Pós-Graduação em Ciência da Computação

"MODELOS DE DISPONIBILIDADE PARA NUVENS PRIVADAS: REJUVENESCIMENTO DE SOFTWARE HABILITADO POR AGENDAMENTO DE MIGRAÇÃO DE VMS"

Por

Matheus D'Eça Torquato de Melo

Dissertação de Mestrado

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

RECIFE

2014

Universidade Federal de Pernambuco
Centro de Informática
Pós-graduação em Ciência da Computação

Matheus D'Eça Torquato de Melo

"MODELOS DE DISPONIBILIDADE PARA NUVENS PRIVADAS: REJUVENESCIMENTO DE SOFTWARE HABILITADO POR AGENDAMENTO DE MIGRAÇÃO DE VMS''

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*

RECIFE

2014

Catalogação na fonte
Bibliotecária Joana D'Arc L. Salvador, CRB 4-572

Melo, Matheus D'Eça Torquato de.
Modelos de disponibilidade para nuvens privadas: rejuvenescimento de software habilitado por agendamento de migração de VMs / Matheus D'Eça Torquato de Melo. – Recife: O Autor, 2014.
107 f.: fig., tab.

Orientador: Paulo Romero Martins Maciel.
Dissertação (Mestrado) - Universidade Federal de Pernambuco. CIN. Ciência da Computação, 2014.
Inclui referências e apêndices

1. Redes de computadores. 2. Computação em nuvem. I. Maciel, Paulo Romero Martins (orientador). II. Título.

004.6 (22. ed.) MEI 2014-111

Dissertação de Mestrado apresentada por **Matheus D'Eça Torquato de Melo** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "**Modelos de Disponibilidade para Nuvens Privadas: Rejuvenescimento de Software Habilitado por Agendamento de Migração de VMs**" orientada pelo **Prof. Paulo Romero Martins Maciel** e aprovada pela Banca Examinadora formada pelos professores:

_____

Prof. Nelson Souto Rosa
Centro de Informática / UFPE

_____

Prof. Gabriel Alves de Albuquerque Junior
Departamento de Estatística e Informática / UFRPE

_____

Prof. Gustavo Rau de Almeida Callou
Departamento de Estatística e Informática / UFRPE

_____

Prof. Paulo Romero Martins Maciel
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 10 de março de 2014.

_____

Profa. Edna Natividade da Silva Barros
Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

A Deus,
À minha família,
À minha amada Carla,
À família IF-Sertão Salgueiro.

Agradecimentos

A Deus, por me dar forças para poder enfrentar os desafios da vida. Agradeço a Ele por ter sido meu sustento e alento durante minha vida.

À minha família, por todo amor gratuito e sem reservas, que a cada dia revigora minhas energias. Em especial ao meu irmão, Lucas, que "sobreviveu"e sofreu comigo durante todo esse tempo de mestrado.

À minha amada Carla, por todo o seu carinho, compreensão e paciência comigo durante essa fase. Agradeço por todos os momentos que, de uma forma ou outra, me ajudou a superar os problemas encontrados.

À maravilhosa equipe de trabalho do Instituto Federal do Sertão Pernambucano, por ter concedido todo o apoio para conclusão desta pesquisa. Em especial aos meus amigos da coordenação de informática, que sempre colaboraram comigo quando foi necessário.

A todos da Banda Parusia, que proporcionaram momentos de descontração e alívio durante este tempo.

Ao professor Paulo Maciel, por ter removido as barreiras para que a pesquisa acontecesse, e por valorosas orientações para o rumo das atividades. Também aos professores Nelson Rosa, Gustavo Callou e Gabriel Alves, por terem aceitado o convite para participar da banca de avaliação.

A todo o pessoal do grupo MoDCS, pelo apoio, opiniões e direcionamentos para as atividades desenvolvidas. A Jean e Rubens, pelas valorosas revisões de artigos. A Julian que não mediu esforços para ajudar quando necessário. A Carlos Mágno, por ter colaborado no desenvolvimento dos testes de validação.

Às demais pessoas que contribuíram para o desenvolvimento desta pesquisa, ou que me ajudaram a prosseguir durante essa fase.

Ao CIn-UFPE e à CAPES, por todo o suporte recebido.

O lance está no ar
É só você gritar bem alto, alto
Mas, se prepare, prepare
Porque nada no mundo é de graça
Você pode até ter medo.
Mas, ande, caminhe
E só não pare
Não pare nunca.

—CASINO BOULEVARD (Rosa de Saron)

Resumo

A computação em nuvem é utilizada para os mais diferentes propósitos, desde sistemas comerciais até aplicações de tempo real, como vídeo *streaming*. Paralisações e quedas de serviço trazem prejuízos financeiros para as organizações provedoras da nuvem pública, além de denegrir a reputação delas. Assim, a disponibilidade representa um dos principais desafios para tornar a computação em nuvem mais confiável. Os administradores precisam ter mecanismos para estimar a disponibilidade de seus sistemas, para poder definir *SLAs* com mais propriedade. Um dos problemas que afeta diretamente a disponibilidade dos sistemas de nuvem é o envelhecimento de software. Tal problema promove uma degradação gradual do desempenho e confiabilidade das aplicações hospedadas em ambientes de nuvens. Nesse contexto, esta dissertação propõe modelos para avaliação de disponibilidade em ambientes de nuvem com rejuvenescimento habilitado por migração de máquinas virtuais. Para tanto, é adotada uma metodologia que favorece a concepção desses modelos, embasada em modelos de disponibilidade de infraestrutura básica de nuvem (*FrontEnd*, Nó e *VM*), que são validados por intermédio da injeção de falhas e reparos em um ambiente real, e em experimentos práticos de envelhecimento e rejuvenescimento de software. Com os modelos de infraestrutura básica da nuvem são geradas fórmulas fechadas, que passam por uma análise de sensibilidade. Experimentos são realizados para indicar a presença de envelhecimento e a efetividade do método de rejuvenescimento proposto. A partir disso, é construído um modelo hierárquico baseado em *RBD* e *SPN* capaz de representar diferentes agendamentos de migração de VMs para rejuvenescimento. Os estudos de caso mostram a aplicabilidade dos modelos propostos. Em um deles é possível concluir que, considerando os modos operacional e de falha exibidos, o *FrontEnd* é o componente mais crítico para a disponibilidade de infraestruturas básicas. E que, considerando o ambiente estudado, o tempo de reparo é mais sensível que o de falha para a disponibilidade estacionária. Em outro, utilizando o modelo SPN, conclui-se que o agendamento correto de migrações pode acarretar ganhos significativos na disponibilidade do sistema, principalmente naqueles que se submetem às cargas de trabalho mais intensas. Ainda são expostos dois estudos de caso adicionais: um para comparar diferentes mecanismos de redundância (cold-standby e warm-standby), e outro para observar o impacto causado pela interrupção em cada migração na disponibilidade estacionária do sistema.

Palavras-chave: Computação em nuvem. Modelos de disponibilidade. Envelhecimento e rejuvenescimento de software. Validação de modelos. Migração de *VM*.

Abstract

Cloud computing has been applied to various purposes going from commercial systems to real-time applications, such as video streaming. Downtime and service outage brings financial losses to Cloud providers and may damage their reputation. Therefore, the availability is one of the main challenges to turn Cloud more reliable. Systems managers need mechanisms to estimate the system availability in order to define SLAs with more property. One of the issues that affect the availability of Cloud system is software aging. This problem promotes a gradual performance and reliability degradation of the applications hosted on a Cloud. This dissertation proposes a set of availability models to Cloud with rejuvenation enabled by VM migration. The methodology adopted emphasizes models construction, which is based on models for a basic infrastructure of a Cloud (FrontEnd, node and VM) validated by injection of faults and repairs in a real testbed. Aging experiments are also made in a real environment. Closed formulas were generated by means of basic infrastructure cloud models considering a sensitivity analysis. Experiments are performed in order to indicate the behavior of aging and the effectiveness of the rejuvenation method proposed. From this, a hierarchical model is proposed based on *RBD* and *SPN* capable of representing different schedules of migrating VMs for rejuvenation. The aging experiments conducted were able to achieve highlight effects caused by this phenomenon. Moreover, the rejuvenation tests depict the effective method used for aging detected. The case studies show the applicability of the proposed models. Initially, it is possible to conclude that, considering the operational and failure modes, the *FrontEnd* is the most critical component to the availability of basic infrastructure. And that considering the environment studied, the repair time is more sensitive than the failure to stationary availability. Based on SPN model is possible to conclude that the correct scheduling migrations can involve significant gains in the system availability, especially in those who are submitted the most intensive workloads.

Keywords: Cloud Computing. Availability Models. Software Aging and Rejuvenation. Models Validation. VM Migration.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

CPU	*Central Processing Unit*
CTMC	*Continuous Time Markov Chain*
DNS	*Domain Name System*
DTMC	*Discrete Time Markov Chain*
eDSPN	*Extended Deterministic Stochastic Petri Net*
HD	*Hard Disk*
IaaS	*Infrastructure as a Service*
KVM	*Kernel-based Virtual Machine*
MTTF	*Mean Time To Failure*
MTTR	*Mean Time To Repair*
PaaS	*Platform as a Service*
PN	*Petri Net*
RBD	*Reliability Block Diagram*
SaaS	*Software as a service*
SAN	*Storage Area Network*
SLA	*Service Level Agreement*
SPN	*Stochastic Petri Net*
SSH	*Secure Shells*
SUT	*System Under Test*
TTARF	*Time To Aging Related Failure*
VIM	*Virtual Infrastructure Manager*
VM	*Virtual Machine*
VMM	*Virtual Machine Monitor*

Sumário

1 Introdução

Os estudos de disponibilidade e confiabilidade em ambientes de computação em nuvem tornam-se mais relevantes à medida que a utilização desses ambientes começa a permear diferentes contextos. Um dos maiores desafios para a migração de aplicações e serviços para a nuvem é a disponibilidade e confiabilidade desses ambientes (CISCO, 2012). Além de afetar a reputação da empresa em questão, as paralisações das aplicações e serviços de empresas geram prejuízos financeiros. Em PATTERSON (2002), foi feita uma estimativa do montante que é desperdiçado em relação ao porte da empresa. Nesse trabalho o autor propõe a seguinte relação:

Gasto estimado por 1 hora de *downtime* = Gastos com empregados por hora $\times$ Fração dos empregados afetados pela paralisação $+$ Rendimento médio por hora $\times$ Fração do rendimento afetada pela paralisação

É necessário ter um método de avaliação de potenciais riscos de paralisações em nuvens. Estimar o *downtime* de nuvens privadas pode ajudar os administradores e gerentes a estipularem os prejuízos oriundos dessas paralisações. Isto gera resultado direto no planejamento estratégico dessas organizações que utilizam o ambiente de computação em nuvem para hospedar aplicações. Além disso, com um estudo sobre o comportamento do sistema da nuvem é possível construir estratégias de redundância e alta disponibilidade para diminuir as complicações trazidas pelo *downtime*. Em DANTAS et al. (2012a), é apresentado um método de redundância aplicado a nuvens privadas. Utilizando modelos analíticos, são feitas avaliações e modificações de parâmetros em busca das melhores configurações. Nesse trabalho é possível observar que existe redução drástica de perdas em ambientes que utilizam o mecanismo de redundância *warm-standby* (GUIMARAES et al., 2011).

É possível classificar os métodos de avaliação em três grandes áreas: a medição, a simulação e a modelagem numérica (JAIN, 1991; TRIVEDI, 2008). Existem diferenças marcantes entre estas três formas de avaliar sistemas computacionais. A medição é o método de avaliação mais preciso dentre estes, porque utiliza instrumentação diretamente no sistema estudado. A medição acaba sendo mais intrusiva, já que precisa de contato direto com o sistema. É também a mais complicada de desempenhar, pois necessita ao menos de um protótipo executável para viabilizar

sua execução. Por outro lado, a simulação necessita de linguagens de simulação adequadas para o sistema em questão, ou até mesmo de uma ferramenta capaz de realizar a simulação do sistema que se pretende avaliar. Contudo, a simulação pode ser executada a qualquer tempo, mesmo se o sistema a ser avaliado ainda não tiver sido implementado. A modelagem numérica, por sua vez, é o método que usualmente requer menor aparato computacional para ser concretizada. No entanto, sua utilização requer uma atenção especial, devido à necessidade de se lidar com aproximações.

Para robustecer os resultados obtidos dos modelos, eles precisam ser validados através de comparações com valores medidos do sistema real (JAIN, 1991). É possível afirmar que o modelo está validado quando os resultados que são obtidos dos modelos se aproximam (com um intervalo de confiança definido) dos resultados medidos numa plataforma real.

Avaliar a disponibilidade de um sistema por intermédio de medições constitui uma operação muito custosa. As medidas mais importantes na avaliação de disponibilidade são **MTTF** (**M**ean **T**ime **T**o **F**ailure - Tempo médio para falha) e **MTTR** (**M**ean **T**ime **T**o **R**epair - Tempo médio para reparo) (MACIEL et al., 2012). E na maioria dos casos, as medidas de *MTTF* podem ser na ordem de meses ou anos, ou seja, difíceis de mensurar nos sistemas reais. Uma abordagem viável para a avaliação de disponibilidade em sistemas computacionais é a utilização de modelos numéricos capazes de representar os comportamentos e atividades realizadas pelo sistema. Através dos modelos é possível obter estimativas de *downtime* e confiabilidade do sistema sem a necessidade de interagir diretamente com ele (SATHAYE; RAMANI; TRIVEDI, 2000).

Outra vantagem em se utilizar modelos analíticos é a possibilidade de encontrar fórmulas fechadas para os ambientes que estão sendo avaliados. Com a utilização de fórmulas, a complexidade da análise dos modelos é reduzida, além de facilitar a análise de sensibilidade nos parâmetros (CHEN; TRIVEDI, 2002).

Logo, para estimar o *downtime* em ambientes de nuvens é possível utilizar a modelagem numérica (DANTAS et al., 2012b). Todavia, um problema recorrente em trabalhos de modelos de disponibilidade para sistemas é a falta de um processo de validação para estes. Devido à dificuldade de realizar medições de métricas de disponibilidade em ambientes reais, a validação de modelos de disponibilidade torna-se uma tarefa complexa de desempenhar.

Em SOUZA et al. (2013) é proposta uma ferramenta para auxiliar no processo de injeção de falhas e reparos em ambientes de nuvem. Tal abordagem pode ser útil para a validação de modelos de disponibilidade de sistemas computacionais. A abordagem de injeção de falhas (CARREIRA J.: SILVA, 1998) permite estudar a resiliência[1] da nuvem (ARLAT et al., 1990), checando como é o seu comportamento perante comportamentos de falhas. No processo de injeção de falhas, o administrador do ambiente consegue controlar a frequência com que as falhas são inseridas, bem como a natureza de cada falha. Porém, em situações reais, nem sempre é possível ter ciência do surgimento de uma falha de modo prévio, além de ser necessário entender

[1] Capacidade de voltar ao seu estado natural após alguma falha

sua natureza para realizar o reparo adequado.

Um dos tipos de falha que surge de modo gradual e prejudica bastante a disponibilidade de um sistema é o envelhecimento de software (GROTTKE; MATIAS; TRIVEDI, 2008; GRAY; SIEWIOREK, 1991). O envelhecimento manifesta-se como uma degradação gradual do estado do software, podendo levá-lo a falhas e travamentos totais. Na computação em nuvem, esse problema foi apresentado por (ARAUJO, 2012), que mostra os problemas de envelhecimento em plataformas *Eucalyptus*.

Para prevenir que problemas de envelhecimento levem o sistema a encontrar falhas, é possível realizar o rejuvenescimento de software (KOURAI; CHIBA, 2011). Trata-se de uma técnica proativa que visa limpar todo o *status* acumulado de envelhecimento. Uma técnica de rejuvenescimento factível para computação em nuvem é o rejuvenescimento baseado na migração de máquinas virtuais (MELO et al., 2013). Com a migração habilitada é possível deslocar a VM para outro nó da nuvem antes de realizar o *reboot* do sistema operacional (ou *restart* de serviços), assim é possível evitar que o serviço seja interrompido por muito tempo.

1.1 Motivação e justificativa

Os estudos de disponibilidade em computação em nuvem tornam-se cada vez mais relevantes à medida que este tipo de plataforma começa a hospedar diferentes tipos de sistemas. Em BUYYA et al. (2009), a computação em nuvem é colocada como uma plataforma em potencial para sistemas com os mais diversos propósitos. Requisitos de disponibilidade e confiabilidade aparecem como as principais barreiras para migrar as aplicações para nuvens (CISCO, 2012). Portanto, a avaliação de disponibilidade faz-se necessária para estas plataformas. Além disto, é imprescindível que administradores desse tipo de ambiente possuam métodos para estimar sua disponibilidade perante diversos cenários, para assim estabelecer *SLA*s (**S**ervice **L**evel **A**greements - Acordos de nível de serviço) adequadas.

Assim, métodos para aumentar disponibilidade são de grande importância em computação em nuvem. Um dos potenciais problemas para a alta disponibilidade em ambientes de computação em nuvem é o envelhecimento de software. Diversos trabalhos (HUANG et al., 1995; CASTELLI et al., 2001) mostram que é necessário dar atenção aos efeitos colaterais relativos ao envelhecimento. Portanto, é necessário propor rejuvenescimentos adequados para obter melhores resultados de disponibilidade.

Logo, estabelecer mecanismos para evitar efeitos de envelhecimento é um passo importante para melhorar a disponibilidade em ambientes de nuvem. É desejável que as metodologias e modelos propostos possam ser reaproveitados e adaptados para atender necessidades específicas de outros ambientes de nuvem.

