 – Alterações de estados

Descrição	Quantidade
Falhas do nó	75
Reparos do nó	75
Falhas da VM	139
Reparos da VM	139
Ocorrências	428

Com isto, obtemos os valores mínimos e máximos da distribuição, como apresentados na Tabela 8.

Tabela 8 – Valores da distribuição

Descrição	Valor
Limite mínimo	0,8272
Limite máximo	1,209

A relação entre os tempos totais de falha e reparos do sistema resulta na variavel γ, demonstrado na Equação 6.4.

$$\gamma = \frac{\eta}{\varepsilon} \tag{6.4}$$

Obtemos valor de γl e γu. Estes, são calculados através da divisão do valor de γ pelos limites mínimo e máximo da distribuição.

De posse destes valores, calculamos o intervalo da disponibilidade. As Equações 6.5 e 6.6 apresentam as fórmulas utilizadas.

$$A_u = \frac{1}{1 + \gamma u} \tag{6.5}$$

$$A_l = \frac{1}{1 + \gamma l} \tag{6.6}$$

A Tabela 9 apresenta os valores obtidos.

Tabela 9 – Intervalos de confiança para γ e A

Descrição	Valor
γl	0,0987
γu	0,1442
A_l	0,8739
A_u	0,9101
Disponibilidade Modelo Acelerado	90,6389%
Disponibilidade Experimento Pontual	89,3375%

Nosso modelo com taxa de falha acelerada demonstra uma disponibilidade de 90,6389%. Um valor de disponibilidade entre os limites mínimo e máximo (A_l e A_u), sugerindo que o modelo foi validado com 95% de certeza, enquanto a disponibilidade pontual do experimento foi de 89,3375%.

6.2 ESTUDO DE CASO II - ARQUITETURA BÁSICA

Este segundo estudo de caso realiza o cálculo da disponibilidade do modelo SPN com um nó, discutido no Capítulo 5 através do uso da ferramenta Mercury (SILVA et al., 2015).

6.2.1 Disponibilidade da arquitetura I

Consideramos o sistema como disponível quando existe uma máquina virtual *Morph* disponível. O cálculo da disponibilidade do modelo proposto SPN é obtido através do uso da Expressão 5.5 na ferramenta Mercury.

Os resultados da disponibilidade obtida com o modelo SPN podem ser observados na Tabela 11. As métricas de disponibilidade, *uptime* e *downtime* foram calculadas para uma máquina virtual e para sete máquinas virtuais. Consideramos sete máquinas virtuais para calcular a disponibilidade orientada à capacidade neste estudo de caso. Sete é o número máximo de máquinas virtuais que o nó suporta, fazendo que seja necessário o cálculo da disponibilidade do modelo, considerando sete máquinas virtuais. Os valores de entrada da SPN proposta por esta seção são apresentados na Tabela 10, os quais foram calculados a partir dos modelos RBD apresentados anteriormente.

Tabela 10 – Parâmetros de entrada do modelo SPN com um nó

Parâmetro	MTTF (h)
MTTF_{VM}	133,7862
MTTR_{VM}	0,0833333
MTTF_{NODE}	216,7422
MTTR_{NODE}	2

Tabela 11 – Resultados do modelo SPN de um nó

VMS	Disponibilidade (%)	*Uptime* (h)	*Downtime* (h)
1	98,985966%	8671,170705	88,829294972
7	99,047576%	8676,567681	83,432318285

6.2.2 Análise de sensibilidade da arquitetura I

Uma análise de sensibilidade por diferencial percentual foi realizada para se determinar o componente mais importante da arquitetura, no quesito disponibilidade.

Esta análise faz uso dos parâmetros de entrada de MTTF e MTTR de cada componente do sistema (Tabela 10), variando cada um destes componentes e analisando a alteração da disponibilidade do sistema. Os parâmetros sofreram uma variação entre -50% e +50% de seu valor original.

A análise de sensibilidade apresentada na Tabela 12 mostra que a taxa de falha do componente Nó ($MTTF_{VM}$) tem o maior valor (0.012586), demonstrando que deve ser este o componente considerado nas propostas de novos modelos que tenham como objetivo o aumento da disponibilidade. Estes resultados da análise são apresentados pelas Figuras 19, 20, 21 e 22. É possível observar que o impacto na disponibilidade é mais afetado pela variação do MTTF do componente nó físico, indicando que este componente deve receber maior atenção no tangente ao aumento de disponibilidade da infraestrutura de forma geral.

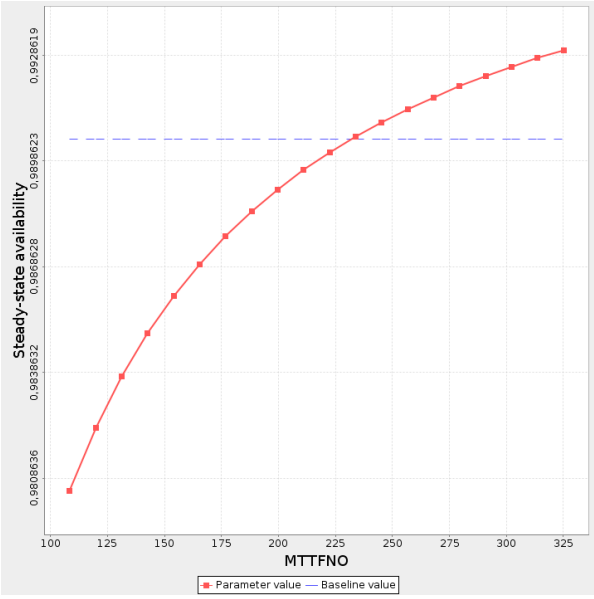

Figura 19 – MTTF do nó

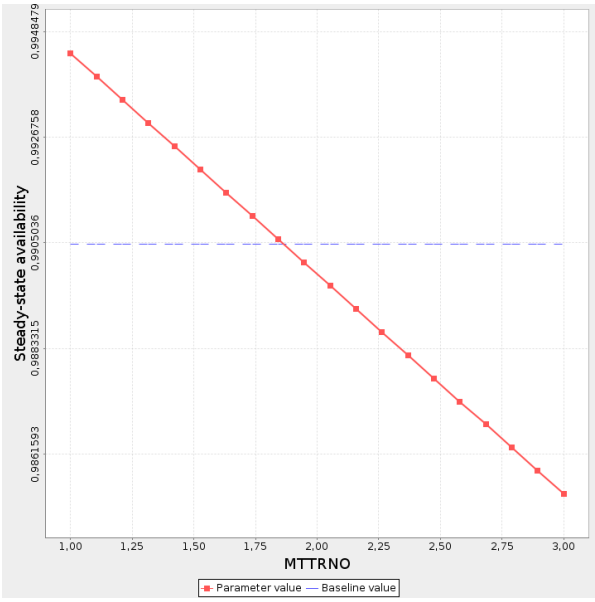

Figura 20 – MTTR do nó

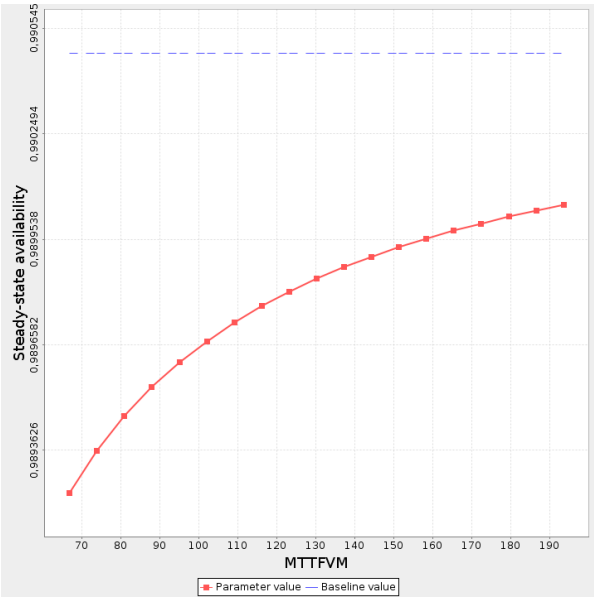

Figura 21 – MTTF da VM

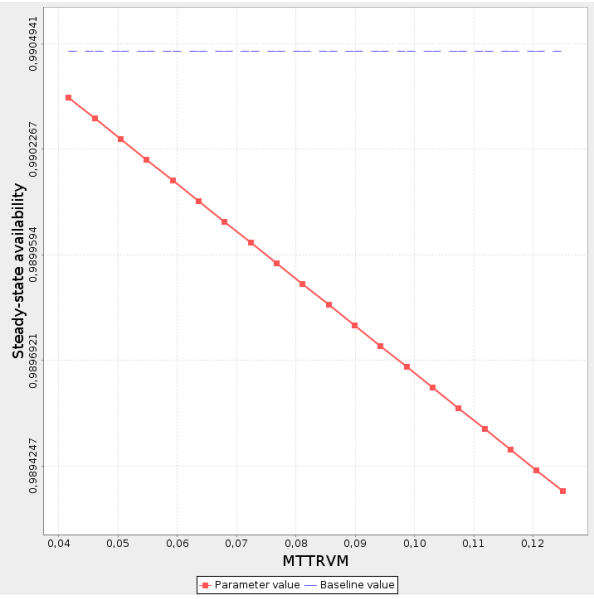

Figura 22 – MTTR da VM

Tabela 12 – Ranking de sensibilidade baseado em derivadas parciais da arquitetura de um nó

Parâmetro	Descrição	-SS(A)-
λ_{Node}	Taxa de falha do nó	0,012586
μ_{Node}	Taxa de reparo do nó	0,009101
μ_{VM}	Taxa de reparo da VM	0,001005
λ_{VM}	Taxa de falha da VM	0,000814

6.2.3 Disponibilidade orientada à capacidade da arquitetura I

A capacidade definida por nosso trabalho é o número de máquinas virtuais que podem ser executadas por cada nó da arquitetura *OpenStack*. A disponibilidade orientada à capacidade é calculada através da Expressão 5.10, e pode ser calculada pelo Mercury através da Expressão 5.8.

Para o cálculo da *COA*, este trabalho considerou que a infraestrutura é capaz de converter um vídeo de 147mb, com a duração de 2 minutos e 17 segundos, em um tempo menor que este. As dimensões do vídeo são consideradas como sendo 1920 x 800 *pixels*, com 24 quadros por segundo, *bit rate* de 8486kbps e codificado em h.264. O resultado esperado é o mesmo vídeo, mas com as dimensões reduzidas em 1280 x 1720 pixels. Considerando que sejam necessárias sete máquinas virtuais para alcançar este objetivo, o estudo da *COA* se faz necessário para identificar a disponibilidade do sistema funcionando com sete máquinas virtuais. A disponibilidade da capacidade total de blocos de serviço (MVM), que chamamos de TCA (*total capacity availability*) é realizada através da Equação 5.9. Os resultados obtidos são apresentados na Tabela 13.

Tabela 13 – Resultados da capacidade do modelo de um nó

Parâmetro	Resultado
Disponibilidade orientada à Capacidade	98,924584
Disponibilidade da Capacidade Total	6,924720

6.3 ESTUDO DE CASO III

Este estudo de caso apresentado faz cálculo da disponibilidade do modelo proposto com dois nós, uma análise de sensibilidade para determinar o fator mais importante na proposição de novos modelos, cujo objetivo é aumentar a disponibilidade, além de também analisar a *COA* e a TCA.

6.3.1 Disponibilidade da arquitetura II

Esta seção apresenta o estudo de caso do modelo com a redundância do componente que mais influencia a disponibilidade do sistema, o nó. O cálculo da disponibilidade, obtida através do Mercury, é apresentado na Expressão 5.12.

Os valores de entrada da SPN proposta são apresentados na Tabela 14. As métricas de disponibilidade, *uptime* e *downtime* foram calculadas para uma máquina virtual e para catorze máquinas virtuais (sete em cada nó).

Tabela 14 – Parâmetros de entrada da SPN com 2 nós

Parâmetro	Descrição	Valor (h)
$MTTF_{N1,N2}$	MTTF dos nós	216,7422
$MTTR_{N1,N2}$	MTTR dos nós	3
$MTTF_{VM1,VM2}$	MTTF das VMS	133,7862
$MTTR_{VM1,VM2}$	MTTR das VMS	0,0833333
T_{N1_N2}, T_{N2_N1}	Tempo para instanciação entre nós	0,0833333

Os resultados do cálculo da disponibilidade são apresentados na Tabela 15. O acréscimo de uma hora ao tempo de reparo do nó se deve ao fato de além do reparo do nó físico, há a recolocação do nó na nuvem.

Tabela 15 – Resultados do modelo SPN com dois nós

VMS	Disponibilidade (%)	Uptime (h)	Downtime (h)
1 VM	99,880721%	8749,551211	10,448788
14 Vms	99,955929%	8756,139410	3,860589

6.3.2 Análise de sensibilidade da arquitetura II

Nesta seção, uma análise de sensibilidade por diferencial percentual também foi realizada, com o objetivo de determinar o componente mais importante, no que se refere à disponibilidade. A análise faz uso dos parâmetros de entrada (Tabela 14), variando seus valores entre -50% e +50% de seu valor original. A análise de sensibilidade apresentada na Tabela 16 mostra o nó com maior importância (0.001153970929142292). O mesmo também pode ser observado nas Figuras 23, 24, 25, e 26, que indicam novamente o nó físico como sendo o componente de maior impacto na disponibilidade do sistema. Por este motivo, a proposição de um novo modelo leva em consideração, e contempla, a existência de três nós.

Tabela 16 – Análise de sensibilidade da arquitetura com dois nós

Parâmtro	Descrição	-SS(A)-
λ_{Node}	Taxa de Falha do nó	0,001153
μ_{Node}	Taxa de reparo do nó	0,001006
μ_{VM}	Repair rate of the VM	0,000814
λ_{VM}	Failure rate of the VM	0,000338

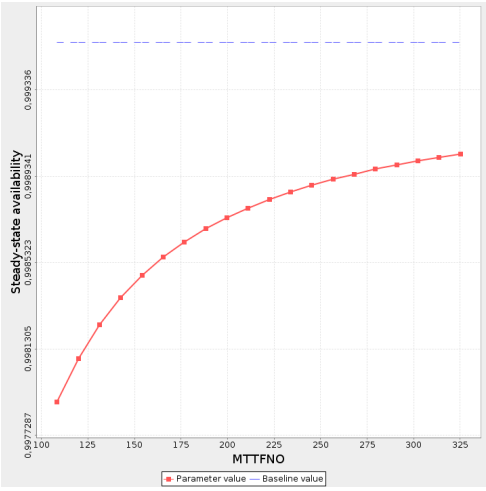

Figura 23 – MTTF do nó

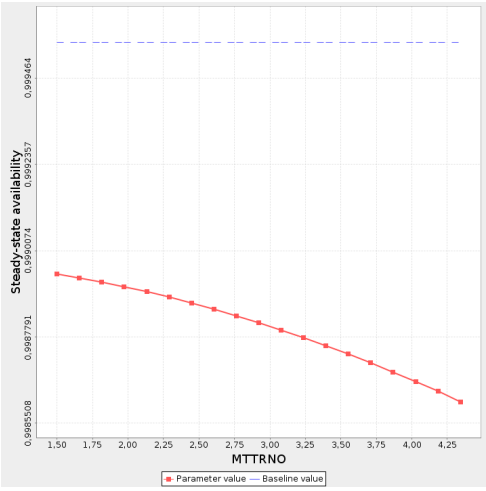

Figura 24 – MTTR do nó

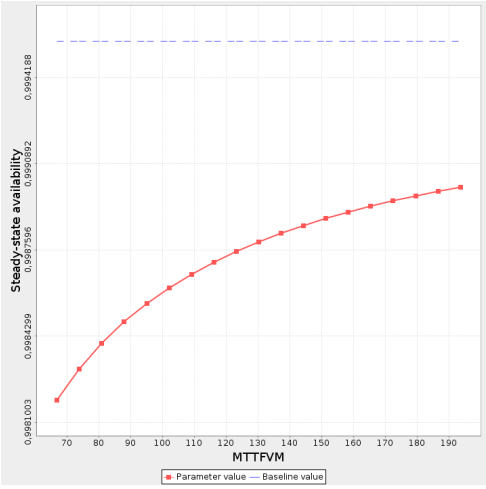

Figura 25 – MTTF da VM

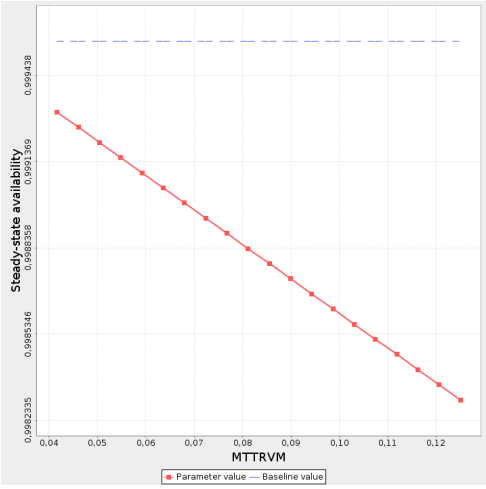

Figura 26 – MTTR da VM

6.3.3 Disponibilidade orientada à capacidade da arquitetura II

No caso desta arquitetura com dois nós, consideramos que a COA é relativa à disponibilidade, considerando sete máquinas virtuais em cada nó, totalizando catorze máquinas virtuais nesta arquitetura.

A Expressão 5.17 foi utilizada no Mercury para obtenção da COA deste modelo, enquanto a Expressão 5.15 demonstra sua fórmula matemática.

Tabela 17 – Resultados da capacidade do modelo de dois nós

Parâmetro	Resultado
Disponibilidade orientada à capacidade	0,998807
Disponibilidade da Capacidade Total	13,958665

6.4 ESTUDO DE CASO IV

Neste último estudo de caso, o cálculo da disponibilidade do modelo proposto com três nós é executado, bem como o cálculo da sua COA.

6.4.1 Disponibilidade da arquitetura III

Neste caso, o estudo da disponibilidade é executado em um modelo acrescentado mais um nó. O sistema é considerado disponível quando existe pelo menos uma máquina virtual *Morph* sendo executada em algum dos nós. O cálculo da disponibilidade, obtida através do Mercury, é demonstrado na Expressão 5.19.

Os valores de entrada da SPN desta seção são apresentados na Tabela 18.

Tabela 18 – Parâmetros de entrada para modelo SPN com três nós

Parâmetro	Descrição	Valor (h)
$\mathrm{MTTF}_{N1,N2,N3}$	MTTF dos nós	216,7422
$\mathrm{MTTR}_{N1,N2,N3}$	MTTR dos nós	3
$\mathrm{MTTF}_{VM1,VM2,VM3}$	MTTF das VMS	133,7862
$\mathrm{MTTR}_{VM1,VM2,VM3}$	MTTR das VMS	0,0833333
$\mathrm{T}_{N1\_N2,N1\_N3}$, $\mathrm{T}_{N2\_N1,N2\_N3}$, $\mathrm{T}_{N3\_N1,N3\_N2}$	Tempo para instanciação entre nós	0,0833333

As métricas de disponibilidade, *uptime* e *downtime* foram calculadas para uma e para vinte e uma máquinas virtuais (sete em cada nó). Os resultados do cálculo da disponibilidade podem ser observados na Tabela 19.