1.2 Objetivos

Este trabalho propõe um conjunto de modelos hierárquicos para avaliação de disponibilidade em nuvens, com rejuvenescimento de software habilitado por agendamento de migrações de máquinas virtuais. A avaliação é realizada para obter políticas adequadas para cada cenário estudado, visando maximizar a disponibilidade estacionária do sistema.

De modo mais específico, para alcançar o objetivo principal, este trabalho propõe-se a:

- Definir modelos de disponibilidade para plataformas de computação em nuvem que apresentam efeitos de envelhecimento de software;
- Agregar o comportamento de rejuvenescimento habilitado por agendamento de migração aos modelos de avaliação de disponibilidade;
- Avaliar diferentes políticas de rejuvenescimento de software em busca da mais adequada para cada cenário estabelecido.

Para robustecer o estudo foram estabelecidas atividades secundárias com o intuito de embasar os modelos gerados para alcançar o objetivo principal. São estas:

- Realizar estudos experimentais em ambientes de computação em nuvem, visando encontrar os indícios de envelhecimento;
- Determinar a efetividade de mecanismos de rejuvenescimento para dirimir os efeitos causados pelo envelhecimento;
- Propor e validar modelos para infraestruturas básicas de computação em nuvem, a fim de serem utilizados como insumo para o aprofundamento dos estudos, que consideram o envelhecimento de software;
- Estudar a relevância das interrupções causadas por cada migração para a disponibilidade estacionária do sistema.

1.3 Uma visão geral

Esta seção apresentará uma visão geral das atividades e métodos adotados para alcançar os objetivos propostos por esta dissertação. Dividiu-se a metodologia em ações menores com objetivos específicos, que resultam em insumos para a produção dos modelos finais.

A primeira atividade pertinente para este objetivo é realizar experimentos em uma plataforma de nuvem real, visando encontrar indícios de envelhecimento e rejuvenescimento que justifiquem a construção dos modelos. Por isso, foram realizados experimentos com estresse

acelerado e monitoramento de recursos em uma plataforma de nuvem privada *OpenNebula* (OPENNEBULA, 2013).

Os experimentos visam detectar efeitos de envelhecimento de software na plataforma estudada e, ainda, checar a efetividade do mecanismo de rejuvenescimento proposto. Algumas ferramentas nativas do sistema operacional *Ubuntu Linux* foram utilizadas para monitorar o ambiente. E alguns scripts para geração de carga de trabalho foram desenvolvidos para testar o ambiente. A carga de trabalho para envelhecimento de software foi baseada em operações de anexação e desmonte discos virtuais em uma VM (ARAUJO, 2012). Foi submetida também uma carga de requisições Web para um servidor alocado na VM, a fim de observar a degradação que a carga de envelhecimento causa em seu desempenho.

A metodologia de rejuvenescimento deve ser adequada para o envelhecimento específico do ambiente. O método selecionado foi o de rejuvenescimento de software habilitado por migração de máquinas virtuais. A migração consiste em mover a *VM* da máquina física que a hospeda (máquina *source*) para outra máquina física (máquina *target*). Assim, a VM que está no sistema que acumula efeitos de envelhecimento (no caso, a máquina *source*) é migrada para outra que, no momento, está livre de efeitos de envelhecimento. Logo que a VM é completamente deslocada para outra máguina física, a máquina *source*, que está com efeitos de envelhecimento acumulado, pode ser rejuvenescida sem interromper o sistema que está sendo disponibilizado pela VM.

Então, os experimentos foram realizados com tempo de duração de treze dias consecutivos, incluindo-se a metodologia de rejuvenescimento de software. Os resultados indicam que a utilização da migração como suporte ao rejuvenescimento de software constitui uma técnica adequada para tratar o envelhecimento observado na plataforma. Toda a descrição detalhada dos experimentos e resultados pode ser encontrada no Capítulo 3. Esses resultados experimentais justificam a criação de modelos de disponibilidade para nuvens que enfrentam problemas de envelhecimento de software.

Após o término dos experimentos que detectaram o envelhecimento, é necessário dar início à construção dos modelos de disponibilidade para nuvens privadas. O primeiro conjunto de modelos proposto é baseado em *RBD* e em *CTMC*. O intuito dos modelos iniciais é representar uma infraestrutura básica de computação em nuvem, composto por um *FrontEnd*, um nó e uma *VM*. Utiliza-se a abordagem hierárquica para a construção dos modelos, com o objetivo de reduzir a complexidade destes. Assim, blocos *RBD* são empregados para representar o comportamento do *FrontEnd* e uma *CTMC* para representar o comportamento do nó que hospeda a *VM*. A partir de cada submodelo foram obtidas fórmulas fechadas para cálculo da disponibilidade. Ao fim, combinando as equações, obteve-se uma fórmula fechada para cálculo da disponibilidade do sistema por completo.

Para aumentar a confiança nos resultados do modelo para infraestrutura básica de nuvem, decidiu-se validá-lo por intermédio de um processo baseado em injeção de falhas e reparos. Experimentos baseados nesse método foram realizados numa plataforma real. Foi necessário

implementar um injetor de falhas e reparos para o ambiente, bem como utilizar ferramentas para monitorar o sistema. Os parâmetros utilizados nos scripts de teste foram obtidos de artigos já publicados. Confrontando os resultados do monitoramento em ambiente real com os resultados obtidos dos modelos, é possível afirmar que o modelo construído realmente corresponde ao comportamento do sistema real.

Com o modelo da infraestrutura validado, a atividade seguinte é adaptá-lo a fim de que se passe a contemplar comportamentos de envelhecimento e rejuvenescimento de software. Assim, foi concebido outro modelo hierárquico capaz de representar tais comportamentos. O propósito da utilização de modelos hierárquicos é reduzir a complexidade na construção dos modelos. A partir da utilização de modelos hieráquicos é possível relacionar diferentes modelos para representar o comportamento do sistema. Os modelos concebidos para os comportamentos de envelhecimento e rejuvenescimento foram baseados em *RBD* e *SPN*. Foi necessário a utilização de diferentes modelos pois o natureza dos fenômenos modelados exige maior dependência entre seus componentes. A arquitetura considerada é similar a que foi considerada para os modelos da infraestrutura básica, acrescidos apenas de uma máquina física para atuar como nó da nuvem. Este acréscimo é realizado por causa da metodologia de rejuvenescimento adotada. O comportamento da migração da VM também é incluído no modelo, levando-se em consideração apenas o *downtime* causado por esta operação. Nesses modelos, a atividade de rejuvenescimento baseada em migrações obedece a um agendamento pré-definido. Existe um submodelo da rede que atua como um relógio, então quando o intervalo de tempo escolhido para o rejuvenescimento é alcançado, a migração está habilitada a ocorrer. Caso os outros pré-requisitos da migração sejam satisfeitos, a migração da *VM* ocorre, liberando assim o nó para o rejuvenescimento. O conjunto de modelos é apresentado no Capítulo 4.

1.4 Estrutura da Dissertação

O trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta o conjunto de conceitos necessários para entender as metodologias, experimentos e modelos propostos neste trabalho. O Capítulo, 3, mostra um estudo experimental de envelhecimento e rejuvenescimento desenvolvido na plataforma de estudo. O Capítulo 4 contém o conjunto de modelos concebidos para avaliação de disponibilidade nas nuvens privadas, seguido por um processo de validação. Logo após, é apresentado o modelo hierárquico utilizado para avaliar políticas de rejuvenescimento de software. Depois, o Capítulo 5 mostra alguns possíveis estudos de caso para os modelos propostos, com o intuito de destacar a aplicabilidade deles. Os trabalhos relacionados são apresentados no Capítulo 6. Por fim, o Capítulo 7 destaca as conclusões e contribuições deste trabalho, bem como os trabalhos futuros.

2

Fundamentação Teórica

Este capítulo apresenta um conjunto de conceitos fundamentais para o entendimento do conteúdo proposto nesta dissertação. É importante salientar que o material apresentado aqui visa fornecer ao leitor subsídios para entender a metodologia e técnicas aplicadas neste trabalho, e que um material mais aprofundado sobre cada tema poderá ser encontrado nas referências utilizadas. Este capítulo está organizado da seguinte maneira: inicialmente são expostos os conceitos relativos à disponibilidade de sistemas. Depois são apresentados conceitos básicos sobre modelos analíticos para avaliação de disponibilidade. Em seguida é apresentada a base do processo de injeção de falhas. Ainda neste capítulo tem-se um levantamento básico sobre computação em nuvem e os conceitos básicos relativos ao envelhecimento e rejuvenescimento de software.

2.1 Disponibilidade

Um dos fundamentos do conceito de disponibilidade é a **confiabilidade** (*reliability*). Confiabilidade pode ser definida como a probabilidade de um Sistema S não falhar até um tempo t. A função da confiabilidade é representada por $R(t)$, onde t corresponde ao tempo que se deseja obter a confiabilidade (MACIEL et al., 2012). Exemplificando: a função para determinar a confiabilidade de um sistema ao fim de seu primeiro mês de vida é $R(720)$, ou seja esta função corresponde à probabilidade do sistema não ter falhado até às 720 horas (um mês). Por definição, $R(0) = 1$ (sistema funcionando).

A **disponibilidade** (*availability*) é um dos atributos mais importantes para sistemas computacionais. Entende-se como disponibilidade a prontidão de um sistema para entregar o serviço correto para um usuário. Em outras palavras, a disponibilidade é utilizada para estimar a proporção de tempo que o sistema está ativo dentro de todo o seu tempo de vida (MACIEL et al., 2012). A disponibilidade pode ser obtida pela através da Equação 2.1.

$$Disponibilidade = \frac{TempoAtivo}{TempoAtivo + TempoInativo} \tag{2.1}$$

Elaborar técnicas para estimar a disponibilidade de um sistema torna-se uma atividade

muito importante, à medida que os administradores precisam de dados para definir requisitos de contrato e *SLA*s de serviço adequados. Especificamente os sistemas *online* (ex. sistemas bancários, sites da Internet) têm a disponibilidade como um dos pontos cruciais para a qualidade do serviço disponibilizado porque solicitações a esses serviços precisam ser atendidas constantemente. Haja vista, que vários desses sistemas já operam utilizando computação em nuvem, é importante estudar a disponibilidade desse ambiente para prover melhores condições aos serviços hospedados. Além disso, é importante estimar a disponibilidade do sistema porque paralisações e interrupções inesperadas na entrega de serviços acarretam prejuízos financeiros e podem comprometer a reputação de seus provedores (HAGEN; SEIBOLD; KEMPER, 2012).

O **MTTF** representa o tempo médio para ocorrência de falhas. Trata-se de uma métrica que corresponde ao tempo estimado para acontecer uma falha. O *MTTF* pode ser calculado através da Expressão 2.2.

$$MTTF = \int_0^\infty R(t) \times dt \tag{2.2}$$

O **MTTR**, por sua vez, é o tempo médio para reparo. O *MTTR* é baseado num atributo conhecido como *manutenabilidade*. Entende-se como manutenabilidade a probabilidade de um Sistema S ser reparado com um tempo t. Geralmente a manutenabilidade é denotada pela função $M(t)$ (MACIEL et al., 2012). A obtenção do *MTTR* pode ser realizada pela Equação 2.3.

$$MTTR = \int_0^\infty M(t) \times dt \tag{2.3}$$

Quando o *MTTF* e o *MTTR* são exponencialmente distribuídos, eles podem ser representados por suas respectivas taxas, onde λ representa a medida inversa do *MTTF* (ou taxa de falha) e μ a medida inversa do *MTTR* (ou taxa de reparo). As taxas mostram a quantidade de ocorrências de falhas/reparos em um determinado intervalo de tempo. Assim, pode-se obter a disponibilidade através da Equação 2.4.

$$Disponibilidade = \frac{\mu}{\lambda + \mu} \tag{2.4}$$

Os valores admitidos pela disponibilidade estão limitados por 0 (mínimo) e 1 (máximo), onde o valor máximo representa a não ocorrência de interrupções durante a entrega do serviço, considerando o intervalo de tempo definido. Exemplificando, se o valor obtido da disponibilidade for de 0,995, isso representa que o sistema estará disponível em 99,5% do intervalo de tempo estudado.

Outro fator a ser destacado no estudo da disponibilidade do sistema é a Indisponibilidade. A Indisponibilidade representa a probabilidade do sistema não estar disponível. Pode-se calculá-la com a Equação 2.5. Utilizando-se do valor da Indisponibilidade é possível obter o *downtime* estimado do sistema em um intervalo de tempo. Caso seja necessário obter o *downtime* anual de um sistema em horas, considerando-se um ano de 365 dias, pode-se utilizar a Equação 2.6.

$$Indisponibilidade = 1 - Disponibilidade \tag{2.5}$$

$$Downtime_{anual} = Indisponibilidade \times 8760 \tag{2.6}$$

Curva da banheira

Estudos prévios do comportamento de falha em componentes de hardware mostram que a taxa de falha varia conforme mostra a Figura 2.1. Esse comportamento é geralmente conhecido por "curva da banheira"(EBELING, 1997).

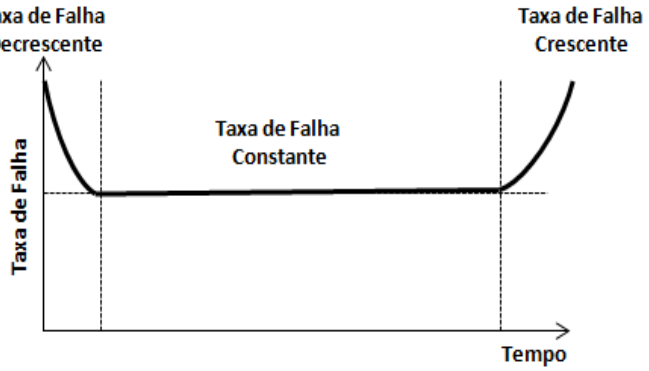


Figura 2.1: Curva da banheira, adaptada de (EBELING, 1997)

A "curva da banheira"é extensivamente estudada em (KLUTKE; KIESSLER; WORTMAN, 2003), onde é apresentada uma visão crítica sobre este padrão e variações oriundas de diferentes distribuições de probabilidade. Contudo, notam-se três comportamentos distintos:

a) A falha prematura (ou *burn-in period*) é o período inicial, onde a ocorrência de falhas possui taxas mais elevadas, ou seja, a probabilidade de ocorrência destas é maior. No caso do hardware essa taxa é alta devido às falhas de fabricação do hardware. É necessário utilizar testes e técnicas para evitar que tais falhas aconteçam.

b) No segundo momento, após o período de falha prematura, o sistema entra na fase de vida útil (*useful life*). Nesse período a taxa de falhas é constante.

c) Taxa de falha crescente (*wear-out period*), representa uma fase na vida do equipamento onde a taxa de falhas cresce com o passar do tempo.

É importante entender o comportamento dos eventos ocorridos nos períodos da curva da banheira para conseguir construir modelos que os representem. Em (TRIVEDI, 2008), são apresentadas algumas características relevantes sobre cada uma das fases da "curva da banheira", tais características auxiliam na construção de modelos para avaliação de disponibilidade. Dentre

as informações relevantes, conclui-se que, perante a natureza dos eventos de falha da curva da banheira, é possível aproximar os eventos de falha em cada uma das fases, utilizando alguma distribuição de probabilidades específica. Logo, para aproximar os modelos dos comportamentos de falha descritos na curva da banheira, é necessário que os eventos modelados ocorram segundo a distribuição de probabilidades adequada. As distribuições propostas em (TRIVEDI, 2008) são as seguintes:

- *Burn-in period* - Taxa de falha decrescente. Uma distribuição de probabilidade que pode ser utilizada para representar estes eventos é a *Weibull*;
- *Useful life* - Taxa de falha constante. Pode ser modelada com a distribuição de probabilidade exponencial;
- *Wear-out period* - Taxa de falha crescente[1]. Distribuições: *Hipo-exponencial, Erlang*.

A distribuição *Erlang* é um caso específico da distribuição hipo-exponencial, onde todas as fases possuem a mesma taxa (TRIVEDI, 2008). Em distribuições com fases, caso o número de fases seja alto, os eventos ocorridos obedecendo à distribuição tenderão ao determinismo, ou seja, um tempo fixo.

2.2 Modelos para avaliação de disponibilidade

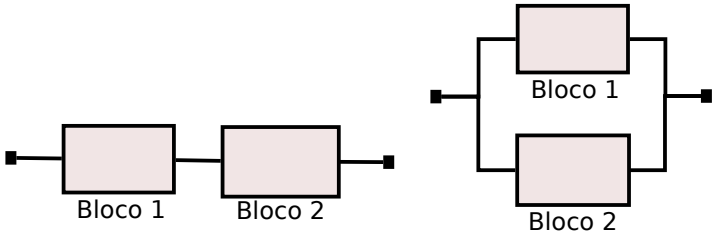
Dentre as técnicas para avaliação de disponibilidade, a modelagem destaca-se por permitir que sejam feitas aferições de disponibilidade sem interagir com o sistema real. Os modelos podem ser classificados como combinacionais e em baseados em espaço-estado. Nesta seção serão exibidos os conceitos básicos relativos aos modelos utilizados para as avaliações deste estudo.

Diagrama de blocos de confiabilidade (RBD)

Os diagramas de blocos de confiabilidade[2] (RBD) (KIM WEYNS, 2013) são modelos combinacionais apresentados como um conjunto de blocos, que podem ser organizados em série, em paralelo ou em estruturas *k-out-of-n*, ou ainda em combinações entre essas organizações (TRIVEDI et al., 1996). A Figura 2.2 traz alguns exemplos de RBD, mostrando estruturas básicas em série (Figura 2.2(a)) e em paralelo (Figura 2.2(b)).