Tabela 19 – Resultados do modelo SPN com três nós

Parâmetro	Disponibilidade (%)	Uptime (h)	Downtime (h)
1 VM	99,899179%	8751,168152	8,831847
7 VMs	99,997907%	8759,845251	0,154748046

Esta seção também faz uma comparação entre os resultados das três arquiteturas propostas. O ganho de disponibilidade é apresentado na Figura 27, e a disponibilidade calculada através dos modelos SPN é apresentada na Tabela 19. Com o aumento da

disponibilidade, há uma diminuição no *downtime* de 88.9086% entre a primeira e a segunda arquitetura. Por sua vez, entre a primeira arquitetura proposta e a última, há uma diminuição de 90.6250% no *downtime* do sistema. A Tabela 20 apresenta as três arquiteturas propostas por este trabalho, além de uma arquitetura correspondente à primeira arquitetura, porém com um tempo de falha acelerado. Esta aceleração foi necessária para a realização dos experimentos de injeção de falha e reparo em um curto espaço de tempo, onde aceleramos em um fator de 10 (dez) o tempo de falha. Pelo fato de a injeção ter sido acelerada, como resultado final de disponibilidade, temos uma diferença da disponibilidade entre as arquiteturas que contemplam um nó.

Tabela 20 – Disponibilidade dos nós

Modelo	Disponibilidade (%)	Uptime(h)	Downtime(h)
Um nó (com aceleração de falha)	90,6389%	7939,9703	820,0296
Um nó	98,9859%	8671,1707	88,8292
Dois nós	99,8807%	8749,5512	10,4487
Três nós	99,8991%	8751,1681	8,8318

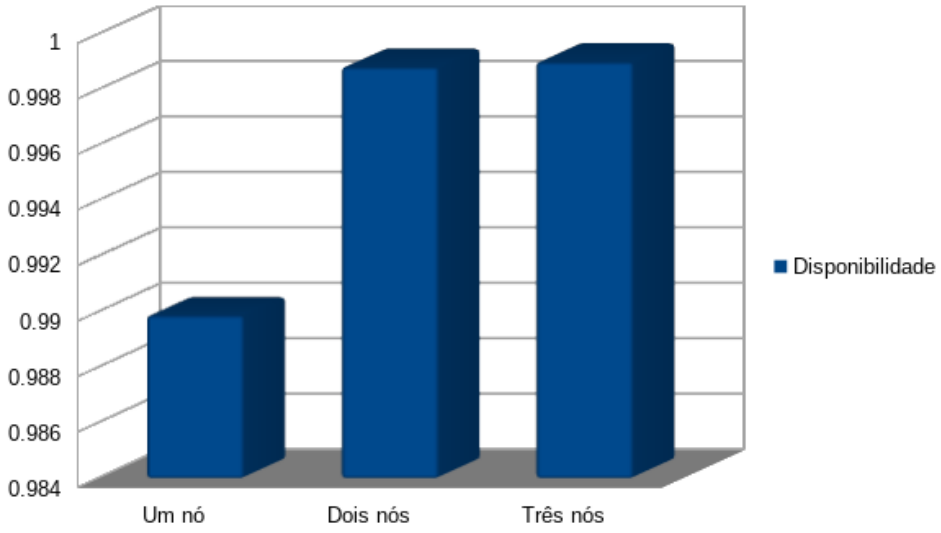

Figura 27 – Disponibilidade das arquiteturas

6.4.2 Disponibilidade orientada à capacidade da arquitetura III

Seguindo a fórmula das seções anteriores, o cálculo da COA considera sete máquinas virtuais para cada nó, totalizando neste caso, vinte e uma máquinas virtuais.

A Expressão 5.22 foi utilizada para calcular a métrica e a Expressão 5.19 foi utilizada no Mercury para realizar o cálculo. A Figura 28 apresenta a diferença entre as disponibilidades orientadas à capacidade, enquanto a Figura 29 apresenta a diferença entre as disponibilidades reais das arquiteturas propostas.

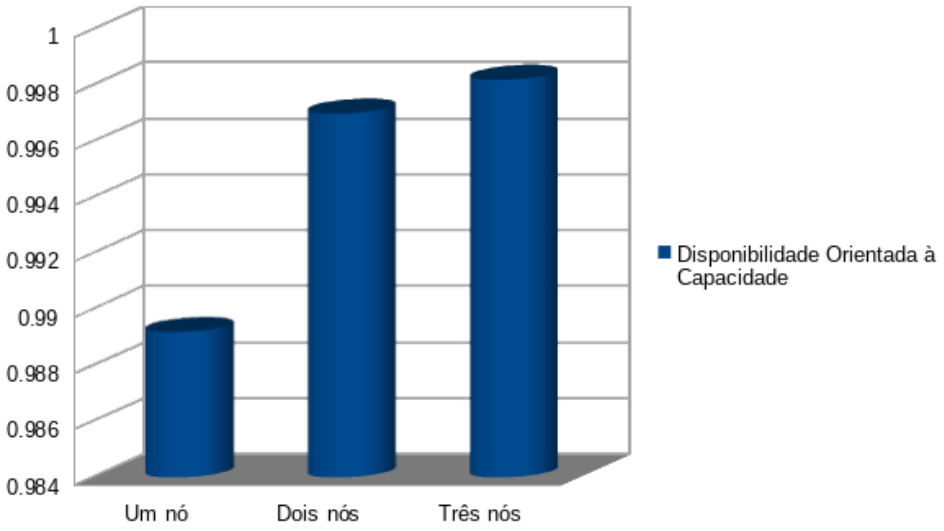

Figura 28 – COA das arquiteturas

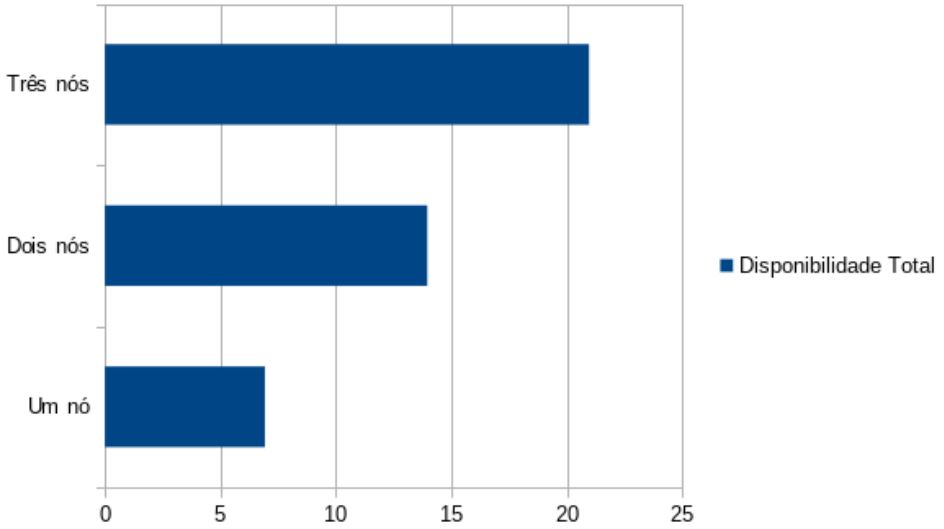

Figura 29 – Disponibilidade da capacidade total

Tabela 21 – Resultados da capacidade do modelo de três nós

Parâmetro	Resultado
Disponibilidade orientada à capacidade	99,825969
Disponibilidade da capacidade total	20,963453

6.5 CONSIDERAÇÕES

Neste capítulo apresentamos a validação do primeiro modelo proposto, através do método de (KEESEE, 1965). Calculamos um intervalo de confiança com auxílio de uma distribuição exponencial F, verificando que o resultado da disponibilidade calculada está dentro dos limites adquiridos.

Este capítulo apresentou quatro estudos de caso de uma infraestrutura de compressão

de vídeo em nuvem privada. Cada um dos estudos de caso é um modelo SPN, que representa os componentes e comportamentos do sistema. Suas transições são tempos exponencialmente distribuídos, enquanto seus lugares representam os possíveis estados do sistema.

Os valores de entrada dos modelos, representando tempos médios de falha e reparo, e tempo de transição entre nós, foram apresentados, bem como todas as equações referentes aos cálculos das métricas desejadas.

O primeiro modelo trata da infraestrutura com um nó, sem nenhuma redundância, tendo os tempos de falha acelerados em um fator de 10. Esta arquitetura possui a disponibilidade mais baixa entre os modelos apresentados. O segundo modelo é igual ao primeiro, com as taxas de falha em seu tempo real. Cálculos para extração da disponibilidade, COA e TCA do modelo também são efetuados. Em seguida, uma análise de sensibilidade é realizada no modelo para identificar o componente de maior relevância no cálculo da disponibilidade.

O terceiro modelo foi criado com uma redundância do nó, identificado anteriormente como componente de maior impacto na disponibilidade. O modelo é analisado e os cálculos de COA, TCA e análise de sensibilidade são extraídos. O resultado da análise de sensibilidade foi levada em consideração durante a abstração de um novo modelo.

O quarto modelo possui uma redundância dupla, sendo que, entre os modelos é o que apresenta a maior disponibilidade. Enquanto o segundo modelo (modelo *baseline* sem aceleração de falha) apresenta uma disponibilidade de 98.9859%, o terceiro modelo possui uma disponibilidade de 99.8991%. Um cálculo da sua COA e TCA também é realizado e apresentado.

7

Conclusão

Por trazer diversas melhorias, a computação em nuvem vem transformando a forma como os serviços hospedados na Internet são mantidos. Pelo fato de haver uma demanda flutuante em alguns destes serviços, a contratação de servidores dedicados que supram esta demanda a todo o tempo é mais custosa, se comparada a alocação e desalocação dos recursos quando necessários. Esta elasticidade de recursos é conhecida como *autoscaling*.

Estas técnicas de *autoscaling*, alinhadas a técnicas de processamento distribuído, permitem que o tempo necessário para realização de tarefas, onde o processamento pode ser distribuído, tenham um tempo de execução relativo à quantidade de recursos alocados.

É o caso da transcodificação de vídeos, uma atividade computacionalmente custosa, e que pode ser distribuída através da arquitetura de nuvem.