A estrutura do RBD apresenta a organização necessária para o sistema funcionar. Em outras palavras, o diagrama representa o modo operacional do ambiente que está sendo modelado. O modo operacional indica quais componentes devem estar funcionando para o sistema responder adequadamente. Caso seja necessário que todos os componentes do sistema estejam funcionando,

[1] Também denominada por *IFR - Increasing Failure Rate*
[2] *Reliability Block Diagram (RBD)*



(a) Série (b) Paralelo

Figura 2.2: Estruturas básicas de RBDs

para que ele responda corretamente, então o modelo deve ser organizado com blocos em série. Caso seja necessário que ao menos um componente esteja funcionando para o sistema responder, os blocos do modelo devem ser organizados em paralelo. Estruturas *k-out-of-n* representam situações onde, para o sistema funcionar é necessário que, ao menos k componentes funcionem dentro de um montante n (KIM WEYNS, 2013). Para o escopo deste trabalho não foram utilizadas estruturas *k out of n*.

Por exemplo, o RBD exibido na Figura 2.3 representa um sistema computacional composto por um Servidor de Nomes (*DNS*) e dois Servidores Web (*ServidorWeb1* e *ServidorWeb2*). Este sistema serve para disponibilizar uma página web que é alocada em ambos os servidores web. Para acessar este conteúdo é necessário, antes, ter o sistema para realizar a tradução de nomes, possibilitando assim a comunicação com um dos Servidores Web. Neste caso, é necessário que o servidor de nome funcione e que ao menos um dos servidores web esteja respondendo às solicitações realizadas.

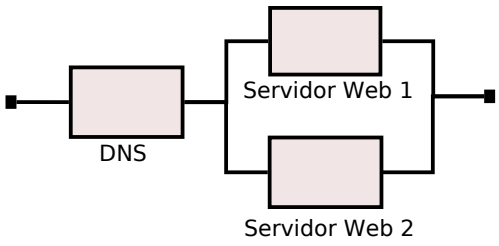


Figura 2.3: RBD Exemplo

A disponibilidade de modelos RBD com blocos em série é obtida através do produto das disponibilidades de cada um dos n blocos componentes (ČEPIN, 2011). Sendo assim, a disponibilidade pode ser calculada a partir da Equação 2.7, onde, A_s (*Availability*) corresponde ao valor da disponibilidade do sistema.

$$A_s = \prod_{i=1}^{n} A_i \tag{2.7}$$

Para modelos RBD em paralelo, a disponibilidade pode ser obtida através da Equação 2.8, onde A_s representa a disponibilidade do sistema; A_i, a disponibilidade do componente i e n o número de componentes

$$A_s = 1 - \prod_{i=1}^{n}(1 - A_i) \tag{2.8}$$

A organização em blocos permite a modelagem de situações onde os comportamentos de falha e reparo de cada componente não afetam os demais. Isso facilita a modelagem de sistemas que não possuem dependência entre os componentes. Por outro lado, como os comportamentos de falha e reparo são isolados nos blocos, não é possível construir situações mais complexas e com dependências entre os componentes. Quando for necessário representar dependência entre os componentes, o projetista deve recorrer aos modelos espaço-estado ou à abordagem hierárquica.

Cadeias de Markov (CTMC)

As cadeias de markov de tempo contínuo (*CTMC*) (BAIER et al., 2003) consistem em processos estocásticos de tempo contínuo que satisfazem a propriedade markoviana[3] e que são divididos em estados bem definidos. Um processo estocástico pode ser definido como um conjunto de variáveis aleatórias $X(t)$ determinadas a partir de um espaço amostral. Os valores assumidos por $X(t)$ são chamados de estados, e o conjunto de todos os estados possíveis é chamado de espaço de estados, I. Assim, o processo estocástico é markoviano se, para todo $t_0 < t_1 < \ldots < t_n < t_{n+1}$ e para todo $X(t_0), X(t_1), \ldots, X(t_n), X(t_{n+1})$, os valores de $X(t_{n+1})$ dependerem somente do último valor $X(t_n)$. Logo, a cadeia de Markov é representada por uma sequência de variáveis aleatórias discretas $X(t_n)$ (estados), onde os valores de t_n podem ser discretos (nas Cadeias de Markov de tempo discreto - *DTMCs*) ou contínuos (*CTMCs*) (BOLCH et al., 1998). As taxas (em *CTMCs*) ou probabilidade (em *DTMCs*) de ocorrência de eventos (transições) obedecem a distribuições exponenciais ou geométricas, respectivamente (STEWART, 1994).

As variáveis aleatórias discretas constituem os estados da cadeia de Markov. As mudanças de estados são chamadas de transições, e representam, no modelo, a probabilidade ou taxa para aquela mudança de estado específica ocorrer (CASSANDRAS; LAFORTUNE, 2008).

As transições entre estados representam a ocorrência de eventos (MACIEL et al., 2012). A Figura 2.4 mostra uma CTMC de exemplo com dois estados e duas transições. Neste modelo o estado 1 representa que o sistema está funcionando. A partir deste, a falha pode ocorrer, levando ao estado 2. A partir do estado 2 apenas o reparo é possível.

A matriz Q possui as informações sobre as transições dos estados na cadeia de Markov. Ela é utilizada para a resolução de cadeias de Markov. Cada elemento localizado fora da diagonal principal representa a taxa de ocorrência dos eventos que efetivam a transição dos estados do sistema. Os elementos contidos na diagonal principal são os valores necessários para que a soma dos elementos de cada linha seja igual a zero. Retomando a CTMC da Figura 2.4, a matriz Q

[3] *Markov property:* o comportamento futuro de um processo depende apenas de seu estado atual e o passado deve ser ignorado (SIEGERT; FRIEDRICH; PEINKE, 1998).

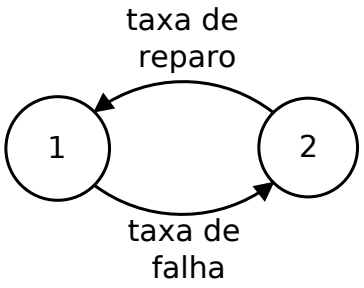


Figura 2.4: CTMC de exemplo

correspondente é exibida na Equação 2.9.

$$Q = \begin{pmatrix} -\text{falha} & \text{falha} \\ \text{reparo} & -\text{reparo} \end{pmatrix} \tag{2.9}$$

O vetor π representa o autovetor unitário da matriz de transição, onde o valor de cada elemento do vetor representa a probabilidade do i-ésimo estado no regime estacionário. Para o exemplo dado, o vetor é exibido na Expressão 2.10.

$$\pi = \left\{ \frac{\text{falha}}{\text{falha} + \text{reparo}}, \frac{\text{reparo}}{\text{falha} + \text{reparo}} \right\} \tag{2.10}$$

A soma de todos os elementos do vetor de probabilidades π deve ser igual a 1 (ARAUJO, 2009). A Equação utilizada para resolução de uma cadeia de Markov é obtida por intermédio da Equação linear 2.11.

$$\pi Q = 0 \tag{2.11}$$

A utilização de CTMCs permite avaliações tanto de desempenho, quanto de disponibilidade. Suas aplicações vão desde a quantificação da vazão de uma linha de produção até a determinação de tempos de falha e reparo em sistemas críticos (BAIER et al., 2003).

Entretanto, construir CTMCs com muitos estados manualmente, pode tornar-se uma atividade suscetível a erros, devido à grande quantidade de detalhes da cadeia.

Redes de Petri (PN)

Propostas por Carl Adam Petri em 1962 (PETRI, 1962), em sua tese de doutorado intitulada "*Kommunikation mit Automaten*"(Comunicação com autômatos), as Redes de Petri (PN) constituem uma ferramenta de modelagem gráfica e matemática aplicável a vários tipos de sistemas. As PNs são adequadas para descrever e estudar sistemas caracterizados por serem concorrentes, assíncronos, distribuídos, paralelos, não determinísticos e/ou estocásticos (MURATA, 1989).

As PNs são constituídas por quatro componentes principais: os lugares (Figura 2.5(a)), as transições (Figura 2.5(b)), os arcos (Figura 2.5(c)) e as marcas ou *tokens* (Figura 2.5(d)).

Os lugares representam as variáveis de estado e as transições correspondem às ocorrências de eventos realizados pelo sistema (MACIEL; LINS; CUNHA, 1996). Em Redes de Petri, a ocorrência de um evento está atrelada a um conjunto de pré-requisitos explícitos nas variáveis de estado do sistema. Ou seja, existe uma interligação entre os lugares e transições que habilita a realização de alguma tarefa no ambiente. Após a ocorrência de algum evento, o sistema pode alcançar outra configuração, e os lugares podem ter suas informações alteradas.

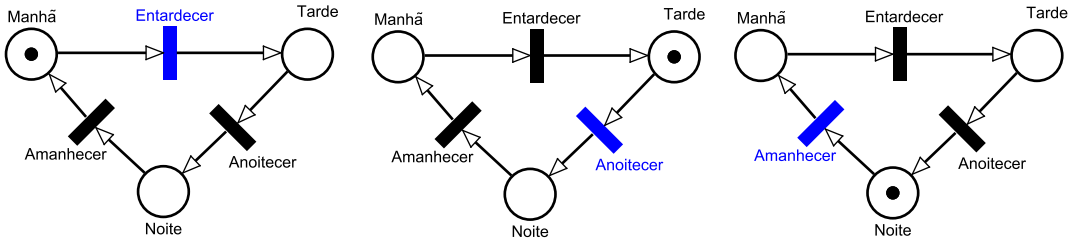
(a) Lugar (b) Transição (c) Arco (d) *token*

Figura 2.5: Componentes das Redes de Petri

Para a modelagem dos eventos e relação entre as variáveis de estado do sistema, nas PNs utilizam-se os lugares interligados às transições por intermédio de arcos dirigidos que indicam o fluxo de trabalho do sistema modelado. Além disto, os arcos definem quais os conjuntos de variáveis que estão envolvidas numa determinada ação, indicando assim quais são as pré-condições para que ela ocorra. Não há interligação direta entre os componentes do mesmo tipo, ou seja, um lugar não é ligado diretamente a outro, o mesmo vale para as transições.

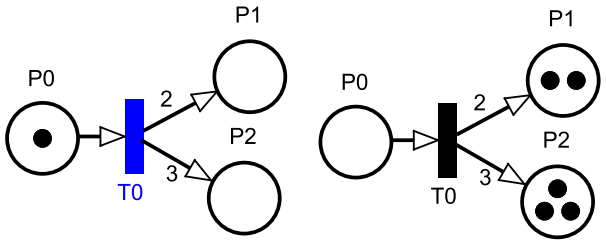
Antes de se conceber as PNs relativas a um comportamento de um sistema qualquer, é necessário entendê-lo de modo detalhado para que as PNs o representem do modo mais fiel possível. Assim, ilustrando os conceitos de Redes de Petri, um exemplo pertinente é apresentado em (MACIEL; LINS; CUNHA, 1996), onde uma PN é utilizada para representar o ciclo de turnos de um dia. No modelo existem três estados: `Manhã`, `Tarde` e `Noite`; e três transições: `Entardecer`, `Amanhecer` e `Anoitecer`. A situação atual do sistema pode ser observada através da posição do *token* na rede. Conforme visto anteriormente, cada transição representa um evento que acontecerá caso as condições sejam atendidas. Neste caso, o sistema representa o ciclo de turnos durante o passar do tempo, assim os eventos se resumem no seguinte: `Entardecer` (transição do lugar `Manhã` para o lugar `Tarde`), `Anoitecer` (transição do lugar `Tarde` para o lugar `Noite`) e `Amanhecer` (transição do lugar `Noite` para o lugar `Manhã`). O ciclo da Rede é apresentado na Figura 2.6, onde é possível observar a situação atual do sistema através do *token*. A posição do *token* na rede habilita o disparo da transição subsequente.

Os arcos que interligam os lugares às transições podem ser multivalorados. Isso indica que um determinado evento pode requerer mais de um *token* para ser realizado, ou ainda, que o resultado de alguma atividade resulta em mais de um *token* para os lugares de saída. Os pesos atribuídos aos arcos indicam a quantidade de *tokens* necessária para a realização de uma tarefa. A Figura 2.7 contém um exemplo.



(a) Manhã (b) Tarde (c) Noite

Figura 2.6: Rede de Petri - Ciclos de turnos do dia, adaptada de (MACIEL; LINS; CUNHA, 1996)



(a) Antes do disparo de `T0` (b) Depois do disparo de `T0`

Figura 2.7: Rede de Petri - Arcos Multivalorados

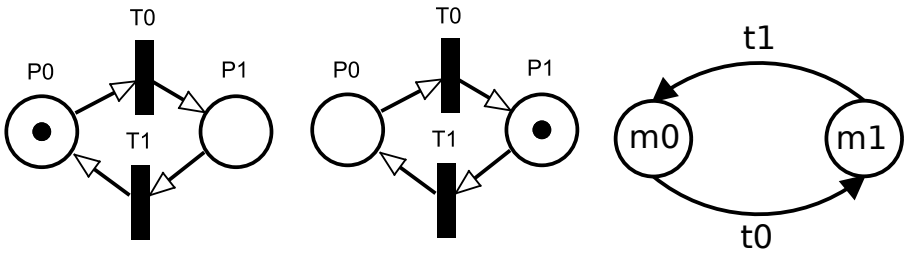
Formalmente, as Redes de Petri são definidas por uma quíntupla $PN = \{P, T, I, O, \mu_0\}$ onde:

- P é o conjunto de lugares;
- T é o conjunto de transições, $P \cap T = \varnothing$;
- $I, O : T \times P \to \mathbb{N}$, são funções que mostram os lugares de entrada e saída das transições, respectivamente;
- $\mu_0 : P \to \mathbb{N}$ é uma função que denota a marcação inicial dos lugares da rede.

As funções I e O mostram, respectivamente, os arcos de entrada e saída de uma dada transição $t \in T$ para um lugar $p \in P$. Para obter-se o peso do arco que conecta o lugar p a t utiliza-se a notação $I(t, p)$. O que também pode ser utilizado para os arcos de saída O. Além da notação de função, é possível utilizar a notação matricial para estas funções, onde I é a matriz de entrada e O a matriz de saída.

A distribuição dos *tokens* dentro da Rede é chamada de **marcação** (*marking*). A marcação determina o estado atual da Rede de Petri, indicando quantos *tokens* cada lugar hospeda em determinados momentos. A partir da marcação é possível entender qual é o estado geral do sistema que está sendo estudado. Além disto, as marcações servem de base para o **grafo de alcançabilidade** (*reachability graph*) (MURATA, 1989) que é um dos principais métodos para a obtenção de resultados numéricos a partir de uma PN.

O grafo de alcançabilidade revela o conjunto de marcações que são atingidas por intermédio do disparo das transições do sistema. De maneira formal, o grafo de alcançabilidade pode ser definido como uma tupla (V, E), onde V é o conjunto de vértices, que correspondem às marcações factíveis da PN e E é o conjunto de arestas rotuladas que mostram as transições responsáveis por levar a rede até a outra marcação. Na Figura 2.8, tem-se uma rede com apenas duas marcações factíveis $M = \{m_0 = |1,0|, m_1 = |0,1|\}$, seu grafo de alcançabilidade é exibido na Figura 2.8(c).



(a) Marcação $m0$ (b) Marcação $m1$ (c) Grafo de alcançabilidade

Figura 2.8: Rede de Petri - Grafo de alcançabilidade

As Redes de Petri são capazes de representar comportamentos de paralelismo e concorrência em sistemas. Utilizando-se de Redes de Petri é possível construir modelos que representem sistemas computacionais (hardware e software), sistemas de manufatura e sistemas de logística de transporte, por exemplo, (PETERSON, 1977; REISIG; ROZENBERG, 1998).

Extended Deterministic Stochastic Petri Nets (eDSPN)

As Redes de Petri Determinísticas Estendidas (*eDSPN*) constituem uma uma extensão da PN lugar-transição (apresentada anteriormente) que permite que o disparo de eventos aconteça, obedecendo a tempos com distribuições exponenciais (transições estocásticas) ou ainda com retardo igual a zero (transição imediata) (KARTSON et al., 1994). Além disto, ainda é possível que as transições disparem com um tempo determinístico, ou seja, um tempo fixo.

Outra característica das *eDSPN*s é o arco inibidor. Os arcos inibidores atuam inibindo o disparo de certas transições. Assim como arcos comuns, os arcos inibidores podem possuir peso. Neste caso, o peso indicará a quantidade necessária para impedir o disparo de uma determinada transição. A Figura 2.9 e 2.10 exibe a representação gráfica das transições estocásticas e dos arcos inibidores.

Para representar diferentes distribuições de probabilidades em modelos *eDSPN*s é necessário utilizar as *throughput nets* ARAUJO (2009). O intuito das *throughput nets* é propor um conjunto de lugares e transições (sub-rede) que corresponda ao comportamento de alguma distribuição de probabilidade específica. Tais sub-redes deverão ser acopladas aos modelos que necessitem de tais distribuições de probabilidade. A Figura 2.11 mostra um conjunto de *throughput nets* que são capazes de representar diferentes distribuições de probabilidades. A

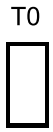
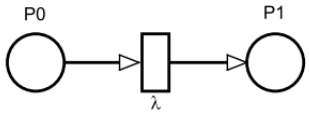


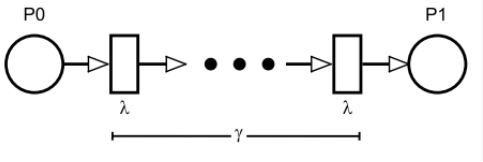
Figura 2.9: Transição Estocástica

Figura 2.10: Arco inibidor

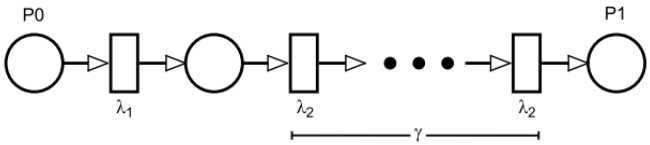
utilização dessas redes específicas faz-se muito útil para representação de sistemas que possuam comportamentos não exponenciais, possibilitando, assim, a geração de modelos mais complexos de Redes de Petri.



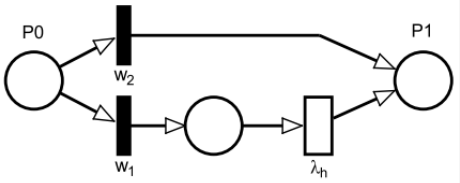
(a) Exponencial.



(b) Erlang.



(c) Hipo-Exponencial.



(d) Hiper-Exponencial.

Figura 2.11: Exemplos de *throughput nets* com distribuições específicas

A resolução de Redes de Petri Estocásticas pode ser realizada por intermédio de simulação ou por intermédio do grafo de alcançabilidade. No segundo caso, os grafos de alcançabilidade de Redes de Petri com marcação e número finito de estados e transições são isomórficas às cadeias de Markov (MURATA, 1989), o que torna a resolução possível. Assim, o processo de resolução é semelhante à resolução de *CTMC*s (quando a resolução é feita através do grafo de alcançabilidade).

Avaliações de desempenho e de disponibilidade podem ser obtidas através de simulações e análises (estacionárias e transientes) efetuadas nas Redes de Petri construídas (BOLCH et al., 2006).

2.3 Injeção de falhas

Técnicas de injeção de falhas são muito úteis para testar a resiliência (DALZIELL; MCMANUS, 2004) de sistemas em geral. Com a injeção deliberada de falhas é possível observar o comportamento de sistemas perante tais ocorrências, bem como checar se os mecanismos utilizados para reparo são realmente efetivos (CLARK; PRADHAN, 1995). Com a utilização de injetores de falhas é permitido inserir tanto falhas no software, quanto no hardware.

As falhas de hardware podem ser de dois tipos: com contato e sem contato (HSUEH; TSAI; IYER, 1997). Na injeção de falha em hardware com contato, o injetor tem contato físico direto com o sistema alvo e pode produzir variações de corrente ou tensão no sistema testado. Na abordagem sem contato, o injetor não se conecta com o sistema alvo. Neste caso, para produzir as falhas, ele produz algum evento externo que acarreta a falha do componente desejado. Exemplos desse tipo de evento: uma radiação intensa ou uma interferência eletromagnética (HSUEH; TSAI; IYER, 1997).

Com a utilização de injeção de falhas em softwares é possível encontrar *bugs* que restaram da implementação. As falhas podem ser inseridas tanto em tempo de execução quanto de compilação (VOAS; MCGRAW, 1997). As falhas em tempo de execução podem ser (ZIADE; AYOUBI; VELAZCO, 2004):

- ***Time-out*** - Falha simples inserida após a contagem de um tempo qualquer. Geralmente ocasiona uma interrupção no sistema. O contador utilizado pode ser implementado em hardware ou em software.
- ***Exception/trap*** - Funciona como uma armadilha para o sistema. Neste caso não há controle de tempo para a execução da falha, pois usualmente esta é disparada quando instruções específicas são executadas.
- ***Code insertion*** - Consiste em inserir linhas de código que ocasionam falhas antes de determinadas instruções. Neste caso, o injetor é parte do próprio software que está sendo testado.

Em SOUZA et al. (2013) é exposta a ferramenta EucaBomber, que é capaz de inserir eventos de falha e reparo no ambiente de computação em nuvem Eucalyptus[4]. A abordagem de inserção utilizada é baseada em software, onde comandos para paralisação e *restart* de serviços são inseridos, baseados em variáveis aleatórias. Utilizando esta abordagem é possível verificar se os comportamentos descritos em modelos correspondem aos comportamentos reais. Ou seja, é uma abordagem factível para a validação de modelos.

2.4 Computação em Nuvem

Aliando capacidades de alta escalabilidade e desempenho, com um ambiente configurável e de disponibilidade, a computação em nuvem torna-se um ambiente propício para diversas aplicações (BUYYA; BROBERG; GOSCINSKI, 2011).

Esta seção contém conceitos fundamentais sobre o paradigma da computação em nuvem, com ênfase nos conteúdos utilizados para o desenvolvimento desta dissertação.

[4]https://www.eucalyptus.com/

Definição

Computação em nuvem é um modelo utilizado para habilitar o acesso ubíquo, conveniente e sob demanda a um conjunto de recursos computacionais compartilhados (rede, servidores, armazenamento, aplicações e serviços) que pode ser rapidamente provido e liberado com esforço gerencial mínimo (MELL; GRANCE, 2011a).

Características

Algumas características básicas da computação em nuvem foram absorvidas de outras tecnologias como: computação de alto desempenho, *grids* computacionais, virtualização, entre outros (GONG et al., 2010).

Em VAQUERO et al. (2009) são elencadas algumas características essenciais de nuvens públicas:

- Virtualização de recursos;
- Arquitetura baseada em serviços;
- Elasticidade;
- Modelo de pagamento baseado em consumo.

A virtualização de recursos é uma alternativa já bem consolidada em diversos cenários, como emulação de plataformas e ambientes que necessitam de maior desempenho computacional. Com a utilização dessa técnica ocorre a desagregação dos serviços da infraestrutura dos recursos físicos (disco rígido, memória RAM, processador, rede). Isto traz benefícios significativos, como agilidade na implantação e disponibilização de recursos, redução de custos adicionais (espaço, energia, refrigeração), além da recuperação de erros de forma mais rápida e menos custosa.

Com a arquitetura baseada em serviços, clientes podem utilizar serviços alocados em nuvem através dos*web browsers*. Unindo a computação em nuvem e a arquitetura baseada em serviços, aplicativos e demais recursos de TI podem ser oferecidos remotamente, como se estivessem localizados localmente. Vale a pena salientar que a arquitetura baseada em serviços permite monitorar em tempo real o uso dos recursos disponibilizados, colaborando assim para uma gerência mais eficiente a todo o sistema (LINTHICUM, 2009). Além disso, dependendo das interfaces definidas, pode-se ampliar consideravelmente o número de potenciais clientes. Recursos disponibilizados via navegador web, por exemplo, podem ser acessados tanto por computadores de mesa quanto por *smartphones* ou *tablets*.

Uma das propostas da computação em nuvem é a impressão de recursos infinitos que são disponibilizados sob demanda (GONG et al., 2010). Para suportar as mais variadas demandas em diferentes momentos, algumas propriedades, como escalabilidade e elasticidade, são necessárias. Com a escalabilidade é possível adequar o sistema a um aumento de carga adicionando uma

quantidade proporcional de recursos. A elasticidade é responsável pelo fornecimento e liberação de recursos em tempo real. Aliando as duas características, obtém-se um ambiente propício para vários propósitos.

Assim como o fornecimento de água, eletricidade e telefonia, a computação em nuvem adota o modelo de pagamento baseado em consumo. Isso é bastante plausível para usuários de TI, uma vez que permite a alocação de novos recursos somente quando necessário. Economia e redução do desperdício de recursos são consequências diretas desta característica.

A redução de custos possibilita que até mesmo pequenas empresas possam iniciar seus serviços mesmo sem um parque computacional robusto e ir crescendo seu aparato conforme a demanda. Características como virtualização e arquitetura baseada em serviços, aumentam a eficiência no uso de recursos, evitando desperdício, além de permitirem disponibilização em larga escala para vários tipos de clientes.

Componentes

A infraestrutura básica da computação em nuvem é dividida em quatro entidades fundamentais. São elas:

a) **Clientes** - Entidades que utilizam os recursos que são disponibilizados pelo provedor dos recursos. Essas entidades podem utilizar algum serviço que já é disponibilizado pelo provedor, ou, até mesmo, alocar suas necessidades no ambiente em nuvem;

b) **Rede** - Esta entidade viabiliza a comunicação entre os clientes e provedores da nuvem. Trata-se de uma entidade extremamente importante para os propósitos da computação em nuvem, pois possibilita a comunicação entre locais geograficamente distintos;

c) **Provedores** - Entidades que detêm o poder computacional que será disponibilizado para os clientes em forma de máquina virtual. Geralmente os grandes provedores dispõem de conjuntos de *datacenters* robustos para poder atender às necessidades de vários clientes ao mesmo tempo. Nos casos de nuvens privadas, o hardware provedor e cliente pode estar localizado dentro da mesma organização;

d) **Máquinas virtuais** - Constituem o modo como os recursos computacionais serão disponibilizados para os clientes. São unidades dotadas de poder computacional (armazenamento, processamento e comunicação) que podem ser instanciadas e removidas com facilidade.

Diagramando-se a estrutura descrita obtém-se um cenário como na Figura 2.12.

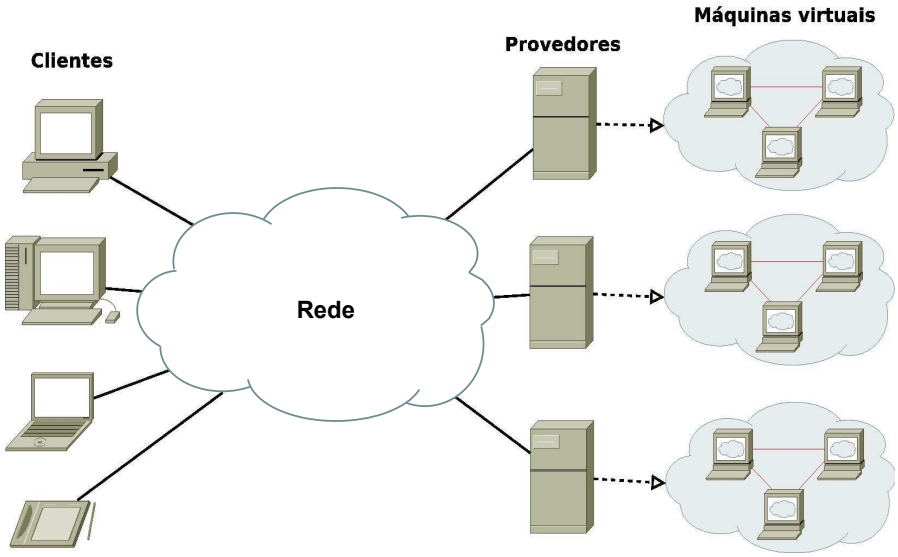


Figura 2.12: Componentes básicos da computação em nuvem

Modelos de implantação

Como visto, computação em nuvem consiste basicamente em conjuntos de recursos computacionais possuídos por uma entidade provedora, que disponibiliza este poder computacional para clientes conforme as suas necessidades. Existem diversas maneiras de se implantar a nuvem no hardware que irá suportá-la. Os modelos são adotados conforme a necessidade do negócio. Fatores como restrição de acesso, tipos de dados e níveis de visão são decisivos para escolha de um determinado modelo. Organizações podem preferir disponibilizar os recursos em nuvem apenas para determinados usuários (SOUSA; MOREIRA; MACHADO, 2009). Os possíveis modelos de implantação podem ser divididos em: nuvem privada, pública, comunitária e híbrida (MELL; GRANCE, 2011b).

Nuvem privada

Neste modelo de implantação, a infraestrutura da nuvem é disponibilizada para apenas uma organização. A gerência desta infraestrutura é desempenhada pela própria organização ou por terceiros. Da mesma forma os recursos físicos que suportam a nuvem podem ser propriedade da organização que usufrui destes recursos, ou de outra organização terceirizada.

Por possuir, geralmente, um pequeno porte, fica mais simples para traçar um perímetro de segurança para a nuvem, permitindo acesso externo apenas para usuários autorizados (BADGER; PATT-CORNER; VOAS, 2011). Esse modelo é comumente aplicado para atender organizações que necessitam de maior segurança nos dados e serviços alocados em nuvem. Uma de suas limitações é justamente a limitação de recursos disponíveis para a nuvem. Aplicações e serviços que serão alocados devem observar atentamente estas limitações, a fim de manter um desempenho aceitável a todo o sistema.

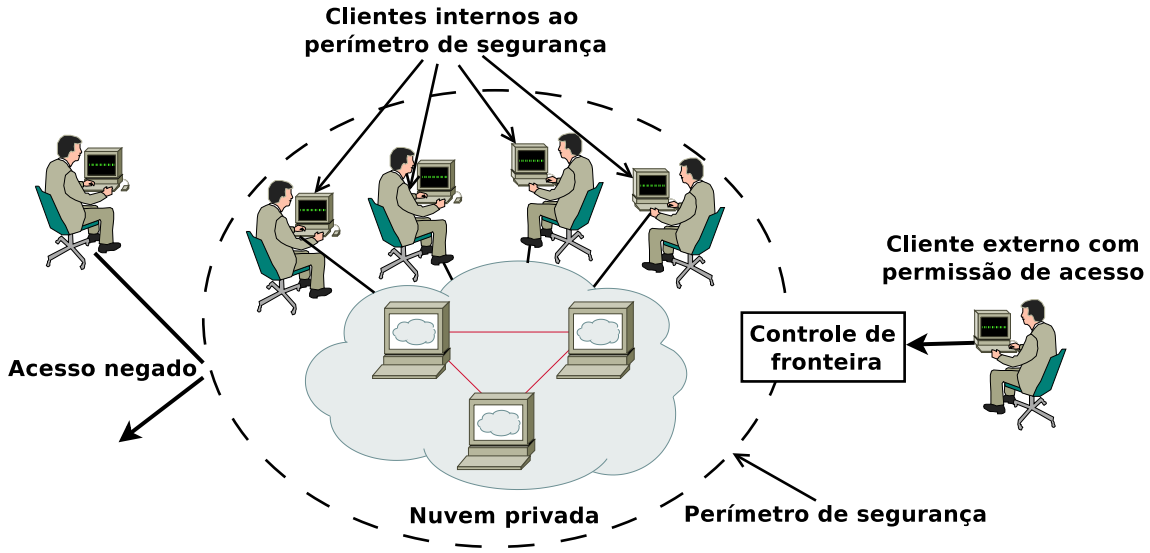


Figura 2.13: Nuvem privada, adaptado de (BADGER; PATT-CORNER; VOAS, 2011)

Nuvem comunitária

O princípio do modelo de implantação da "nuvem comunitária"(*community cloud*) é semelhante ao da nuvem privada. A diferença mais notória entre os dois modelos é o número de organizações clientes e provedoras envolvidas. Na nuvem comunitária, a nuvem é compartilhada entre organizações que possuem alguns objetivos compartilhados. Assim como na nuvem privada, a infraestrutura que comporta a nuvem pode residir tanto dentro das organizações parceiras como em um ambiente terceirizado. A estrutura dos provedores pode também ser compartilhada por mais de uma organização, desde que atenda aos requisitos dos clientes (MELL; GRANCE, 2011b).

Esta abordagem é baseada na técnica conhecida como ecossistemas digitais. Neste modelo de computação, as entidades que participam do sistema podem assumir mais de um tipo de papel. No caso da nuvem comunitária, as organizações podem ser clientes e provedoras ao mesmo tempo. Os recursos são distribuídos de modo a manter o sistema equilibrado (BRISCOE; MARINOS, 2009).

Nuvem pública

Neste modelo, a nuvem é disponibilizada para usuários em geral, para os mais variados propósitos. A infraestrutura que aloca a nuvem é de propriedade de organizações de grande porte, dotadas de robusto aparato computacional. Tal característica cria nos clientes a ilusão de recursos computacionais infinitos, constituindo assim um ambiente adequado para inúmeros tipos de aplicações e serviços. Os recursos destas nuvens são acessados por meio da Internet.

Os clientes de nuvens públicas alugam seus recursos adotando um modelo de pagamento por uso. As taxas são mensuradas pelo uso de processador por hora, ou por poder de armazenamento por dia, entre outros. Ao fim do contrato com o cliente, os recursos utilizados

por ele são liberados para serem alocados por outros clientes. Uma das desvantagens do uso da nuvem pública é a segurança (SRINIVASA; NAGESWARA; EKUSUMA, 2009). Por ser pública, usuários mal intencionados podem tentar atacar o sistema. Uma das medidas preventivas adotadas é a criação de políticas e regras impostas aos usuários por intermédio de técnicas de *login* e atribuição de privilégios (DYCK, 2011).

Nuvem híbrida

A nuvem híbrida é a junção de dois ou mais modelos de implantação numa mesma nuvem (MELL; GRANCE, 2011b). Neste tipo de nuvem, os recursos físicos são interligados por intermédio de uma interface padronizada, permitindo que eles atuem em conjunto e de forma transparente para o usuário final. Devido ao hibridismo de sua composição pode ser particularmente difícil de gerenciar estruturas de nuvem híbrida (BADGER; PATT-CORNER; VOAS, 2011). É necessário, portanto, delimitar bem as premissas de cada aplicação que será alocada em nuvem híbrida.

As motivações para uso de nuvens híbridas são diversas. Organizações com propósitos semelhantes podem compartilhar seus recursos em nuvem. Empresas que necessitam de grande poder computacional apenas em picos de requisição podem criar um canal para unir sua nuvem com um provedor de nuvem pública externo a fim de balancear cargas.