Diversos serviços que provêem conteúdo de vídeo dependem da transcodificação para transmissão de seus conteúdos. A transmissão em tempo real requer que a transcodificação seja realizada de forma célere, e que esta plataforma de compressão esteja sempre disponível.

A ocorrência de falhas em qualquer arquitetura é inevitável; entretanto, pode ser contornada através de mecanismos de tolerância a falhas que aumentem a sua disponibilidade.

Este trabalho propôs três modelos SPN representando arquiteturas de transcodificação de vídeo com diferentes números de nós, buscando o aumento da disponibilidade.

O primeiro modelo representa a arquitetura *baseline* que dispõe de um nó. Sua disponibilidade foi calculada juntamente com sua *COA* e TCA. Em seguida, efetuamos uma análise de sensibilidade e detectamos o componente que mais influencia na disponibili-

dade. A representação deste primeiro modelo ocorreu através da modelagem hierárquica, utilizando como base modelos RBD para modelar subsistemas envolvidos e extrair as métricas de MTTF e MTTR utilizadas por este e subsequentes modelos.

Propomos o segundo modelo com redundância do nó, identificado pela análise de sensibilidade do primeiro modelo como fator mais impactante na disponibilidade. Calculamos sua disponibilidade, *COA* e TCA. Novamente, efetuamos uma análise de sensibilidade para identificar os gargalos na disponibilidade.

O terceiro modelo SPN considera uma redundância dupla do nó, também identificado pela análise de sensibilidade do segundo modelo como fator de maior impacto na disponibilidade. Calculamos sua disponibilidade, *COA* e TCA.

A validação do primeiro modelo ocorreu através do uso da ferramenta de injeção de falha e reparo desenvolvida por este trabalho. O experimento nos proveu com dados relativos aos tempos de disponibilidade e indisponibilidade do sistema, bem como a quantidade de falhas e reparos. Utilizamos estas informações para validar com 95% de certeza o nosso primeiro modelo.

Enquanto o primeiro modelo sem aceleração apresentou uma disponibilidade de 98.9859%, o segundo modelo mostra a disponibilidade em 99.8807%, um ganho possível através da redundância dupla do nó. O terceiro modelo apresentou uma disponibilizadade de 99.8991%. Os resultados são úteis no auxílio do planejamento de uma nuvem de transcodificação de vídeo com maior disponibilidade.

7.1 CONTRIBUIÇÕES

Relativo ao que foi apresentado no presente estudo, podemos destacar algumas contribuições:

Proposta de modelos hierárquicos que representam o funcionamento de um serviço de transcodificação de vídeo em nuvem privada *OpenStack*.

A utilização de técnica de análise de sensibilidade por diferencial percentual. Com a identificação dos gargalos na disponibilidade da arquitetura, foi possível propor novos modelos que têm como objetivo o aumento da disponibilidade.

Proposição de novos modelos para o provimento do serviço de transcodificação em nuvem, considerando mecanismos de redundância.

O desenvolvimento de uma aplicação de injeção de falhas e reparos, que pode auxiliar na validação dos modelos desenvolvidos.

7.2 TRABALHOS FUTUROS

Durante a construção desta dissertação, percebemos que uma das maiores limitações se dá pelo fato da complexidade na inclusão de nós, nos modelos apresentados. Assim, se

faz necessária a criação de um modelo SPN genérico, para inclusão de nós em forma de tokens, para que possamos calcular a disponibilidade com N nós.

Além disso, também se faz necessária a avaliação de desempenho da transcodificação, posto que, para ocorrer uma transmissão ao vivo, o desempenho é peça fundamental para que não haja gargalos no sistema.

Referências

AHMED, N.; NATARAJAN, T.; RAO, K. R. Discrete cosine transform. *IEEE transactions on Computers*, IEEE, v. 100, n. 1, p. 90–93, 1974.

AIDEMARK, J. et al. Goofi: Generic object-oriented fault injection tool. In: IEEE. *2001 International Conference on Dependable Systems and Networks*. [S.l.], 2001. p. 83–88.

Apache Software Foundation. *Apache HTTP server project*. 1995. Available at: <http://www.apache.org>.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, IEEE, v. 1, n. 1, p. 11–33, 2004.

BELLARD, F. Ffmpeg. *http://ffmpeg. mplayerhq. hu/*, 2001.

BELLARD, F.; NIEDERMAYER, M. et al. Ffmpeg. *Availabel from: http://ffmpeg. org*, 2012.

BEZERRA, M. C. et al. Availability modeling and analysis of a vod service for eucalyptus platform. In: IEEE. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2014. p. 3779–3784.

BEZERRA, M. C. et al. Availability evaluation of a vod streaming cloud service. In: IEEE. *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. [S.l.], 2015. p. 765–770.

BOLCH, G. et al. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. [S.l.]: John Wiley & Sons, 2006.

BOSE, S.; STIRPE, P. *System, method and applications real-time messaging over HTTP-based protocols*. [S.l.]: Google Patents, 2002. US Patent App. 09/824,541.

BRILHANTE, J. et al. Eucabomber 2.0: A tool for dependability tests in eucalyptus cloud infrastructures considering vm life-cycle. In: IEEE. *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2014. p. 2669–2674.

CARREIRA, J. et al. Xception: Software fault injection and monitoring in processor functional units. *Dependable Computing and Fault Tolerant Systems*, SPRINGER-VERLAG, v. 10, p. 245–266, 1998.

CHATZOPOULOS, D. et al. Video compression in the neighborhood: An opportunistic approach. In: IEEE. *Communications (ICC), 2016 IEEE International Conference on*. [S.l.], 2016. p. 1–6.

CISCO. *Cisco Visual Networking Index: Forecast and Trends, 2017–2022*. [S.l.], 2019.

CISCO, V. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 2018.

CLOTH, L. et al. Model checking markov reward models with impulse rewards. In: IEEE. *2005 International Conference on Dependable Systems and Networks (DSN'05)*. [S.l.], 2005. p. 722–731.

COMMITTEE, J. et al. Digital compression and coding of continuous-tone still images. *Int. Org. Standardization ISO/IEC, JTC1 Commitee Draft, JPEG*, p. 8–R8, 1990.

COMMUNITY, O. *Networking Neutron concepts*. 2019. <https://docs.openstack.org/neutron/rocky/install/concepts.html>. [Online; accessed 19-May-2019].

COSTA, I. et al. Availability evaluation and sensitivity analysis of a mobile backend-as-a-service platform. *Quality and Reliability Engineering International*, Wiley Online Library, v. 32, n. 7, p. 2191–2205, 2016.

DANTAS, J. et al. Models for dependability analysis of cloud computing architectures for eucalyptus platform. *International Transactions on Systems Science and Applications*, v. 8, p. 13–25, 2012.

DANTAS, J. et al. Hierarchical model and sensitivity analysis for a cloud-based vod streaming service. In: IEEE. *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. [S.l.], 2016. p. 10–16.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, ACM, v. 51, n. 1, p. 107–113, 2008.

DREPPER, U. *In-kernel virtual machine for low overhead startup and low resource usage*. [S.l.]: Google Patents, 2013. US Patent 8,429,629.

DURAES, J. A.; MADEIRA, H. S. Emulation of software faults: A field data study and a practical approach. *Ieee transactions on software engineering*, IEEE, v. 32, n. 11, p. 849–867, 2006.

FECHEYR-LIPPENS, A. A review of http live streaming. *Internet Citation*, p. 1–37, 2010.

GAO, G.; WEN, Y. Morph: A fast and scalable cloud transcoding system. In: ACM. *Proceedings of the 2016 ACM on Multimedia Conference*. [S.l.], 2016. p. 1160–1163.

GARCIA, A.; KALVA, H. Cloud transcoding for mobile video content delivery. In: IEEE. *2011 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.], 2011. p. 379–380.

GERMAN, R. *Performance analysis of communication systems with non-Markovian stochastic Petri nets*. [S.l.]: John Wiley & Sons, Inc., 2000.

GRAY, C.; CHERITON, D. BOOK. *Leases: An efficient fault-tolerant mechanism for distributed file cache consistency*. dl.acm.org, 1989. 678 cites: https://scholar.google.com/scholar?cites=16253568565428613985&as$_s$dt = 2005&sciodt = 0, 5&hl = en.Availableat :<https : //dl.acm.org/citation.cfm?id = 74870>.

GROUP, P. et al. Php hypertext preprocessor (2008). *URL http://www. php. net*, 2007.

HABIBI, A. Hybrid coding of pictorial data. *IEEE Transactions on Communications*, IEEE, v. 22, n. 5, p. 614–624, 1974.

HAMBY, D. A review of techniques for parameter sensitivity analysis of environmental models. *Environmental monitoring and assessment*, Springer, v. 32, n. 2, p. 135–154, 1994.

HEIMANN, D. I.; MITTAL, N.; TRIVEDI, K. S. Availability and reliability modeling for computer systems. In: *Advances in Computers*. [S.l.]: Elsevier, 1990. v. 31, p. 175–233.

HOFFMAN, F. O.; MILLER, C. W. *Uncertainties in environmental radiological assessment models and their implications*. [S.l.], 1983.

KAFHALI, S. E.; SALAH, K. Stochastic modelling and analysis of cloud computing data center. In: IEEE. *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*. [S.l.], 2017. p. 122–126.

KANAWATI, G. A.; KANAWATI, N. A.; ABRAHAM, J. A. Ferrari: A tool for the validation of system dependability properties. In: IEEE. *[1992] Digest of Papers. FTCS-22: The Twenty-Second International Symposium on Fault-Tolerant Computing*. [S.l.], 1992. p. 336–344.

KAROLEWICZ, K.; BĘBEN, A. Cloud-based adaptive video streaming: Content storage vs. transcoding optimization methods. In: IEEE. *2017 IEEE Symposium on Computers and Communications (ISCC)*. [S.l.], 2017. p. 523–528.