Arquitetura

A arquitetura baseada em serviços é característica inerente à computação em nuvem. Recursos são disponibilizados de maneira transparente para o usuário final, por intermédio de uma interface. Por possuir esta característica a computação em nuvem adota uma premissa de disponibilizar *tudo-como-serviço* ou *XaaS (everything-as-a-service)*. Os recursos disponibilizados sob a forma de serviço são divididos em três camadas principais: Software como serviço - SaaS (*software-as-a-service*), Plataforma como serviço - PaaS (*platform-as-a-service*) e Infraestrutura como serviço - IaaS (*infrastructure-as-a-service*). Estas camadas são organizadas conforme a Figura 2.14, onde as camadas inferiores estão mais próximas do nível de gerentes de infraestrutura, enquanto as camadas superiores estão mais próximas do nível de usuário finais. É importante lembrar que o modelo e nomenclatura das camadas aqui apresentados, mesmo sendo adotado pela maioria da literatura, não é absolutamente consolidado entre os autores de computação em nuvem.

SaaS

Pioneiro dentre os serviços em nuvem, software como serviço tem suas raízes nos Provedores de Serviços de Aplicações, ou Application Service Providers (ASP) (HURWITZ et al., 2009). Nos provedores de serviços de aplicações, as aplicações ficam hospedadas no servidor e consomem os recursos deste. Por já existir a mais tempo, e estar no mais alto nível

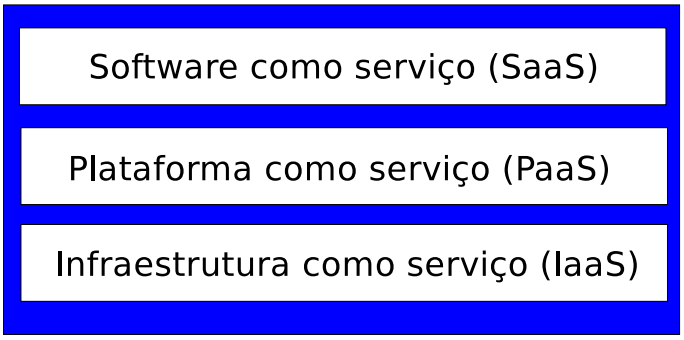


Figura 2.14: Modelo conceitual de computação em nuvem

da arquitetura da nuvem, SaaS é a camada mais ubíqua da computação em nuvem atualmente. Aplicativos como gerenciadores de correio eletrônico e processadores de texto já são utilizados com muita naturalidade na grande rede.

Em suma, SaaS consiste em aplicativos alocados remotamente, onde os recursos necessários para esta aplicação (ex. banco de dados, poder de processamento) são de responsabilidade do provedor. Os aplicativos são disponibilizados para os clientes por intermédio de interfaces.

Um dos aspectos inovadores que merece destaque em SaaS é que, além de poder usufruir de aplicações que são largamente disponibilizadas, como ferramentas de escritório online, usuários finais podem também alocar seus próprios aplicativos em nuvem. Desta forma pequenas empresas e até mesmo entusiastas podem divulgar suas aplicações e serviços dentro da grande rede. Outro possível contexto para SaaS é a alocação de aplicativos para utilização interna à organização. Isso desatrela o serviço da plataforma que o utilizará, ou seja, não é mais necessário instalar e consumir o recurso de cada dispositivo cliente.

PaaS

A plataforma como serviço é muito útil para desenvolvedores de aplicações, pois constitui um ambiente bastante útil e versátil para testes, simulações e desenvolvimento de aplicações. Hurwitz et al resumem PaaS como uma plataforma de computação dotada de capacidades de desenvolvimento, *middleware* e implantação (HURWITZ et al., 2009). A utilização de plataforma como serviço permite a desenvolvedores simularem e desenvolverem aplicações de alto custo computacional dentro de um ambiente que é escalável e elástico, conforme a demanda de cada aplicação.

CIURANA (2009) expõe uma das plataformas como serviço mais conhecida atualmente, o Google App Engine [5], mostrando algumas características e técnicas de desenvolvimento para esta plataforma. Existe um conjunto de funcionalidades agregadas ao Google App Engine que denotam bem o potencial de plataformas como serviço, são elas:

- Permitir aos desenvolvedores construir aplicações públicas ou de interesse pessoal com suporte a transações, autenticação uniforme e disponibilidade e escalabilidade

[5] Google App Engine - http://code.google.com/appengine/

robustas.

- Aplicativos desenvolvidos com a plataforma podem ser disponibilizados como SaaS e consumidos diretamente pelos navegadores web dos clientes.
- Possibilidade de integrar e consumir *web services* de plataformas como serviço (*PaaS*) terceirizadas.

IaaS

IaaS, ou infraestrutura como serviço, é a camada da arquitetura de computação em nuvem que está localizada mais próxima ao gerentes de infraestrutura. Esta é a camada responsável por disponibilizar poder computacional (rede, armazenamento, processamento) como serviço. Desta forma, organizações podem alugar recursos computacionais e alocar seus respectivos serviços.

A infraestrutura como serviço traz consigo novas possibilidades interessantes para o mercado, como suporte a empreendimentos em estado inicial, onde organizações não dispõem de um parque computacional robusto para executar as aplicações. Além de também tornar-se ótima alternativa para empresas que obedecem a alguma sazonalidade, onde são necessários maiores recursos computacionais apenas em períodos determinados. Isso acarreta uma economia acentuada em aquisição de recursos físicos e em gastos adicionais, como energia, espaço físico e refrigeração do ambiente.

O intuito da IaaS é disponibilizar um conjunto de recursos com possibilidades de expansão e contração, com escalabilidade sob demanda, tolerância a falhas e hospedagem de sistemas operacionais (LENK et al., 2009). Este *pool* é construído por intermédio direto da virtualização. Por utilizar tal tecnologia, a IaaS acaba evitando desperdício de recursos físicos, além de agregar a si as demais vantagens da virtualização.

Além disso, é possivel utilizar IaaS tanto em nuvens privadas quanto em nuvens públicas. Existem serviços de IaaS amplamente disponíveis através da Internet. Uma das nuvens IaaS mais conhecidas é a Amazon Elastic Compute Cloud (Amazon EC2) [6]. Nela é possivel alugar e personalizar *VM*s para rodar os serviços desejados. Para utilização de nuvens IaaS privadas, é necessário a construção de todo ambiente de IaaS no hardware disponível.

Tecnologias Associadas

Conforme visto, a computação em nuvem engloba outras diversas tecnologias, absorvendo suas qualidades a fim de formar um paradigma robusto e bastante útil para os mais variados propósitos. Nesta subseção serão destacadas algumas tecnologias, associadas ao paradigma de computação em nuvem, utilizadas para o desenvolvimento desta dissertação.

[6] Amazon Web Services - http://aws.amazon.com/pt/ec2/

Virtualização

A virtualização é uma tecnologia fundamental para conseguir a elasticidade e escalabilidade prometida pela computação em nuvem. De maneira simples, a virtualização é a capacidade de emular recursos de hardware em dispositivos que possuem algum poder computacional suficiente para esta emulação. Existem vários possíveis produtos de virtualização, como máquinas virtuais, VPNs e virtualização de armazenamento. A computação em nuvem utiliza-se da maioria destes produtos. Dentre eles, o mais presente em computação em nuvem são as máquinas virtuais.

Em síntese, a virtualização possui os subsídios necessários para emular capacidades de armazenamento, processamento e rede, dentro de outro hardware. Com isto, é possível criar unidades de computação dedicadas para alguns serviços determinados, unidades estas, que são denominadas de máquinas virtuais. Com o uso de máquinas virtuais, é possível que vários sistemas operacionais funcionem ao mesmo tempo e de maneira independente, alocando o mesmo hardware.

Outro ponto importante em virtualização é a virtualização de rede. A rede virtual é o meio de comunicação entre máquinas virtuais. Com ela é possível deslocar a execução de um processo de uma máquina para outra. Além disso, ainda é possivel abstrair a camada de serviço da infraestrutura, ou seja, podem-se criar novos modelos de rede, utilizando a mesma infraestrutura física de comunicação (CARAPINHA; JIMENEZ, 2009).

Uma das aplicações mais interessantes para virtualização, é a virtualização de servidores. Esta técnica consiste na hospedagem de vários sistemas operacionais independentes em uma única máquina física (FLORAO et al., 2008). Essa é a abordagem real para reduzir a subutilização de servidores físicos, além de prover gerenciamento centralizado para conjunto de servidores.

Hypervisor

O hypervisor, ou VMM (*Virtual Machine Monitor* - Monitor de máquinas virtuais), consiste numa camada inserida entre o hardware e o sistema operacional de forma a possibilitar que os vários sistemas operacionais funcionem em um mesmo hardware. Esses sistemas operam de forma independente. Mesmo localizados na mesma máquina física, as instruções passam pelo hypervisor antes de encontrar o hardware. Desta maneira, os sistemas são acrescidos de características de confiabilidade, além de um bom isolamento de falhas (GOLDBERG, 1974).

Por permitir alocação de vários sistemas no mesmo hardware, o hypervisor torna-se essencial para os propósitos da computação em nuvem. Dois hypervisors bem conhecidos e aplicáveis em ambientes de computação em nuvem são o Linux KVM [7] e o Xen Hypervisor [8]. O primeiro deles foi aplicado no trabalho corrente.

[7] Kernel Based Virtual Machine - http://www.linux-kvm.org
[8] Xen Hypervisor - http://xen.org/products/xenhyp.html

VIM - Virtual Infrastructure Manager

Dentro do contexto de IaaS, onde existem diversos servidores físicos e recursos virtuais para serem gerenciados, é pertinente a presença de softwares que auxiliem nessa gerência, a fim de manter o controle de todo o sistema centralizado. E esse é justamente o papel do Gerenciador de Infraestrutura Virtual ou VIM (*Virtual Infrastructure Manager*). Geralmente, esse software gerente utiliza-se de um conjunto de ferramentas para manipular e monitorar o comportamento de toda a nuvem. Ele também é conhecido como software de orquestração da nuvem.

Dentre as tarefas do VIM, podem-se destacar (SOTOMAYOR et al., 2009):

- Prover uma visão uniforme e homogênea dos recursos virtualizados, independentemente de qual plataforma de virtualização está em uso.
- Gerenciar todo o ciclo de vida das máquinas virtuais, incluindo a criação de redes virtuais de forma dinâmica para interligação de grupos de máquinas virtuais e gerenciar os requisitos de armazenamento para cada uma (implantação de imagens de disco das VMs).
- Suportar políticas configuráveis para alocação de recursos para alcançar objetivos específicos (alta disponibilidade, consolidação de servidores para minimizar o consumo de energia, entre outros).
- Adaptar-se para mudanças nas necessidades de recursos da organização, como picos de requisição e mudanças em recursos físicos defeituosos.

OpenNebula IaaS

A maioria dos experimentos e modelos apresentados neste trabalho podem ser generalizados e adaptados, de modo geral, para qualquer ambiente de nuvem. Mas, a base dos estudos foi construída a partir da plataforma OpenNebula[9]. Dentre os Gerentes de Infraestrutura disponíveis, optou-se pelo OpenNebula devido a sua fácil implantação nas máquinas provedores, a característica de centralização das atividades administrativas em uma só máquina, denominada de *FrontEnd* (a arquitetura será abordada a seguir) e a capacidade da migração de máquinas virtuais entre os nós, esta última não tinha sido encontrada em outros Gerentes de Infraestrutura até o desenvolvimento deste trabalho.

O OpenNebula é uma nuvem *IaaS* baseada numa arquitetura clássica de cluster, conforme a Figura 2.15. O componente *FrontEnd* atua como um gerente do serviço, rodando os processos necessários para comunicação com os nós e monitorando o ambiente por completo. O principal processo do OpenNebula é o *ONED*. No *FrontEnd*, é instalado o pacote do OpenNebula propriamente dito. Além disto, todas as operações administrativas da nuvem, como definição e

[9] http://opennebula.org/

instanciação de VMs, cadastro de novos nós e imagens, entre outras, devem ser executadas por intermédio do *FrontEnd*.

Os *Clusters Nodes* cedem os recursos computacionais para a execução das máquinas virtuais. As imagens das VMs podem ser guardadas em dispositivos de armazenamento de dados (HDs, SANs). Contudo, estes devem estar disponíveis para acesso via rede, através do protocolo SSH. Esta capacidade permitirá que as imagens das VMs sejam transferidas entre os nós do ambiente, possibilitando assim a instanciação e migração delas. Por fim, é necessário que todos esses componentes estejam interligados por uma rede de comunicação de dados.

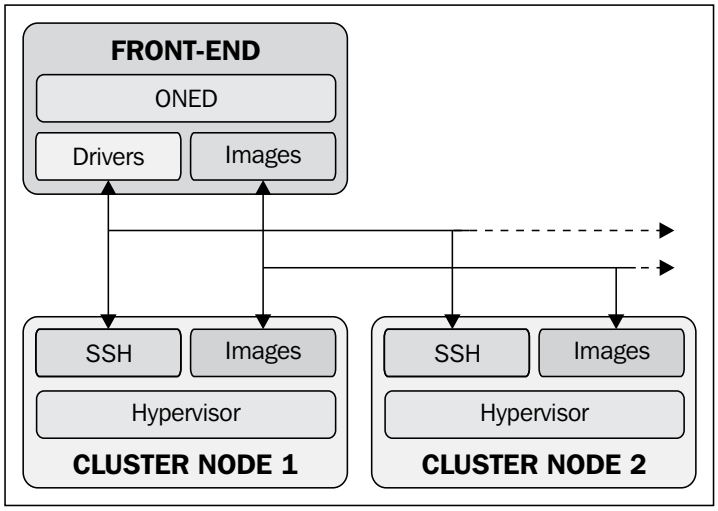


Figura 2.15: Arquitetura OpenNebula. Extraído de (TORALDO, 2012)

O processo de instalação e configuração do OpenNebula pode ser encontrado em sua documentação oficial[10]. Outra alternativa é a utilização do livro **OpenNebula 3 Cloud Computing** (TORALDO, 2012), que também contém detalhes de instalação para vários *hypervisors* diferentes.

Live Migration

A operação de *live migration* constitui-se em uma ferramenta poderosa para administradores de *datacenters*. Se uma máquina física necessitar ser removida do ambiente, é possível migrar as VMs que consomem seus recursos para outro(s) nó(s) com um tempo mínimo de interrupção no serviço, habilitando, assim, a mudança, de modo a reduzir ao máximo as consequências à disponibilidade e responsividade oriundas da operação em questão (CLARK et al., 2005).

O algoritmo padrão de migração de máquinas virtuais do *hypervisor* KVM (utilizado no ambiente de testes) é o algoritmo de pré-cópia. Sua execução possui 6 (seis) estágios diferentes, descritos na Figura 2.16.

O algoritmo de *live migration* ***precopy*** consiste na migração sucessiva de páginas da memória RAM da VM de um nó para outro. Esse processo é realizado de modo iterativo, em

[10]http://opennebula.org/documentation:archives

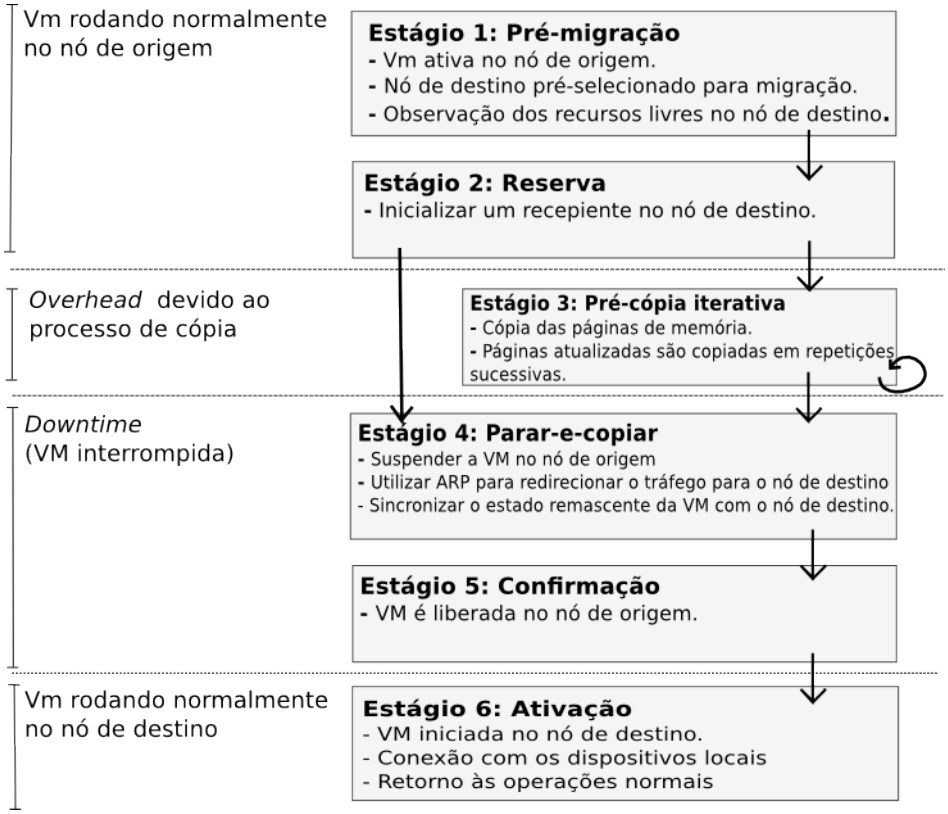


Figura 2.16: Fases do processo de live migration. Adaptado de (CLARK et al., 2005)

cada iteração as páginas de memória modificadas são transferidas para o nó que está recebendo a VM. O estágio seguinte é desviar o fluxo de requisições para o nó alvo da migração. Após a sincronização, a VM está liberada para voltar ao funcionamento normal.