KEESEE, W. *A Method of Determining a Confidence Interval for Availability*. [S.l.], 1965.

KHEDHER, O.; CHOWDHURY, C. D. *Mastering OpenStack*. [S.l.]: Packt Publishing Ltd, 2017.

KLEINROCK, L. *Queueing systems*. [S.l.], 1975.

LIAO, W.; LI, V. O. The split and merge protocol for interactive video-on-demand. *IEEE multimedia*, IEEE, v. 4, n. 4, p. 51–62, 1997.

MACIEL, P. R. et al. Dependability modeling. In: *Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions*. [S.l.]: IGI Global, 2012. p. 53–97.

MARSAN, M. A.; CONTE, G.; BALBO, G. A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, v. 2, n. 2, p. 93–122, 1984.

MARTIN, F. et al. Modelling an adaptive-rate video-streaming service using markov-rewards models. In: IEEE. *First International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks*. [S.l.], 2004. p. 92–99.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MELO, C. et al. Availability models for synchronization server infrastructure. In: IEEE. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. [S.l.], 2016. p. 003658–003663.

MELO, C. et al. Capacity-oriented availability model for resources estimation on private cloud infrastructure. In: IEEE. *Dependable Computing (PRDC), 2017 IEEE 22nd Pacific Rim International Symposium on*. [S.l.], 2017. p. 255–260.

MELO, R. et al. Vod in eucalyptus platform: Availability modeling and sensibility analysis. In: IEEE. *10th International Conference on Network and Service Management (CNSM) and Workshop*. [S.l.], 2014. p. 288–291.

MELO, R. et al. Video on demand hosted in private cloud: Availability modeling and sensitivity analysis. In: IEEE. *Dependable Systems and Networks Workshops (DSN-W), 2015 IEEE International Conference on*. [S.l.], 2015. p. 12–18.

MERRITT, L.; VANAM, R. x264: A high performance h. 264/avc encoder. *online] http://neuron2. net/library/avc/overview_x264_v8_5. pdf*, 2006.

MINITAB, I. *MINITAB reference manual*. [S.l.]: Minitab, 1991.

MOLLOY, M. K. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. Phd Thesis (PhD Thesis), 1981. AAI8201138.

MURATA, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, IEEE, v. 77, n. 4, p. 541–580, 1989.

MYSQL, A. *MySQL*. 2001.

MYSQL, A. Mysql database server. *Internet WWW page, at URL: http://www. mysql. com (last accessed/1/00)*, 2004.

NEDELCU, C. *Nginx HTTP Server: Adopt Nginx for Your Web Applications to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever*. [S.l.]: Packt Publishing Ltd, 2010.

NURMI, D. et al. The eucalyptus open-source cloud-computing system. In: IEEE COMPUTER SOCIETY. *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. [S.l.], 2009. p. 124–131.

PEREIRA, R. et al. An architecture for distributed high performance video processing in the cloud. In: IEEE. *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. [S.l.], 2010. p. 482–489.

RANDELL, B. System structure for software fault tolerance. *Ieee transactions on software engineering*, ieeexplore.ieee.org, 1975. 2428 cites: https://scholar.google.com/scholar?cites=15028600751407867618as$_sdt$ = 2005&$sciodt$ = 0,5&hl = $en.Availableat$:<$https$: //$ieeexplore.ieee.org/abstract/document/6312842/$>.

REESE, W. Nginx: the high-performance web server and reverse proxy. *Linux Journal*, Belltown Media, v. 2008, n. 173, p. 2, 2008.

REIS, G. et al. Swift: Software implemented fault tolerance. *Proceedings of the ...*, dl.acm.org, 2005. Available at: <https://dl.acm.org/citation.cfm?id=1048991>.

SANNER, M. F. et al. Python: a programming language for software integration and development. *J Mol Graph Model*, v. 17, n. 1, p. 57–61, 1999.

SCHAFFER, H. E. X as a service, cloud computing, and the need for good judgment. *IT professional*, IEEE, v. 11, n. 5, p. 4–5, 2009.

SCHOTT, J. *Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization.* apps.dtic.mil, 1995. 1000 cites: https://scholar.google.com/scholar?cites=18065763319098608279&as$_s$*dt* = 2005&*sciodt* = 0, 5&*hl* = *en.Availableat* :<*https* : *//apps.dtic.mil/docs/citations/ADA*296310>.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. Comparison of multiple iaas cloud platform solutions. In: *Proceedings of the 7th WSEAS International Conference on Computer Engineering and Applications,(Milan-CEA 13). ISBN*. [S.l.: s.n.], 2012. p. 978–1.

SEFRAOUI, O.; AISSAOUI, M.; ELEULDJ, M. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, Foundation of Computer Science, v. 55, n. 3, 2012.

SHANNON, C. E. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec*, v. 4, n. 142-163, p. 1, 1959.

SHVACHKO, K. et al. The hadoop distributed file system. In: *MSST*. [S.l.: s.n.], 2010. v. 10, p. 1–10.

SILVA, B. et al. Astro: An integrated environment for dependability and sustainability evaluation. *Sustainable computing: informatics and systems*, Elsevier, v. 3, n. 1, p. 1–17, 2013.

SILVA, B. et al. Mercury: An integrated environment for performance and dependability evaluation of general systems. In: *Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference, DSN*. [S.l.: s.n.], 2015.

SOUZA, D. et al. Eucabomber: Experimental evaluation of availability in eucalyptus private clouds. In: IEEE. *2013 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.], 2013. p. 4080–4085.

SOUZA, D. et al. A tool for automatic dependability test in eucalyptus cloud computing infrastructures. *Computer and Information Science*, Canadian Center of Science and Education, v. 6, n. 3, p. 57, 2013.

SULLIVAN, G. J.; WIEGAND, T. Video compression-from concepts to the h. 264/avc standard. *Proceedings of the IEEE*, IEEE, v. 93, n. 1, p. 18–31, 2005.

TRIVEDI, K. S. et al. Reliability analysis techniques explored through a communication network example. Citeseer, 1996.

TSAI, T.; IYER, R. Ftape-a fault injection tool to measure fault tolerance. In: *10th Computing in Aerospace Conference*. [S.l.: s.n.], 1995. p. 1041.

ZHAO, X. et al. Computation offloading for h. 264 video encoder on mobile devices. In: IEEE. *Computational Engineering in Systems Applications, IMACS Multiconference on.* [S.l.], 2006. v. 2, p. 1426–1430.

8

Apêndice

8.1 MERCURY SCRIPT

```
################################################
################################################
#####Cdigo referente ao Modelo RBD FRONTEND#####
################################################
################################################

t = 100;
RBD ModelFE{
   block hw( MTTF = 8760.0, MTTR = 1.666666667);
   block os( MTTF = 2890.0, MTTR = 0.5);
   block nginx( MTTF = 788.4000000000001, MTTR = 0.5);
   series s0(hw, os, nginx );
   top s0;
   metric av = availability;
   metric rel = reliability( time = t );
   metric mttf = mttf;
   metric mttr = mttr;
}
main{
```

```
   av = solve(ModelFE, av);
   rel = solve(ModelFE, rel);
   mttf = solve(ModelFE, mttf);
   mttr = solve(ModelFE, mttr);
   println("Availability FE: " .. av );
   println("Reliability FE: " .. rel );
   println("Mean time to failure FE: " .. mttf );
   println("Mean time to repair FE: " .. mttr );
}




################################################
################################################
#####Cdigo referente ao Modelo RBD MVM##########
################################################
################################################
t = 100;
RBD ModelMVM{
   block vm( MTTF = 2880.0, MTTR = 0.02);
   block so( MTTF = 2890.0, MTTR = 0.02);
   block python( MTTF = 788.4000000000001, MTTR = 0.5);
   block database( MTTF = 788.4000000000001, MTTR = 0.5);
   block ffmpeg( MTTF = 336.0, MTTR = 0.5);
   block php( MTTF = 788.4000000000001, MTTR = 0.5);
   series s0(vm, so, python, database, ffmpeg, php );
   top s0;
   metric av = availability;
   metric rel = reliability( time = t );
   metric mttf = mttf;
   metric mttr = mttr;
}
main{
   av = solve(ModelMVM, av);
   rel = solve(ModelMVM, rel);
   mttf = solve(ModelMVM, mttf);
   mttr = solve(ModelMVM, mttr);
   println("Availability MVM: " .. av );
   println("Reliability MVM: " .. rel );
   println("Mean time to failure MVM: " .. mttf );
```

```
   println("Mean time to repair MVM: " .. mttr );
}


################################################
################################################
#####Cdigo  referente ao Modelo RBD MAIN NODE####
################################################
################################################

t = 100;
RBD ModelMN{
   block hw( MTTF = 8760.0, MTTR = 1.666666667);
   block os( MTTF = 2880.0, MTTR = 0.5);
   block neutron( MTTF = 788.4000000000001, MTTR = 1.0);
   block kvm( MTTF = 2880.0, MTTR = 1.0);
   block nova( MTTF = 788.4000000000001, MTTR = 1.0);
   block cinder( MTTF = 788.4000000000001, MTTR = 1.0);
   series s0(hw, os, neutron, kvm, nova, cinder );
   top s0;
   metric av = availability;
   metric rel = reliability( time = t );
   metric mttf = mttf;
   metric mttr = mttr;
}
main{
   av = solve(ModelMN, av);
   rel = solve(ModelMN, rel);
   mttf = solve(ModelMN, mttf);
   mttr = solve(ModelMN, mttr);
   println("Availability MN: " .. av );
   println("Reliability MN: " .. rel );
   println("Mean time to failure MN: " .. mttf );
   println("Mean time to repair MN: " .. mttr );
}


################################################
################################################
 Cdigo  referente ao Modelo RBD GENERAL STRUCTURE#
################################################
################################################
```

```
t = 100;
RBD ModelGS{
   block fe( MTTF = 578.5136473276605, MTTR = 0.5771994341877017);
   block mvm( MTTF = 133.78627125505434, MTTR = 0.45602851693669133);
   block mn( MTTF = 216.7422680412349, MTTR = 0.9805344071379704);
   series s0(fe, mvm, mn );
   top s0;
   metric av = availability;
   metric rel = reliability( time = t );
   metric mttf = mttf;
   metric mttr = mttr;
}
main{
   av = solve(ModelGS, av);
   rel = solve(ModelGS, rel);
   mttf = solve(ModelGS, mttf);
   mttr = solve(ModelGS, mttr);
   println("Availability GS: " .. av );
   println("Reliability GS: " .. rel );
   println("Mean time to failure GS: " .. mttf );
   println("Mean time to repair GS: " .. mttr );
}

##################################################
##################################################
#####Cdigo referente ao Modelo SPN COM 1 N e acelerao
##################################################
##################################################
MTTFVM = 13.378627125505434;
MTTRVM = 0.0833333;
MTTFNO = 21.674226804123487;
MTTRNO = 2;
tokens=0;
SPN Model{

   place P0( tokens= tokens );
   place P1;
   place P2( tokens= 1 );
   place P3;
```

```
   immediateTransition TI0(
      inputs = [P0],
      outputs = [P1],
      guardExpression = #P2==0
   );

   timedTransition TE0(
      inputs = [P1],
      outputs = [P0],
      guardExpression = #P3==0,
      delay = MTTRVM
   );

   timedTransition TE1(
      inputs = [P0],
      outputs = [P1],
      delay = MTTFVM
   );

   timedTransition TE2(
      inputs = [P3],
      outputs = [P2],
      delay = MTTRNO
   );

   timedTransition TE3(
      inputs = [P2],
      outputs = [P3],
      delay = MTTFNO
   );

   metric disp = stationaryAnalysis( expression = "P{(#P0>0)AND(#P2>0)}" );
   metric coa1 = stationaryAnalysis( expression = "P{(#P0)=1}" );
   metric coa2 = stationaryAnalysis( expression = "P{(#P0)=2}*2" );
   metric coa3 = stationaryAnalysis( expression = "P{(#P0)=3}*3" );
   metric coa4 = stationaryAnalysis( expression = "P{(#P0)=4}*4" );
   metric coa5 = stationaryAnalysis( expression = "P{(#P0)=5}*5" );
   metric coa6 = stationaryAnalysis( expression = "P{(#P0)=6}*6" );
   metric coa7 = stationaryAnalysis( expression = "P{(#P0)=7}*7" );
}

main {
```

```
 tokens=1;
disp = solve( Model,disp );
 print("DISPONIBILIDADE 1 N 1 VM: ");
 println(disp);
tokens=7;
 disp = solve( Model,disp );
 print("DISPONIBILIDADE 1 N 7 VMS: ");
 println(disp);
 coa1 = solve( Model,coa1 );
 coa2 = solve( Model,coa2 );
 coa3 = solve( Model,coa3 );
 coa4 = solve( Model,coa4 );
 coa5 = solve( Model,coa5 );
 coa6 = solve( Model,coa6 );
 coa7 = solve( Model,coa7 );
 print("COA 1 N 7 VMS: ");
 println((coa1+coa2+coa3+coa4+coa5+coa6+coa7)/7);
 print("CAPACIDADE REAL 1 N 7 VMS: ");
 println((coa1+coa2+coa3+coa4+coa5+coa6+coa7));
 tokens=1;
 percentageDifference(
         model_ = "Model",
         metric_ = "disp",
         samplingPoints = 20,
         parameters = (

       MTTFVM = [ MTTFVM-(MTTFVM*0.5), MTTFVM+(MTTFVM*0.5) ],
       MTTRVM = [ MTTRVM-(MTTRVM*0.5), MTTRVM+(MTTRVM*0.5) ],
       MTTFNO = [ MTTFNO-(MTTFNO*0.5), MTTFNO+(MTTFNO*0.5) ],
       MTTRNO = [ MTTRNO-(MTTRNO*0.5), MTTRNO+(MTTRNO*0.5) ]
         ),

         output = (
             type = "swing",
             yLabel = "Steady-state availability",
             baselineValue = disp
         )
  );

}
```

```
################################################
################################################
#####Cdigo  referente ao Modelo SPN COM 1 N#####
################################################
################################################


MTTFVM = 133.78627125505434;
MTTRVM = 0.0833333;
MTTFNO = 216.74226804123487;
MTTRNO = 2;
tokens=0;
SPN Model{

   place P0( tokens= tokens );
   place P1;
   place P2( tokens= 1 );
   place P3;


   immediateTransition TI0(
      inputs = [P0],
      outputs = [P1],
      guardExpression = #P2==0
   );

   timedTransition TE0(
      inputs = [P1],
      outputs = [P0],
      guardExpression = #P3==0,
      delay = MTTRVM
   );

   timedTransition TE1(
      inputs = [P0],
```

```
      outputs = [P1],
      delay = MTTFVM
   );

   timedTransition TE2(
      inputs = [P3],
      outputs = [P2],
      delay = MTTRNO
   );

   timedTransition TE3(
      inputs = [P2],
      outputs = [P3],
      delay = MTTFNO
   );

   metric disp = stationaryAnalysis( expression = "P{(#P0>0)AND(#P2>0)}" );
   metric coa1 = stationaryAnalysis( expression = "P{(#P0)=1}" );
   metric coa2 = stationaryAnalysis( expression = "P{(#P0)=2}*2" );
   metric coa3 = stationaryAnalysis( expression = "P{(#P0)=3}*3" );
   metric coa4 = stationaryAnalysis( expression = "P{(#P0)=4}*4" );
   metric coa5 = stationaryAnalysis( expression = "P{(#P0)=5}*5" );
   metric coa6 = stationaryAnalysis( expression = "P{(#P0)=6}*6" );
   metric coa7 = stationaryAnalysis( expression = "P{(#P0)=7}*7" );
}

main {
   tokens=1;
  disp = solve( Model,disp );
   print("DISPONIBILIDADE 1 N 1 VM: ");
   println(disp);
  tokens=7;
   disp = solve( Model,disp );
   print("DISPONIBILIDADE 1 N 7 VMS: ");
   println(disp);
   coa1 = solve( Model,coa1 );
   coa2 = solve( Model,coa2 );
   coa3 = solve( Model,coa3 );
   coa4 = solve( Model,coa4 );
   coa5 = solve( Model,coa5 );
   coa6 = solve( Model,coa6 );
   coa7 = solve( Model,coa7 );
```

```
   print("COA 1 N 7 VMS: ");
   println((coa1+coa2+coa3+coa4+coa5+coa6+coa7)/7);
   print("CAPACIDADE REAL 1 N 7 VMS: ");
   println((coa1+coa2+coa3+coa4+coa5+coa6+coa7));
   tokens=1;
   percentageDifference(
           model_ = "Model",
           metric_ = "disp",
           samplingPoints = 20,
           parameters = (

         MTTFVM = [ MTTFVM-(MTTFVM*0.5), MTTFVM+(MTTFVM*0.5) ],
         MTTRVM = [ MTTRVM-(MTTRVM*0.5), MTTRVM+(MTTRVM*0.5) ],
         MTTFNO = [ MTTFNO-(MTTFNO*0.5), MTTFNO+(MTTFNO*0.5) ],
         MTTRNO = [ MTTRNO-(MTTRNO*0.5), MTTRNO+(MTTRNO*0.5) ]
           ),

           output = (
               type = "swing",
               yLabel = "Steady-state availability",
               baselineValue = disp
           )
    );

}



################################################
################################################
#####Cdigo  referente ao Modelo SPN COM 2 NS####
################################################
################################################

MTTFVM = 133.78627125505434;
MTTRVM = 0.0833333;
MTTFNO = 216.74226804123487;
MTTRNO = 3;
tokens=0;
tokens2=0;
SPN Model{
```

```
place N1_OFF;
place N1_ON( tokens= 1 );
place N1_VM_OFF;
place N1_VM_ON( tokens= tokens );
place N2_OFF;
place N2_ON( tokens= 1 );
place N2_VM_OFF;
place N2_VM_ON( tokens= tokens2 );
place VM_TRANSITION_N1_N2;
place VM_TRANSITION_N2_N1;


immediateTransition NODE1_DOWN_VM_DOWN(
   enablingFunction = "#N1_ON=0",
   inputs = [N1_VM_ON],
   outputs = [N1_VM_OFF]
);

immediateTransition NODE2_DOWN_VM_DOWN(
   enablingFunction = "#N2_ON=0",
   inputs = [N2_VM_ON],
   outputs = [N2_VM_OFF]
);

immediateTransition VM_DOWN_TRANSITION_N1_N2(
   enablingFunction = "#N2_ON=1",
   inputs = [N1_VM_OFF],
   outputs = [VM_TRANSITION_N1_N2]
);

immediateTransition VM_DOWN_TRANSITION_N2_N1(
   enablingFunction = "#N1_ON=1",
   inputs = [N2_VM_OFF],
   outputs = [VM_TRANSITION_N2_N1]
);

timedTransition MTTFNO1(
   inputs = [N1_ON],
   outputs = [N1_OFF],
   delay = MTTFNO
);
```

```
timedTransition MTTFNO2(
   inputs = [N2_ON],
   outputs = [N2_OFF],
   delay = MTTFNO
);

timedTransition MTTFVM1(
   inputs = [N1_VM_ON],
   outputs = [N1_VM_OFF],
   delay = MTTFVM
);

timedTransition MTTFVM2(
   inputs = [N2_VM_ON],
   outputs = [N2_VM_OFF],
   delay = MTTFVM
);

timedTransition MTTRNO1(
   inputs = [N1_OFF],
   outputs = [N1_ON],
   delay = MTTRNO
);

timedTransition MTTRNO2(
   inputs = [N2_OFF],
   outputs = [N2_ON],
   delay = MTTRNO
);

timedTransition MTTRVM1(
   inputs = [N1_VM_OFF],
   outputs = [N1_VM_ON],