Além do algoritmo de ***precopy***, existem outros algoritmos para realizar a migração de máquinas virtuais. O algoritmo ***post-copy*** - consiste em transferir imediatamente o estado do processador, para permitir que a VM possa começar a executar na máquina física de destino. Após ter transferido o estado do processador é que as páginas da memória RAM começam a ser transferidas pela rede (HINES; DESHPANDE; GOPALAN, 2009). Outra abordagem é a ***full trace and replay***, que consiste em gerar um *trace* de execução na máquina física de origem, um algoritmo de sincronização para orquestrar a VM de origem e a de destino até que se alcance um estado consistente (LIU et al., 2009). Outra abordagem possível é a de *parar-e-copiar*, onde a execução da VM é interrompida e ela é enviada via rede para a máquina física de destino.

2.5 Envelhecimento e rejuvenescimento de software

O envelhecimento de software pode ser definido como uma degradação gradual do estado do software durante o seu período de execução (GROTTKE; MATIAS; TRIVEDI, 2008). As causas do envelhecimento estão ligadas à acumulação de erros e *bugs* enfrentados durante a sua execução. Tal comportamento pode ser nocivo ao sistema ao ponto de levá-lo a falhas generalizadas ou a travamentos, afetando assim sua disponibilidade (HUANG et al., 1995). Por

muitas vezes, a presença de erros e problemas relativos ao envelhecimento é imperceptível. As consequências do envelhecimento vão desde erros numéricos e inconsistência de dados até a exaustão de recursos do sistema operacional (GROTTKE; MATIAS; TRIVEDI, 2008).

Por ser um processo gradual, o envelhecimento de software é mais perceptível em sistemas que precisam rodar por um longo intervalo de tempo e de modo contínuo (CASTELLI et al., 2001). Sendo assim, são necessários mecanismos para monitoramento de efeitos de envelhecimento para entender esse comportamento em cada sistema específico (AVRITZER; WEYUKER, 1997).

Sobre a disponibilidade de sistemas afetados por envelhecimento de software, uma das métricas de interesse é o *Time To Aging Related Failure* (**TTARF**)[11] mostra uma estimativa do tempo necessário para a ocorrência de uma falha relacionada ao envelhecimento de software (HUANG et al., 1995). Alguns estudos mostram que esta métrica está ligada à carga de trabalho que é submetida ao sistema objeto de estudo (BAO; SUN; TRIVEDI, 2005).

Os efeitos de envelhecimento de software em computação em nuvem foram investigados em (ARAUJO et al., 2011). Nesse trabalho são exibidos resultados relativos a experimentos de envelhecimento executados na plataforma *Eucalyptus*. Os resultados mostram que, conforme o passar do tempo, há um consumo exagerado de recursos computacionais devido a vazamentos de memória (*memory leaks*).

O processo de remoção dos efeitos acumulados por envelhecimento de software é denominado rejuvenescimento de software. O rejuvenescimento consiste na aplicação de métodos para evitar que os efeitos relativos ao envelhecimento levem o sistema a falhar. Trata-se de uma estratégia proativa para gerenciamento deste tipo de falha, onde as operações realizadas pretendem remover os problemas acarretados, acumulados pelo envelhecimento (KOURAI; CHIBA, 2011). Exemplos de operações de rejuvenescimento são *restart* de serviços e *reboot* de sistemas operacionais (MELO et al., 2013). Essas atividades permitem que os erros acumulados sejam removidos através da criação de novas instâncias (processos) de softwares que estavam previamente envelhecidos (MATIAS et al., 2006).

No contexto de computação em nuvem, alguns trabalhos publicados apresentam mecanismos e modelos para realização de rejuvenescimento de software. Uma das abordagens interessantes é a utilização de migração de máquinas virtuais para suportar o rejuvenescimento com uma interrupção de serviços mínima (MACHIDA; KIM; TRIVEDI, 2010).

[11] Tempo para falha relacionada ao envelhecimento

3

Experimentos de envelhecimento e rejuvenescimento de software em nuvens privadas

Conforme visto na Seção 2.5, o envelhecimento de software é um dos grandes desafios a ser enfrentado para alcançar alta disponibilidade em sistemas computacionais. Assim sendo, neste capítulo será apresentado um estudo experimental de envelhecimento e rejuvenescimento de software em plataformas de computação em nuvem OpenNebula, utilizando o *KVM* como *hypervisor*. O objetivo principal é observar a ocorrência de envelhecimento de software na plataforma base de estudo e verificar se o rejuvenescimento que é proposto é realmente efetivo. Os resultados experimentais do rejuvenescimento de software indicam que a utilização de migração de máquinas virtuais constitui um mecanismo de suporte ao rejuvenescimento adequado para o envelhecimento detectado.

É importante salientar que existem outras técnicas para a realização do rejuvenescimento de software em ambientes de computação em nuvem. Em ARAUJO et al. (2011), por exemplo, é apresentado um mecanismo de rejuvenescimento baseado em séries temporais e predição. Porém, para o escopo deste trabalho utilizou-se apenas o rejuvenescimento baseado em migração de máquinas virtuais, pois esse mecanismo mostrou-se adequado para o ambiente OpenNebula e KVM utilizado na experimentação realizada.

3.1 Arquitetura do ambiente de testes

Para a realização dos experimentos considerou-se um ambiente de testes real composto por quatro máquinas físicas. A organização destas máquinas é exposta na Figura 3.1 e a configuração das máquinas é apresentada na Tabela 3.1. Trata-se de um ambiente de nuvem com o *VIM* OpenNebula 3.6. Os nós da nuvem utilizam o *KVM*[1] (versão 1.0) como *hypervisor*.

[1] http://www.linux-kvm.org/

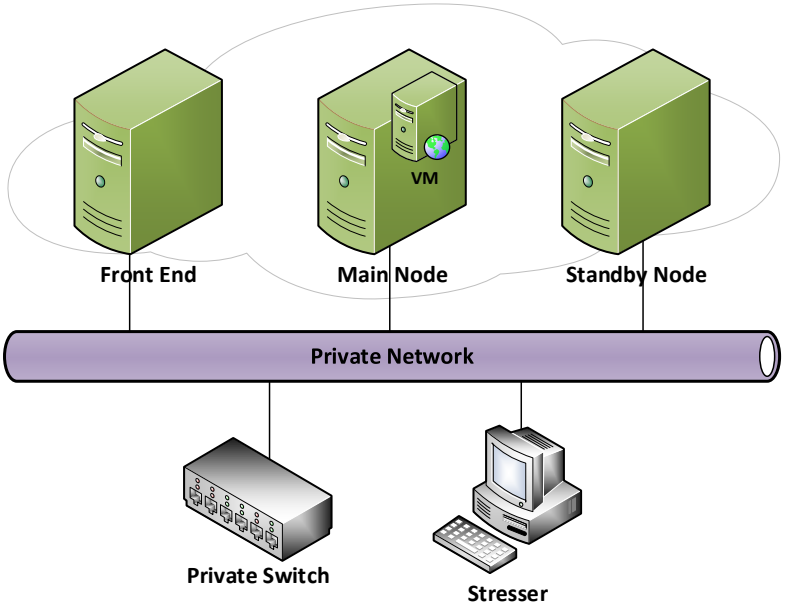


Figura 3.1: Ambiente de testes para os experimentos de aging

Tabela 3.1: Componentes do ambiente de testes do experimento de envelhecimento

Nome	Descrição	Processador	RAM	SO
FrontEnd	Gerente da nuvem	Intel Core i3 - 4 núcleos de 3.10 GHz	4 GB	Ubuntu Server 12.04 (kernel 3.2.0-23)
Main Node, Standby Node	Nós executores da nuvem	Intel Core i3 - 4 núcleos de 3.10 GHz	4 GB	Ubuntu Server 12.04 (kernel 3.2.0-23)
Stresser	Monitor e *stresser* da nuvem	Intel Core 2 Quad - 4 núcleos de 2.66Ghz	4 GB	Ubuntu Desktop 12.10 (kernel 3.5.0-36)

A VM utilizada para os testes foi customizada com o Sistema Operacional Ubuntu Server[2] 12.04 e um servidor Web Apache2[3] que hospeda uma página HTML simples. Na arquitetura, o *Main Node* assume o papel de nó principal alocando a única VM do ambiente. O *Standby Node*, por sua vez, é utilizado como um nó reserva que pode ser empregado para propósitos de migração de *VM*s na plataforma. O *FrontEnd* atua como gerente de toda a infraestrutura virtualizada.

3.2 Metodologia dos experimentos

Com o intuito de detectar indícios de envelhecimento de software e checar a efetividade do mecanimos de rejuvenescimento proposto para a plataforma estudada, foi conduzido um estudo experimental baseado em alguns trabalhos previamente publicados (ARAUJO, 2012; ARAUJO

[2]http://www.ubuntu.com/
[3]http://httpd.apache.org/

et al., 2011; MATOS et al., 2012). A visão geral da estratégia adotada pode ser vista na Figura 3.2. É importante ressaltar que os testes são realizados com o intuito de revelar características de envelhecimento e rejuvenescimento de software na plataforma, e não a descoberta de tempos de falha (*TTARF*).

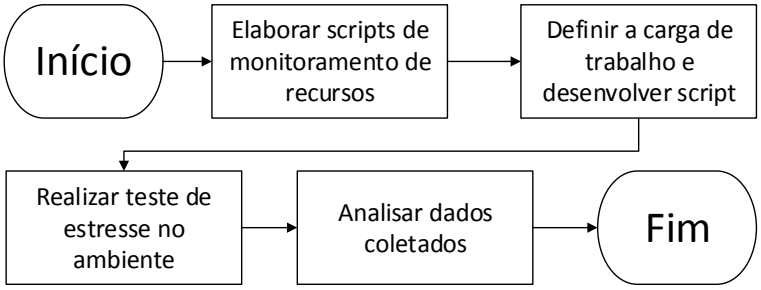


Figura 3.2: Estratégia adotada para os experimentos

Para investigar a ocorrência de envelhecimento de software, é necessário monitorar os prováveis recursos que sofrerão com o envelhecimento. Sendo assim, métodos de monitoramento precisam ser elaborados, visando gerar arquivos de saída ou relatórios que serão analisados a posteriori. A ocorrência natural de envelhecimento pode demorar muito tempo para ocorrer, o que tornaria o processo de experimentação inviável. No entanto, é possível definir cargas de trabalho que estressem o sistema de modo acelerado para que o surgimento do envelhecimento seja mais rápido.

Monitoramento do sistema

O primeiro passo para a execução do experimento de envelhecimento é a definição de mecanismos e métodos de monitoramento dos recursos computacionais do ambiente. Pode-se utilizar alguma ferramenta já existente para esse monitoramento, desde que esta permita observar o consumo dos recursos computacionais de interesse. Além disso, é importante que os scripts ou ferramentas utilizadas para monitorar o ambiente gerem arquivos de *log* ou relatórios para que seja possível realizar análise dos resultados obtidos.

Utilizando-se da linguagem de programação *Shell Script*, foram desenvolvidos scripts de monitoramento para os principais componentes do ambiente. A *Shell Script* torna-se adequada por ter acesso direto aos softwares nativos do Linux, o que facilita o desenvolvimento dos scripts de monitoramento (MITCHELL; OLDHAM; SAMUEL, 2001). As ferramentas utilizadas foram: `ps` - para monitoramento do processo da VM, `mpstat` - para monitoramento do status do processador e `free` - para monitoramento do estado da memória RAM. Vale a pena salientar que todas elas estão disponíveis nativamente no Sistema Operacional Ubuntu[4]. Optou-se por

[4]http://www.ubuntu.com/

monitorar apenas estes recursos porque estudos publicados (MATOS et al., 2012) mostram que a manifestação de envelhecimento neles é mais acentuada.

Carga de trabalho

Uma das tarefas mais críticas em experimentos de envelhecimento de software é a definição da carga de trabalho que será enviada ao sistema. A finalidade desta carga de trabalho deve ser estressar o sistema. Como o intuito dos testes é acelerar os efeitos relativos ao envelhecimento de software, a carga selecionada não precisa satisfazer os moldes de situações reais (ARAUJO et al., 2011). O objetivo dos experimentos é encontrar indícios de envelhecimento de software a partir da repetição sucessiva de operações. Cargas de trabalhos mais intensas podem fazer com que o sistema manifeste o envelhecimento mais precocemente. Além disto, os tempos de ocorrência dos eventos podem ser acelerados a fim de que o envelhecimento ocorra mais cedo. Se a carga de trabalho não for bem definida, o processo de envelhecimento pode demorar a acontecer, transformando assim o experimento realizado em um trabalho inócuo.

Assim sendo, decidiu-se adaptar a carga de trabalho utilizada em um experimento prévio de envelhecimento na plataforma *Eucalyptus* (MATOS et al., 2012) na tentativa de encontrar indícios de envelhecimento no ambiente de testes apresentado. A carga de trabalho selecionada consiste numa repetição sucessiva de operações para anexação e desmonte de discos virtuais (de 1GB) na VM do ambiente, o pseudo-código do script é apresentado no Algoritmo 1. As operações de anexação e desmonte dos discos virtuais foram realizadas a partir da ferramenta ***onevm*** (BLANCO; SOTOMAYOR, 2010), que é fornecida pelo próprio *OpenNebula*. Optou-se pela utilização da anexação de 15 discos virtuais porque, em testes realizados previamente, observou-se que quantidades menores do que esta não seriam suficientemente relevantes para destacar os efeitos de envelhecimento de software. O tempo de espera de 15 segundos é relativo à espera pelo software de anexação e desmonte de discos, que possui tempo de confirmação após estas operações.

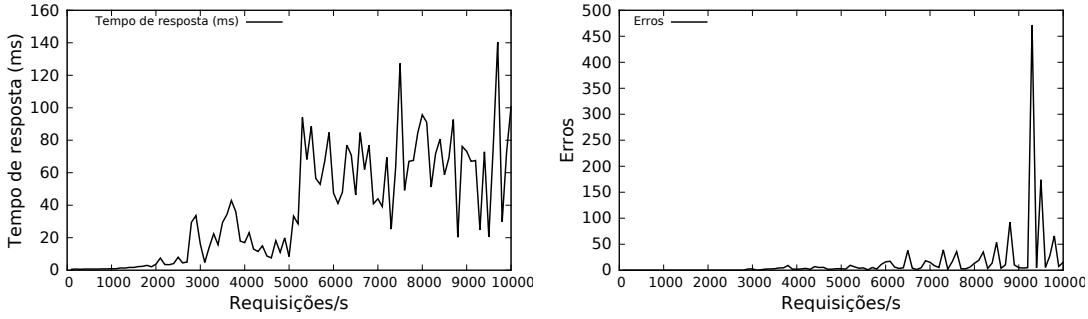
Algoritmo 1 Carga de trabalho para envelhecimento

```
loop
    while DiscosAnexados < 15 do
        Anexar(Disco1GB);
        Esperar(15 segundos);
    end while
    while DiscosAnexados <= 1 do
        Desmontar(Disco);
        Esperar(15 segundos);
    end while
end loop
```

Conforme visto na Seção 2.5, os efeitos causados pelo envelhecimento de software são refletidos na degradação do desempenho de aplicações, que os sistemas que enfrentam

envelhecimento hospedam. Portanto, além de submeter a carga de trabalho de envelhecimento para o sistema, decidiu-se gerar uma carga de trabalho para submeter ao servidor Web alocado na VM. As requisições Web devem ser enviadas para o servidor por intermédio de uma ferramenta *benchmark*. O intuito é observar as oscilações de comportamento do tempo de resposta e a quantidade de erros do servidor Web alocado no sistema que enfrenta o envelhecimento de software.

Para determinar qual carga seria submetida ao servidor web, fez-se necessário um teste de capacidade no servidor Web alocado na VM. Para o teste, submete-se uma carga de requisições com taxa crescente a fim de descobrir qual a taxa de requisições que o servidor é capaz de responder sob condições aceitáveis (erros quase nulos e tempo de resposta mínimo). Assim, é possível definir uma carga de trabalho com taxa de requisições constante que leve o sistema até uma situação de estresse, mas que não produza degradação nos resultados do desempenho. Para alcançar isto, utilizou-se a ferramenta ***Autobench***[5] que é um *wrapper* escrito em *Perl* para o *benchmark* ***httperf*** (MOSBERGER; JIN, 1998).



(a) Tempo de resposta (ms) (b) Erros

Figura 3.3: Teste de capacidade na VM

A Figura 3.3 mostra que a VM pôde suportar 2000 requisições por segundo com desempenho aceitável e erros quase nulos. Após esta taxa de requisições (2000 req/s), é possível notar que o comportamento do desempenho do servidor Web (erros e tempo de resposta) começa a ser degradado. Conforme explicado anteriormente, neste passo deve-se definir a máxima carga de requisições Web que o servidor seja capaz de responder sem degradação do seu desempenho. Percebe-se então, através da Figura 3.3, que a carga de trabalho que atende a esses requisitos é a de 2000 requisições por segundo.

Para o *benchmark* utilizado, o tempo de resposta é apresentado em milissegundos e representa o tempo entre o envio do primeiro byte da requisição e o recebimento do primeiro byte da resposta. Os erros representam o montante de erros observados durante o envio da carga de trabalho. Eles podem ser *timeout* de conexão e de *socket*, conexão recusada e outros.

Definiu-se, então, que a carga de trabalho utilizada para os experimentos deve ser composta pela carga de envelhecimento e pela carga submetida ao servidor web. Assim, será

[5]http://www.xenoclast.org/autobench/

possível observar a degradação de desempenho acarretada pelo envelhecimento de software. Para evitar que a carga do servidor Web atrapalhe os testes de envelhecimento, exaurindo os recursos, optou-se pela carga de 2000 req/s, pois a VM mostrou-se apta a atender esta carga sob condições normais (via teste de capacidade realizado). O ***Autobench*** foi configurado para realizar as requisições Web com a taxa constante de 2000 requisições por segundo. Relembrando que a carga de trabalho final não precisa ser baseada em cargas de trabalho reais, basta que ela seja suficiente para mostrar os indícios de envelhecimento num espaço de tempo aceitável (ARAUJO, 2012). A visão geral da carga submetida é descrita na Figura 3.4.