   delay = MTTRVM,
   guardExpression = #N1_ON==1
);

timedTransition MTTRVM2(
   inputs = [N2_VM_OFF],
   outputs = [N2_VM_ON],
   delay = MTTRVM,
   guardExpression = #N2_ON==1
```

```
   );

   timedTransition TRANSITION_N1_N2(
      inputs = [VM_TRANSITION_N1_N2],
      outputs = [N2_VM_ON],
      delay = MTTRVM
   );

   timedTransition TRANSITION_N2_N1(
      inputs = [VM_TRANSITION_N2_N1],
      outputs = [N1_VM_ON],
      delay = MTTRVM
   );

   metric disp = stationaryAnalysis( expression =
       "P{(#N1_VM_ON>0)OR(#N2_VM_ON>0)}" );
   metric coa1 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=1}*1" );
   metric coa2 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=2}*2" );
   metric coa3 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=3}*3" );
   metric coa4 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=4}*4" );
   metric coa5 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=5}*5" );
   metric coa6 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=6}*6" );
   metric coa7 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=7}*7" );
   metric coa8 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=8}*8" );
   metric coa9 = stationaryAnalysis( expression = "P{(#N1_VM_ON+#N2_VM_ON)=9}*9" );
   metric coa10 = stationaryAnalysis( expression =
       "P{(#N1_VM_ON+#N2_VM_ON)=10}*10" );
   metric coa11 = stationaryAnalysis( expression =
       "P{(#N1_VM_ON+#N2_VM_ON)=11}*11" );
   metric coa12 = stationaryAnalysis( expression =
       "P{(#N1_VM_ON+#N2_VM_ON)=12}*12" );
   metric coa13 = stationaryAnalysis( expression =
       "P{(#N1_VM_ON+#N2_VM_ON)=13}*13" );
   metric coa14 = stationaryAnalysis( expression =
       "P{(#N1_VM_ON+#N2_VM_ON)=14}*14" );
}

main {
   tokens=7;
   tokens2=7;
   coa1 = solve( Model,coa1 );
   coa2 = solve( Model,coa2 );
```

```
coa3 = solve( Model,coa3 );
coa4 = solve( Model,coa4 );
coa5 = solve( Model,coa5 );
coa6 = solve( Model,coa6 );
coa7 = solve( Model,coa7 );
coa8 = solve( Model,coa8 );
coa9 = solve( Model,coa9 );
coa10 = solve( Model,coa10 );
coa11 = solve( Model,coa11 );
coa12 = solve( Model,coa12 );
coa13 = solve( Model,coa13 );
coa14 = solve( Model,coa14 );
tokens=1;
tokens2=0;
disp = solve( Model,disp );
print("DISPONIBILIDADE 2 NS 1 VM: ");
println(disp);
tokens=7;
tokens2=7;
disp = solve( Model,disp );
print("DISPONIBILIDADE 2 NS 7 VMS: ");
println(disp);
print("COA 2 NS 7 VMS: ");
println((coa1+coa2+coa3+coa4+coa5+coa6+coa7+coa8+coa9+coa10+coa11+coa12+coa13+coa14)/14);
print("CAPACIDADE REAL 2 NS 7 VMS: ");
println((coa1+coa2+coa3+coa4+coa5+coa6+coa7+coa8+coa9+coa10+coa11+coa12+coa13+coa14));
tokens=1;
tokens2=0;
percentageDifference(
              model_ = "Model",
              metric_ = "disp",
              samplingPoints = 20,
              parameters = (

     MTTFVM = [ MTTFVM-(MTTFVM*0.5), MTTFVM+(MTTFVM*0.5) ],
     MTTRVM = [ MTTRVM-(MTTRVM*0.5), MTTRVM+(MTTRVM*0.5) ],
     MTTFNO = [ MTTFNO-(MTTFNO*0.5), MTTFNO+(MTTFNO*0.5) ],
     MTTRNO = [ MTTRNO-(MTTRNO*0.5), MTTRNO+(MTTRNO*0.5) ]
              ),

              output = (
                    type = "swing",
```

```
                    yLabel = "Steady-state availability",
                    baselineValue = disp
                )
    );

}
```

```
################################################
################################################
#### Cdigo  referente ao Modelo SPN COM 3 NS#####
################################################
################################################
```

```
MTTFVM = 133.78627125505434;
MTTRVM = 0.0833333;
MTTFNO = 216.74226804123487;
MTTRNO = 2.5;
tokens=0;
tokens2=0;
SPN Model{

   place N1_OFF;
   place N1_ON( tokens= 1 );
   place N1_VM_OFF;
   place N1_VM_ON( tokens= tokens );
   place N2_OFF;
   place N2_ON( tokens= 1 );
   place N2_VM_OFF;
   place N2_VM_ON( tokens= tokens2 );
   place N3_OFF;
   place N3_ON( tokens= 1 );
   place N3_VM_OFF;
   place N3_VM_ON( tokens= tokens2 );
   place VM_TRANSITION_N1_N2;
```

```
place VM_TRANSITION_N1_N3;
place VM_TRANSITION_N2_N1;
place VM_TRANSITION_N2_N3;
place VM_TRANSITION_N3_N1;
place VM_TRANSITION_N3_N2;


immediateTransition NODE1_DOWN_VM_DOWN(
   enablingFunction = "#N1_ON=0",
   inputs = [N1_VM_ON],
   outputs = [N1_VM_OFF]
);

immediateTransition NODE2_DOWN_VM_DOWN(
   enablingFunction = "#N2_ON=0",
   inputs = [N2_VM_ON],
   outputs = [N2_VM_OFF]
);

immediateTransition TI0(
   enablingFunction = "#N3_ON>0",
   inputs = [N1_VM_OFF],
   outputs = [VM_TRANSITION_N1_N3]
);

immediateTransition TI1(
   enablingFunction = "#N3_ON>0",
   inputs = [N2_VM_OFF],
   outputs = [VM_TRANSITION_N2_N3]
);

immediateTransition TI2(
   enablingFunction = "#N3_ON=0",
   inputs = [N3_VM_ON],
   outputs = [N3_VM_OFF]
);

immediateTransition VM_DOWN_TRANSITION_N1_N2(
   enablingFunction = "#N2_ON=1",
   inputs = [N1_VM_OFF],
   outputs = [VM_TRANSITION_N1_N2]
);
```

```
immediateTransition VM_DOWN_TRANSITION_N2_N1(
   enablingFunction = "#N1_ON=1",
   inputs = [N2_VM_OFF],
   outputs = [VM_TRANSITION_N2_N1]
);

immediateTransition VM_DOWN_TRANSITION_N3_N1(
   enablingFunction = "#N1_ON=1",
   inputs = [N3_VM_OFF],
   outputs = [VM_TRANSITION_N3_N1]
);

immediateTransition VM_DOWN_TRANSITION_N3_N2(
   enablingFunction = "#N2_ON=1",
   inputs = [N3_VM_OFF],
   outputs = [VM_TRANSITION_N3_N2]
);

timedTransition MTTFNO1(
   inputs = [N1_ON],
   outputs = [N1_OFF],
   delay = MTTFNO
);

timedTransition MTTFNO2(
   inputs = [N2_ON],
   outputs = [N2_OFF],
   delay = MTTFNO
);

timedTransition MTTFNO3(
   inputs = [N3_ON],
   outputs = [N3_OFF],
   delay = MTTFNO
);

timedTransition MTTFVM1(
   inputs = [N1_VM_ON],
   outputs = [N1_VM_OFF],
   delay = MTTFVM
);
```

```
timedTransition MTTFVM2(
   inputs = [N2_VM_ON],
   outputs = [N2_VM_OFF],
   delay = MTTFVM
);

timedTransition MTTFVM3(
   inputs = [N3_VM_ON],
   outputs = [N3_VM_OFF],
   delay = MTTFVM
);

timedTransition MTTRNO1(
   inputs = [N1_OFF],
   outputs = [N1_ON],
   delay = MTTRNO
);

timedTransition MTTRNO2(
   inputs = [N2_OFF],
   outputs = [N2_ON],
   delay = MTTRNO
);

timedTransition MTTRNO3(
   inputs = [N3_OFF],
   outputs = [N3_ON],
   delay = MTTRNO
);

timedTransition MTTRVM1(
   inputs = [N1_VM_OFF],
   outputs = [N1_VM_ON],
   delay = MTTRVM,
   guardExpression = #N1_ON==1
);

timedTransition MTTRVM2(
   inputs = [N2_VM_OFF],
   outputs = [N2_VM_ON],
   delay = MTTRVM,
```

```
   guardExpression = #N2_ON==1
);

timedTransition MTTRVM3(
   inputs = [N3_VM_OFF],
   outputs = [N3_VM_ON],
   delay = MTTRVM,
   guardExpression = #N3_ON>0
);

timedTransition TRANSITION_N1_N2(
   inputs = [VM_TRANSITION_N1_N2],
   outputs = [N2_VM_ON],
   delay = MTTRVM
);

timedTransition TRANSITION_N1_N3(
   inputs = [VM_TRANSITION_N1_N3],
   outputs = [N3_VM_ON],
   delay = MTTRVM
);

timedTransition TRANSITION_N2_N1(
   inputs = [VM_TRANSITION_N2_N1],
   outputs = [N1_VM_ON],
   delay = MTTRVM
);

timedTransition TRANSITION_N2_N3(
   inputs = [VM_TRANSITION_N2_N3],
   outputs = [N3_VM_ON],
   delay = MTTRVM
);

timedTransition TRANSITION_N3_N1(
   inputs = [VM_TRANSITION_N3_N1],
   outputs = [N1_VM_ON],
   delay = MTTRVM
);

timedTransition TRANSITION_N3_N2(
   inputs = [VM_TRANSITION_N3_N2],
```

```
   outputs = [N2_VM_ON],
   delay = MTTRVM
);

metric disp = stationaryAnalysis( expression =
    "P{(#N1_VM_ON>0)OR(#N2_VM_ON>0)OR(#N3_VM_ON>0)}" );
metric disp2 = stationarySimulation( expression =
    "P{(#N1_VM_ON>0)OR(#N2_VM_ON>0)OR(#N3_VM_ON>0)}" );
metric coa1 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=1}*1" );
metric coa2 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=2}*2" );
metric coa3 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=3}*3" );
metric coa4 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=4}*4" );
metric coa5 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=5}*5" );
metric coa6 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=6}*6" );
metric coa7 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=7}*7" );
metric coa8 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=8}*8" );
metric coa9 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=9}*9" );
metric coa10 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=10}*10" );
metric coa11 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=11}*11" );
metric coa12 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=12}*12" );
metric coa13 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=13}*13" );
metric coa14 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=14}*14" );
metric coa15 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=15}*15" );
metric coa16 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=16}*16" );
metric coa17 = stationarySimulation( expression =
    "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=17}*17" );
```

```
   metric coa18 = stationarySimulation( expression =
       "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=18}*18" );
   metric coa19 = stationarySimulation( expression =
       "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=19}*19" );
   metric coa20 = stationarySimulation( expression =
       "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=20}*20" );
   metric coa21 = stationarySimulation( expression =
       "P{(#N1_VM_ON+#N2_VM_ON+#N3_VM_ON)=21}*21" );
}

main {
   tokens=7;
   tokens2=7;
   coa1 = solve( Model,coa1 );
   coa2 = solve( Model,coa2 );
   coa3 = solve( Model,coa3 );
   coa4 = solve( Model,coa4 );
   coa5 = solve( Model,coa5 );
   coa6 = solve( Model,coa6 );
   coa7 = solve( Model,coa7 );
   coa8 = solve( Model,coa8 );
   coa9 = solve( Model,coa9 );
   coa10 = solve( Model,coa10 );
   coa11 = solve( Model,coa11 );
   coa12 = solve( Model,coa12 );
   coa13 = solve( Model,coa13 );
   coa14 = solve( Model,coa14 );
   coa15 = solve( Model,coa15 );
   coa16 = solve( Model,coa16 );
   coa17 = solve( Model,coa17 );
   coa18 = solve( Model,coa18 );
   coa19 = solve( Model,coa19 );
   coa20 = solve( Model,coa20 );
   coa21 = solve( Model,coa21 );
   tokens=1;
   tokens2=0;
   disp = solve( Model,disp );
   print("DISPONIBILIDADE 3 NS 1 VM: ");
   println(disp);
   tokens=7;
   tokens2=7;
   disp = solve( Model,disp2 );
```

```
   print("DISPONIBILIDADE 3 NS 7 VMS: ");
   println(disp);
   print("COA 3 NS 7 VMS: ");
   println((coa1+coa2+coa3+coa4+coa5+coa6+coa7+coa8+coa9+coa10+coa11+coa12+coa13+coa14+coa15+c
   print("CAPACIDADE REAL 3 NS 7 VMS: ");
   println((coa1+coa2+coa3+coa4+coa5+coa6+coa7+coa8+coa9+coa10+coa11+coa12+coa13+coa14+coa15+c
   tokens=1;
   tokens2=0;
   percentageDifference(
                model_ = "Model",
                metric_ = "disp",
                samplingPoints = 20,
                parameters = (

        MTTFVM = [ MTTFVM-(MTTFVM*0.5), MTTFVM+(MTTFVM*0.5) ],
        MTTRVM = [ MTTRVM-(MTTRVM*0.5), MTTRVM+(MTTRVM*0.5) ],
        MTTFNO = [ MTTFNO-(MTTFNO*0.5), MTTFNO+(MTTFNO*0.5) ],
        MTTRNO = [ MTTRNO-(MTTRNO*0.5), MTTRNO+(MTTRNO*0.5) ]
                ),

                output = (
                      type = "swing",
                      yLabel = "Steady-state availability",
                      baselineValue = disp
                )
     );

}
```