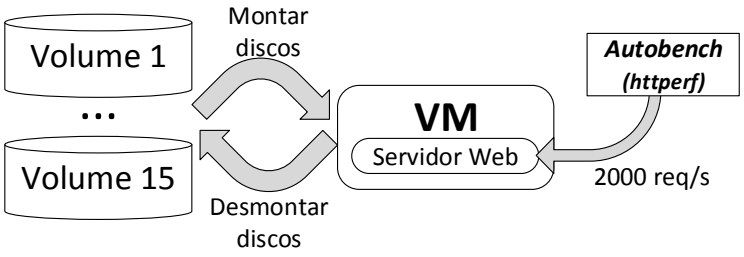


Figura 3.4: Geração da carga de trabalho

Além disto, para evitar interferências recorrentes na plataforma de estudo, definiu-se que o monitor do sistema operasse com intervalo de tempo de 30 segundos entre cada observação.

Estratégia de rejuvenescimento

A carga de trabalho utilizada nos experimentos consiste em operações de anexação e desmonte de discos virtuais. A carga é enviada pela ferramenta ***onevm*** diretamente para o gerente de máquinas virtuais (*hypervisor*) do nó físico que hospeda a VM. Entende-se que os efeitos de envelhecimento deverão ocorrer no componente de software responsável por lidar com a carga de trabalho submetida. No caso deste experimento, o responsável por lidar com as operações de anexação e desmonte dos discos da VM é o *hypervisor*, confirmando assim os resultados apresentados em (ARAUJO et al., 2011).

Para mitigar os efeitos relativos ao envelhecimento de software é necessário definir um mecanismo de rejunescimento para o *hypervisor KVM*. Além disso, é desejável que o modo como o rejuvenescimento é realizado melhore a disponibilidade do sistema. Como as operações de rejuvenescimento acarretam downtime no sistema que será rejuvenescido, faz-se necessário definir políticas que realizem essas operações de modo a maximizar a disponibilidade estacionária do sistema.

Rejuvenescimento habilitado por migração de VMs

O ambiente *OpenNebula* em conjunto com o *hypervisor KVM* permite a execução de migrações de máquinas virtuais entre nós físicos com tempos de interrupção mínimos nos serviços. Esta operação é denominada *live migration* (ver Seção 2.4). Tomando como base esta funcionalidade e as características específicas do envelhecimento de software encontrado, é possível definir uma política de rejuvenescimento baseada em migrações de máquinas virtuais que vise diminuir o *downtime* sofrido pelo sistema.

A estratégia de rejuvenescimento de software por agendamento de migrações está descrita na Figura 3.5. Tal mecanismo foi baseado nos trabalhos (MELO et al., 2013; MACHIDA; KIM; TRIVEDI, 2013). Esse método é concebido considerando o mesmo conjunto de componentes (*VM, FrontEnd, MainNode* e *StandbyNode*) e a mesma organização apresentada no experimento anterior (Figura 3.1).

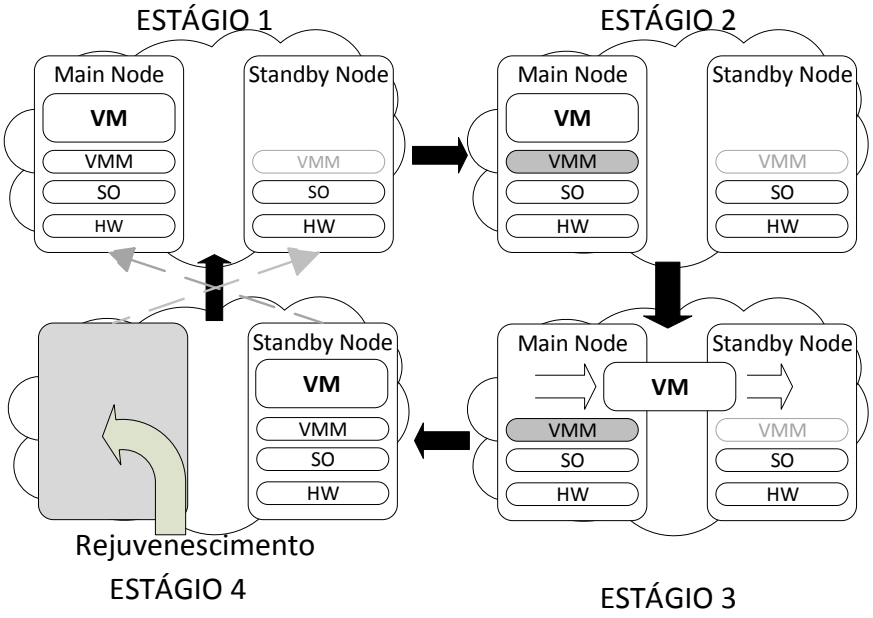


Figura 3.5: Estratégia adotada para o rejuvenescimento

A metodologia de rejuvenescimento está descrita na Figura 3.5 e se resume nas seguintes atividades:

a) O estado inicial do sistema não contém nenhum efeito de envelhecimento acumulado. Aplicações e serviços alocados na *VM* têm desempenho aceitável.

b) Com o passar do tempo, os efeitos de envelhecimento começam a aparecer. Geralmente, os indícios observados são: degradação no desempenho das aplicações e acúmulo de erros. Logo, é necessário realizar o rejuvenescimento de software a fim de evitar falhas vindouras. Então, o processo de migração de VMs é iniciado.

c) Para evitar que o rejuvenescimento do *VMM* paralise o serviço que está sendo realizado pela VM, deve-se migrar esta para o outro nó (*Standby Node*). É importante ressaltar que durante o processo de migração ocorre uma interrupção nos serviços da *VM*, fato esse que pode ser relevante em estudos de disponibilidade. Mais informações sobre o processo de *live migration* podem ser encontradas na Seção 2.4.

d) Depois que a migração da VM é completada, o nó que estava envelhecido (*Main Node*) pode ser submetido ao processo de rejuvenescimento de software.

e) O nó que antes era o *Standby Node* assume o papel de *Main Node*, pois estará portando a VM, e o antigo *Main Node* torna-se *Standby Node*.

O método proposto pretende evitar interrupções mais longas no serviço disponibilizado pela VM, movendo-a para outro nó antes de rejuvenescer. É importante salientar que no nó reserva (*Standby Node*) o *hypervisor* (VMM) está ativo, porém não está em operação, pois nenhuma VM está sendo executada. Por não estar resolvendo operações e tratando nenhuma VM, não é possível afirmar que ele envelhece por algum motivo, pois a carga de trabalho está intimamente ligada aos efeitos de envelhecimento de software (BAO; SUN; TRIVEDI, 2005). Para garantir que o nó reserva esteja livre de envelhecimento de software basta reiniciá-lo por completo antes de realizar a migração das máquinas virtuais.

Organização de tempo do experimento

A proposta da utilização de migração de máquinas virtuais como mecanismo de rejuvenescimento foi apresentada em (MELO et al., 2013; MACHIDA; KIM; TRIVEDI, 2013). Em ambos os casos, o comportamento é representado por intermédio de modelos analíticos.

Visando checar os efeitos de envelhecimento de software e a efetividade do método de migração de máquinas virtuais como mecanismo de rejuvenescimento de software, esta seção apresenta alguns experimentos realizados em uma plataforma real. Os experimentos foram conduzidos considerando a arquitetura e carga de trabalho apresentadas na seção 3.2.

A metodologia aplicada foi dividida em três etapas principais:

a) **Fase de estresse** - Consiste na aplicação do testes de estresse realizados diretamente na plataforma, mantendo a carga de trabalho apresentada na Figura 3.4. Esta fase tem o intuito de acelerar o aparecimento de indícios de envelhecimento de software na plataforma.

b) **Fase de espera** - Após a fase de aceleração, a carga de trabalho é interrompida. O objetivo desta fase é destacar os efeitos de envelhecimento acumulados durante a primeira fase. Além disso, espera-se que os efeitos de envelhecimento sejam estabilizados, já que a duração do experimento não é suficiente para revelar efeitos relativos ao envelhecimento de modo natural (não acelerado).

c) **Fase do Rejuvenescimento** - A última fase do experimento, que consiste na execução do processo de rejuvenescimento. Após o sistema ter acumulado os efeitos do envelhecimento (duas fases anteriores), o rejuvenescimento é iniciado com a migração da *VM*. O monitor continua a recolher informações do ambiente após a migração com o intuito de verificar se os efeitos de envelhecimento são removidos após a operação de migração.

Observou-se que o tempo necessário para a execução do experimento foi de 13 dias, pois, os arquivos de resposta do monitoramento do sistema são gerados de modo iterativo, o que permite a análise destes em paralelo com a execução do experimento. O experimento rodou durante 13 dias consecutivos. A organização do tempo para cada fase pode ser observada na Figura 3.6.

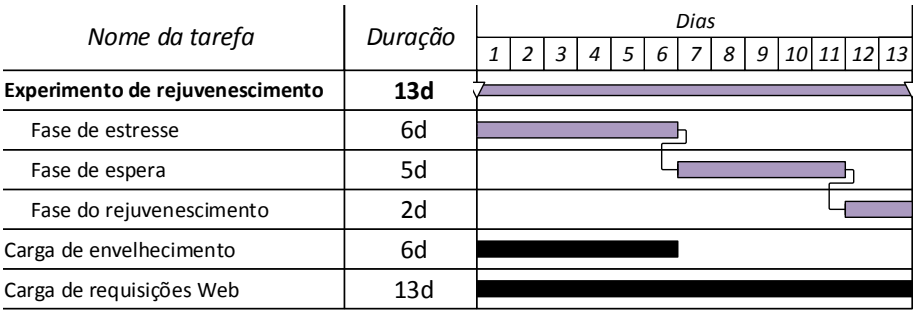


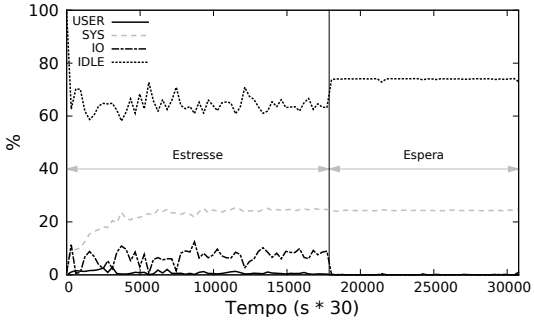
Figura 3.6: Distribuição de tempo para cada fase do experimento de rejuvenescimento

3.3 Resultados e análises

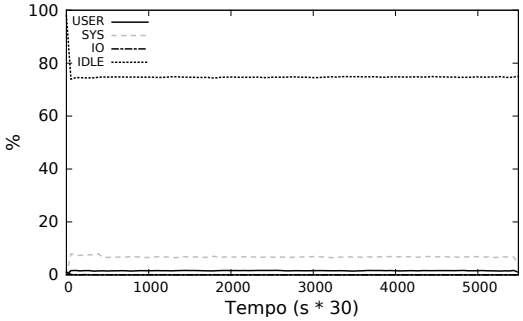
Os resultados do experimento podem ser vistos na Figura 3.7. Os resultados do monitoramento do nó estão divididos em duas colunas. A primeira diz respeito ao nó físico que comportou a VM antes do rejuvenescimento (*Nó AR*) e a segunda ao comportamento do nó físico após a migração da VM (*Nó DR*). Por fim são apresentados os resultados de erros e tempo de resposta do servidor Web. Como a carga de solicitações foi enviada por um agente externo à nuvem (*Stresser*), não houve interrupção no envio de solicitações, mesmo durante o processo de migração. Assim sendo, os resultados no *benchmark* ***Autobench*** são apresentados num gráfico só. Os gráficos indicam os limites de cada uma das fases do experimento (*Estresse*, *Espera* e *Rejuvenescimento*).

O monitoramento do processador (*CPU*) - presente nas Figuras reffig:cpunode-bm, fig:cpunode-am - é feito pela ferramenta ***mpstat***[6] que fornece a medição de diferentes atividades no processador. Sendo assim, no resultado do monitoramento do processador os itens observados denotam as seguintes atividades:

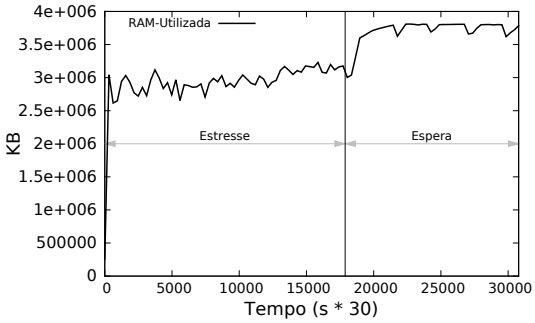
[6]http://www.linuxcommand.org/man_ pages/mpstat1.html



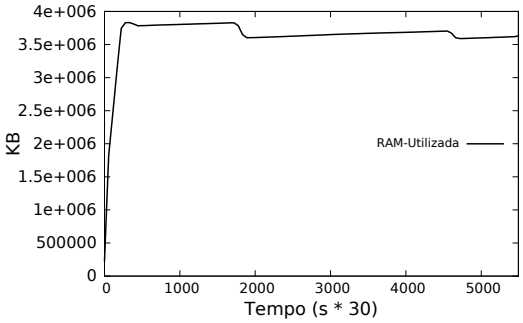
(a) Consumo CPU - *Nó AR*



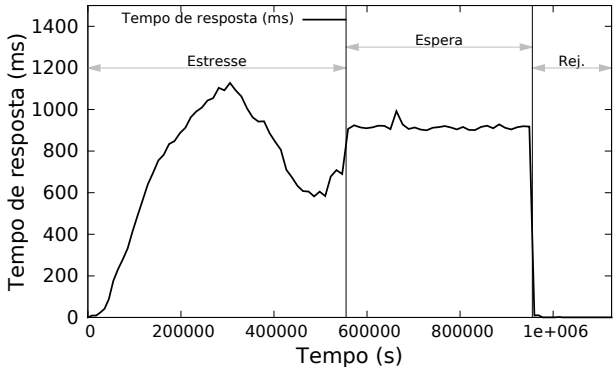
(b) Consumo CPU - *Nó DR*



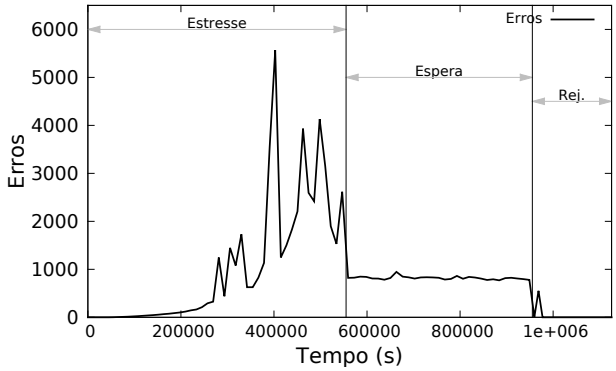
(c) Consumo RAM - *Nó AR*



(d) Consumo RAM - *Nó DR*



(e) Tempo de resposta



(f) Erros

Figura 3.7: Resultados dos experimentos de rejuvenescimento

USER - Mostra o percentual da utilização do processador que está relacionado com a execução no nível do usuário (aplicativos).

SYS - Mostra o percentual da utilização do processador que está relacionado com execuções no nível do sistema (kernel).

IO - Percentual do tempo que o processador fica ocioso por causa de uma requisição de entrada/saída no disco.

IDLE - Percentual de tempo que o processador fica ocioso, excluindo-se a espera por operação de entrada/saída no disco.

3.3.1 Análise dos resultados

Os gráficos mostram as consequências do envelhecimento e rejuvenescimento executados durante os testes. Os resultados do monitoramento do nó físico mostram que o envelhecimento de software afeta diretamente a maneira como os recursos são alocados. Na Figura 3.7, é possível observar que durante a fase de estresse, o consumo de recursos oscila de modo substancial. Percebe-se que a carga de trabalho submetida ao sistema o conduz a um estado instável. Nota-se com os resultados do monitoramento do processador, que a espera por entrada e saída (IO) oscila durante a fase de estresse do sistema. Isso se deve às solicitações de anexação e desmonte de disco, que necessitam esperar por informações oriundas da rede. Contudo, é importante salientar que, observando apenas os resultados do monitoramento do nó físico (CPU e RAM), durante a fase de estresse, não é possível afirmar a existência de indícios de envelhecimento de software.

Observando-se os resultados do servidor Web (tempo de resposta e erros) nota-se um acúmulo gradual de degradação no estado do servidor Web. Tal comportamento revela indícios de envelhecimento de software na plataforma. Em outras palavras, a VM necessita de comunicação intensa com o *hypervisor* durante a fase de estresse, pois a carga de trabalho realizada por este último realiza anexação e desmonte de discos na própria VM.

A fim de corroborar os indícios levantados na fase de estresse, a fase de espera revela que a degradação sofrida é mantida mesmo após a interrupção da carga de trabalho. Tal fenômeno confirma a existência de efeitos de envelhecimento de software no ambiente estudado, pois a degradação dos recursos foi mantida mesmo após a fase de estresse do ambiente. Entretanto essa degradação permanece de modo constante. Também é possível perceber que a alocação do processador para resolver solicitações do sistema (item *SYS* da Figura 3.7(a)) é crescente durante a fase de estresse e mantém-se em níveis mais altos mesmo depois do estresse. Com o comportamento dessa fase é possível concluir que os efeitos de degradação observados na fase de estresse são oriundos de envelhecimento de software e não do *overhead* de operações de anexação e desmonte de discos.