8.2 CÓDIGO MODCS INJECTOR

```
from threading import Thread
import subprocess
import thread
import time
import datetime
import numpy

from keystoneauth1.identity import v2
from keystoneauth1 import session
from keystoneclient.v2_0 import client
```

```
token = '012345SECRET99TOKEN012345'
endpoint = 'http://192.168.206.130:35357/v2.0'
auth = v2.Token(auth_url=endpoint, token=token)
sess = session.Session(auth=auth)
keystone = client.Client(session=sess)

# Define a function for the thread

def injetar_reparo_falha(tamanhofalha, escalafalha, tamanhoreparo, escalareparo,
    comandofalha, comandoreparo, nomeservico):
   while 1:
      try:
         tempofalha = numpy.random.exponential(tamanhofalha, escalafalha)
         print "Proxima falha do servico "+str(nomeservico)+ ": " + str(tempofalha)
         time.sleep(tempofalha)
         sinalizar("injecao de falha")
         falhabkp=comandofalha
         if (comandofalha.find("$exp")>-1):
            comandofalha = comandofalha.replace("$exp",str(tempofalha))
         output =
             subprocess.check_output['bash','-c',comandofalha.replace('"','')])
             #subprocess.Popen(comandofalha, shell=True,
             stdout=subprocess.PIPE).stdout.read().rstrip()
         comandofalha = falhabkp
         temporeparo = numpy.random.exponential(tamanhoreparo, escalareparo)
         print "Proximo reparo: "+str(temporeparo)
         time.sleep(temporeparo)
         reparobkp=comandoreparo
         if (comandoreparo.find("$exp")>-1):
            comandoreparo = comandoreparo.replace("$exp",str(temporeparo))
         output =
             subprocess.check_output['bash','-c',comandoreparo.replace('"','')])
             #subprocess.Popen(comandoreparo, shell=True,
             stdout=subprocess.PIPE).stdout.read().rstrip()#
         comandorepado=reparobkp
         sinalizar("injecao de reparo")
      except Exception as e:
         print ">>>"+str(e)

def monitorar(servicon,checagemservico):
   print "monitorando"
   servico = 1
```

```
   while 1:
      try:
         import subprocess
         output =
             subprocess.check_output(['bash','-c',checagemservico.replace("'",'')]).rstrip()
             #subprocess.Popen(checagemservico.replace('"',''), shell=True,
             stdout=subprocess.PIPE).stdout.read().rstrip()
         #print "output: " + str(output)
         if servico==1 and str(output)=='0':
            print "entrou 1"
            servico=0
            with open("monitoramento.txt", "a") as myfile:
               data = datetime.datetime.now()
               print str(data) +servicon+" D"
               myfile.write(str(data) + servicon+" D"+"\n")
         elif servico==0 and str(output)=='1':
            print "entrou 2"
            servico=1
            with open("monitoramento.txt", "a") as myfile:
               data = datetime.datetime.now()
               print str(data) +servicon+" U"
               myfile.write(str(data) + servicon+"U"+"\n")
      except Exception as e:
         print str(e)

def sinalizar(string):
   with open("monitoramento.txt", "a") as myfile:
      data = datetime.datetime.now()
      myfile.write(str(data) + " ### "+string+"\n")



print """
    __  _______     __            ____       _          __
   /  |/  / __ \____/ /_________  / _/___    (_)__  _____/ /_____ _____
  / /|_/ / / / / __ / ___/ ___/ / // __ \ / / _ \/ ___/ __/ __ \/ ___/
 / /  / / /_/ / /_/ / /__(__ ) _/ // / / / / / __/ /__/ /_/ /_/ / /
/_/  /_/\____/\__,_/\___/____/ /___/_/ /_/_/ /\___/\___/\__/\____/_/
                                    /___/                          """

def captura_inputs():
   try:
```

```
      nome = ""
      while nome != "k":
         print "Digite o nome do servico"
         nome = raw_input()#"mysql" #
         if nome=="k":
            continue
         print "Digite o comando para verificar o servico"
         comandoverificarservico = raw_input()
         print "Digite o comando para derrubar o servico"
         comandoderrubarservico = raw_input()
         print "Digite o comando para levantar o servico"
         comandolevantarservico = raw_input()
         print "Digite a taxa de falha do servico"
         taxadefalha = raw_input()
         print "Digite a taxa de reparo do servico"
         taxadereparo = raw_input()
         try:
            t = Thread(target=monitorar, args=(nome, comandoverificarservico))
            #t.daemon = True
            t.start()
            t2 = Thread(target=injetar_reparo_falha,
                args=(taxadefalha,1,taxadereparo,1,comandoderrubarservico,comandolevantarservic
            #t2.daemon = True
            t2.start()
         except e:
            print "erro" + str(e)
         nome='k'


   except Exception as e:
      print "AAA" + str(e)

   while 1:
      pass

def captura_csv():
   i=0
   with open("inputmi.txt", "r") as ins:
       array = []
       for line in ins:
      try:
         if i==0:
```

```
            next
         print str(i) + "\n"
         linha = line.split(",")
         print line
         if (str(linha[0][0])=="#"):
            print "Ignorando valor comentado"
            next
         nome = linha[0]
         comandoverificarservico = str(linha[1])
         comandoderrubarservico = str(linha[2])
         comandolevantarservico = str(linha[3])
         taxadefalha = float(linha[4])
         taxadereparo = float(linha[5])
         try:
            t = Thread(target=monitorar, args=(nome, comandoverificarservico))
            #t.daemon = True
            t.start()
            t2 = Thread(target=injetar_reparo_falha,
                args=(taxadefalha,1,taxadereparo,1,comandoderrubarservico,comandolevantarservic
            #t2.daemon = True
            t2.start()
         except e:
            print "erro" + str(e)
      except Exception as e:
         print "erro" + str(e)
         pass

print "[A]rquivo ou [I]nput?"
resposta = raw_input()
if resposta == "A":
   captura_csv()
else:
   captura_inputs()
```