A última fase do experimento, é a fase do rejuvenescimento. Essa fase é iniciada após a realização da operação da migração da máquina virtual com propósito de rejuvenescimento de

software. Conforme explicado anteriormente a Figura 3.7 contém os resultados do monitoramento dos recursos antes e depois da migração da VM. Nas Figuras 3.7(d) e 3.7(b) é possível notar que a oscilação na alocação de recursos é amenizada após o processo de migração. O efeito do rejuvenescimento de software é destacado nos gráficos de desempenho do servidor Web (Figura 3.7(e) e Figura 3.7(f)). Observando a Figura 3.7(e) nota-se que o tempo de resposta, que está degradado pelos efeitos de envelhecimento, é reduzido substancialmente após a migração. O mesmo ocorre com os erros acumulados, que são reduzidos após a migração. O pico de erros, logo após a migração, justifica-se porque os testes não foram interrompidos durante o processo de migração que paralisa o sistema por um determinado período, ocasionando assim os erros encontrados.

Conclui-se então que os métodos adotados para o rejuvenescimento foram efetivos na plataforma estudada. Além disto, é possível confirmar que o processo de envelhecimento realmente ocorre no ambiente estudado, dado que o sistema acumula os efeitos de envelhecimento acarretados pelo teste de envelhecimento.

3.4 Considerações finais

Este capítulo apresentou um estudo experimental de envelhecimento e rejuvenescimento de software realizado em uma nuvem privada. O estudo foi conduzido numa plataforma OpenNebula 3.6, construída sobre um ambiente de virtualização com três componentes: *FrontEnd, MainNode e StandbyNode*, onde os nós executores possuem o *KVM 1.0* como *hypervisor*. A carga de trabalho para os testes de envelhecimento foi determinada com base em estudos anteriores (MATOS et al., 2012) e adaptada para o ambiente do OpenNebula. Os testes foram executados por 13 (treze) dias consecutivos, pois observou-se que esse tempo era o suficiente para a coleta dos resultados. O experimento foi dividido em três fases (estresse, espera e rejuvenescimento) visando destacar os efeitos relativos ao envelhecimento e rejuvenescimento de software na plaforma estudada.

A partir dos resultados obtidos é possível obter algumas conclusões relevantes. Inicialmente, os resultados da fase de estresse revelam que o sistema sofre uma degradação cumulativa quando é submetido ao processo de anexação e desmonte de discos virtuais. É possível observar nos resultados do monitoramento do servidor Web (Figuras 3.7(e) e 3.7(f)) que os efeitos de degradação afetam diretamente o tempo de resposta e a quantidade de erros observada no experimento. Entretanto, apenas a degradação da qualidade do serviço não é possível para afirmar a presença de indícios de envelhecimento de software. Por tal motivo, é necessário que exista a fase de espera do experimento, para observar se a degradação é oriunda de *overhead* ou do envelhecimento propriamente dito.

Os resultados da fase de espera revelam que os efeitos de degradação ficam acumulados, mesmo após a interrupção do envio da carga de trabalho ao sistema em si. Nos resultados do monitoramento do servidor Web, percebe-se que tanto os erros quanto o tempo de resposta

permanecem estáveis durante essa fase.

Com os resultados de rejuvenescimento de software é possível perceber que os efeitos de envelhecimento de software podem ser removidos a partir da migração de máquinas virtuais. É importante salientar que o rejuvenescimento baseado em migração de VMs mostrou-se efetivo nesse caso porque o envelhecimento de software afeta o *hypervisor* do ambiente. A migração permite que a VM usufrua das funcionalidades do *hypervisor* que não sofre efeitos de envelhecimento de software, localizado na máquina secundária do ambiente.

4
Modelos

Neste capítulo será apresentado um conjunto de modelos analíticos utilizados para alcançar os objetivos propostos por esta dissertação. Inicialmente são apresentados modelos que representam uma infraestrutura básica de computação em nuvem com apenas três componentes: *FrontEnd*, um nó e uma *VM*. Este modelo é utilizado como base para modelos mais complexos, construídos posteriormente. A abordagem adotada para a construção do modelo é a hierárquica, com submodelos baseados em *RBD*s e *CTMC*. Na Seção 4.2 é apresentado um conjunto de modelos que consideram o envelhecimento de software e políticas de rejuvenescimento baseadas em agendamento de migrações de VMs. Nesse caso, os modelos propostos também são hierárquicos, compostos por *RBD* e *SPN*, e permitem a avaliação de políticas de rejuvenescimento de software.

4.1 Modelos da infraestrutura básica

Nesta seção serão apresentados os modelos para avaliação de disponibilidade em uma infraestrutura básica de uma nuvem privada (*FrontEnd*, um nó e uma VM).

Arquitetura do sistema

Para realizar o estudo, utilizou-se uma infraestrutura básica de ambientes de nuvem como ambiente de testes similar a que foi apresentada na Seção 3.1. Neste estudo específico utilizou-se o *VIM* OpenNebula IaaS Cloud 3.6.0[1], mas é importante salientar que os modelos concebidos são genéricos e podem ser estendidos para outras plataformas. Porém, para os modelos da infraestrutura básica ignorou-se o comportamento do nó *Standby Node*, pois ele é utilizado para receber as migrações relativas ao rejuvenescimento de software. Para o estudo da infraestrutura básica decide-se ignorar os componentes adicionais, para se ter a arquitetura mais simples e genérica. Esta arquitetura considerada pode ser vista na Figura 4.1.

Nesta arquitetura, o ***Front End*** é a máquina responsável por gerir todo o ambiente da nuvem em questão. Já o ***Main Node*** é o nó de processamento da nuvem, ou seja, a máquina que

[1] `http://opennebula.org`

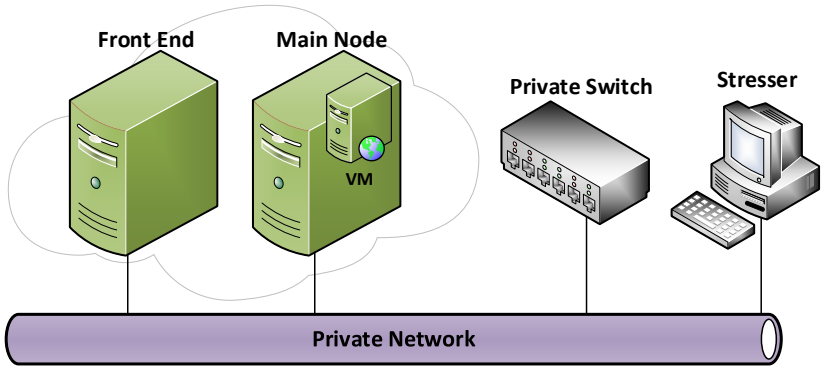


Figura 4.1: Arquitetura da infraestrutura básica de computação em nuvem

fornece seus recursos computacionais para a nuvem. No ambiente, o *Main Node* é utilizado para alocar a *VM*. O *Stresser* é utilizado para injetar falhas e monitorar o ambiente de nuvem.

Modo operacional

Neste ambiente, entende-se que o sistema se tornará indisponível quando ocorrer alguma falha na VM ou no nó ou no *FrontEnd*. O defeito na *VM* leva a uma paralisação no serviço disponibilizado na nuvem, o que gera paralisação nos serviços. O mesmo vale um defeito no nó (máquina física) que tem a VM alocada, caso ele falhe, a VM ficará indisponível. Como o *FrontEnd* é responsável por realizar operações de gerenciamento na nuvem, quando se torna defeituoso, perde-se o controle de acesso às VMs e fica impossível realizar operações de manutenção e gerenciamento em todo o ambiente de nuvem.

Assim sendo, o modo operacional para a plataforma pode ser sintetizado na Expressão lógica 4.1 (onde o FrontEnd, o nó e a VM são representados por FE, ND e VM, respectivamente).

$$Ambiente_{up} = FE_{up} \wedge ND_{up} \wedge VM_{up} \tag{4.1}$$

Modelos de disponibilidade

Para melhor organização, utilizou-se uma abordagem hierárquica de modelagem. Tal abordagem permite a utilização de submodelos para componentes específicos dos modelos (macromodelos), favorecendo assim a modularização e redução de complexidade. Devido à independência entre os componentes, para a avaliação de disponibilidade na infraestrutura básica da nuvem, utilizou-se o macromodelo baseado em blocos *RBD*. Como o comportamento de falhas e reparos do nó físico afeta o comportamento da *VM* do ambiente, decidiu-se que estes componentes deveriam ser modelados em um submodelo capaz de representar tal dependência. No caso, optou-se por utilizar as *CTMC*s. Para o macromodelo inicial, utilizou-se a nomenclatura *CloudNode* para denotar o comportamento de falhas e reparos da *VM* e do nó. A Figura 4.2 abaixo mostra o macromodelo concebido para avaliar a infraestrutura básica de ambientes de

nuvens privadas.

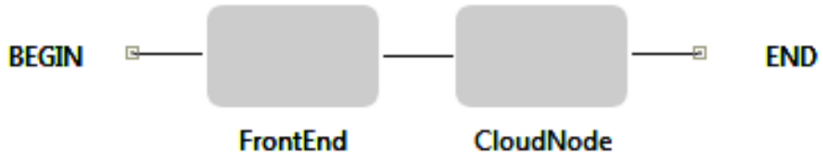


Figura 4.2: Macromodelo de disponibilidade do ambiente

Conforme visto anteriormente (Expressão 4.1), para o sistema estar ativo faz-se necessário que todos os componentes da arquitetura estejam funcionando. Logo, o modelo concebido para representar o modo operacional do sistema é composto por dois blocos RBD em série, *FrontEnd* e *CloudNode* (que representa o nó e a VM juntos). Com esta representação, a disponibilidade pode ser obtida através da Equação 4.2, onde A representa Disponibilidade, e o sistema, o *FrontEnd* e o *CloudNode* são representados por s, fe, cn, respectivamente.

$$A_s = A_{fe} \times A_{cn} \tag{4.2}$$

Este modelo é considerado o macromodelo na abordagem hierárquica utilizada. Sendo assim, cada um dos blocos *RBD* possui um submodelo atrelado.

Bloco FrontEnd

O bloco FrontEnd representa o subsistema de gerenciamento da nuvem. Seus componentes principais são:

- Hardware (HW);
- Sistema Operacional (SO) - O sistema operacional da máquina física que suporta o subsistema;
- *Manager* (Mg) - Software responsável por monitorar e gerenciar a nuvem.

Todos estes componentes são importantes para o funcionamento do subsistema *FrontEnd*. Sendo assim, se qualquer um deles falhar, o *FrontEnd* apresentará um defeito. Logo, para representar o comportamento do *FrontEnd*, utilizou-se um RBD com três blocos em série, cada bloco representando um componente do *FrontEnd*. O modelo pode ser visto na Figura 4.3.

A disponibilidade do FrontEnd é representada através da Equação 4.3. A descrição de cada parâmetro utilizado pode ser encontrada na Tabela 4.1.

$$A_{fe} = \frac{\mu_{hw}\mu_{mg}\mu_{so}}{\mu_{so}(\mu_{mg}(\lambda_{hw}+\mu_{hw})+\lambda_{mg}\mu_{hw})+\lambda_{so}\mu_{hw}\mu_{mg}} \tag{4.3}$$

Figura 4.3: RBD FrontEnd

Tabela 4.1: Parâmetros genéricos para o RBD FrontEnd

Parâmetro	Descrição
λ_{hw}	Taxa de falha do Hardware
μ_{hw}	Taxa de reparo do Hardware
λ_{so}	Taxa de falha do Sistema Operacional
μ_{so}	Taxa de reparo do Sistema Operacional
λ_{mg}	Taxa de falha do manager
μ_{mg}	Taxa de reparo do manager

Bloco CloudNode

O bloco *CloudNode* representa o comportamento do *Main Node* e da *VM* juntos. Conforme visto na arquitetura, o *Main Node* executa uma *VM* que aloca um servidor Web. Isto representa uma das infraestruturas mais básicas de uma nuvem privada. Devido à dependência entre os componentes, utilizou-se as CTMCs para modelar o comportamento do nó de processamento. O modelo pode ser visto na Figura 4.4[2].

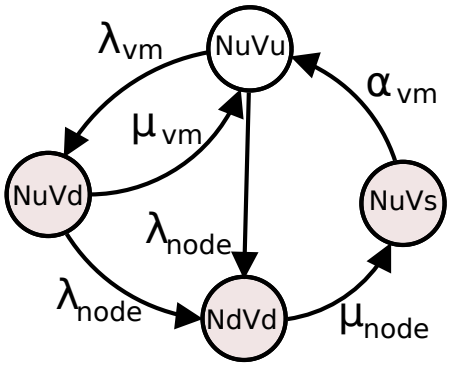


Figura 4.4: CTMC CloudNode

O estado `NuVu` é o estado inicial do sistema, representa que tanto a *VM* quanto o nó estão funcionando. Neste estado dois prováveis eventos podem ocorrer: falha na *VM* (com taxa exponencialmente distribuída λ_{vm}) ou falha no nó (com taxa λ_{node}). Caso ocorra uma falha na *VM* (`NuVd`), é possível reparar a VM (evento que ocorre com a taxa exponencialmente distribuída μ_{vm}) fazendo o sistema voltar ao estado de funcionamento normal (`NuVu`), ou ainda, é possível ocorrer uma falha no nó. Independente do estado da *VM*, a falha do Nó leva a *VM* ao estado de inatividade (`NdVd`), pois a *VM* depende do nó para funcionar. O reparo, após uma falha no nó, é

[2] **N**=Nó, **V**=VM, **u**=Up (funcionando), **d**=Down (em falha), **s**=Stopped (paralisada)

feito em dois passos. Primeiro é realizado o reparo no componente do Nó que o levou à falha (`NuVs`), e então a VM é reiniciada (α_{vm}).

A partir do modelo da Figura 4.4 é possível obter uma expressão para o cálculo da disponibilidade do *CloudNode* (Equação 4.4).

$$A_{cn} = \frac{\alpha_{vm}\mu_{node}(\lambda_{node}+\mu_{vm})}{(\alpha_{vm}(\lambda_{node}+\mu_{node})+\lambda_{node}\mu_{node})(\lambda_{node}+\lambda_{vm}+\mu_{vm})} \tag{4.4}$$

Retomando o modelo macro, com a obtenção das equações 4.3 e 4.4 é possível concluir a fórmula da disponibilidade do ambiente.

$$A_s = \frac{\alpha_{vm}\mu_{hw}\mu_{mg}\mu_{node}\mu_{so}(\lambda_{node}+\mu_{vm})}{(\alpha_{vm}(\lambda_{node}+\mu_{node})+\lambda_{node}\mu_{node})(\lambda_{node}+\lambda_{vm}+\mu_{vm})(\mu_{so}(\mu_{mg}(\lambda_{hw}+\mu_{hw})+\lambda_{mg}\mu_{hw})+\lambda_{so}\mu_{hw}\mu_{mg})} \tag{4.5}$$

É possível simplificar esta expressão transformando as taxas de falha e reparo dos componentes do *FrontEnd* em um componente só. Ou seja,

$$A_{fe} = \frac{\mu_{fe}}{\lambda_{fe}+\mu_{fe}} \tag{4.6}$$

aplicando isso na equação anterior tem-se:

$$A_s = \frac{\alpha_{vm}\mu_{fe}\mu_{node}(\lambda_{node}+\mu_{vm})}{(\lambda_{fe}+\mu_{fe})(\alpha_{vm}(\lambda_{node}+\mu_{node})+\lambda_{node}\mu_{node})(\lambda_{node}+\lambda_{vm}+\mu_{vm})} \tag{4.7}$$

utilizando-se da equação 4.7 é possível obter a disponibilidade do ambiente modelado.

Validação do modelo

O crédito atribuído às avaliações de modelo de disponibilidade está atrelado à sua capacidade de representar as situações reais do ambiente (GOEL, 1985). O processo de validação de modelos de confiabilidade e disponibilidade é, em geral, uma tarefa complexa devido à natureza dos fenômenos estudados. Em modelos de desempenho, por exemplo, o processo de validação pode ser realizado confrontando os resultados obtidos através dos modelos com os resultados de *benchmarks* e medições realizadas diretamente no ambiente (LAVENBERG, 1983). Entretanto, os modelos de confiabilidade e disponibilidade lidam com fenômenos inesperados, como falhas, travamentos e interrupções de serviços. E, assim como em outros modelos analíticos, é necessário conjuntos de dados para obtenção de estatísticas necessárias para a validação (BUTLER; FINELLI, 1993). Os valores utilizados para parametrizar os modelos são obtidos por estimativas e fornecidos por fabricantes (no caso do hardware). Além disso, algumas falhas podem demorar até anos para ocorrer, o que dificulta ainda mais o processo de validação (SCHROEDER; GIBSON, 2007).

Existem alguns métodos que visam avaliar a confiabilidade de um sistema (HAMLET,

1994). Geralmente, estes métodos são baseados na aceleração de tempo de falha dos ambientes estudados a fim de testar a sua robustez e verificar o comportamento do sistema perante tais eventos (CHAN, 2004). Um dos métodos que pode ser utilizado para a avaliação de confiabilidade de um ambiente computacional é a injeção de falhas (ARLAT et al., 1993).

Baseando-se no processo de injeção de falhas e reparos, é proposto um *workflow* para a validação dos modelos. O processo de injeção de falhas e reparos foi baseado em (SOUZA et al., 2013). No artigo, uma ferramenta é proposta para avaliação de disponibilidade em ambientes de nuvem. O processo adotado é sintetizado na Figura 4.5.

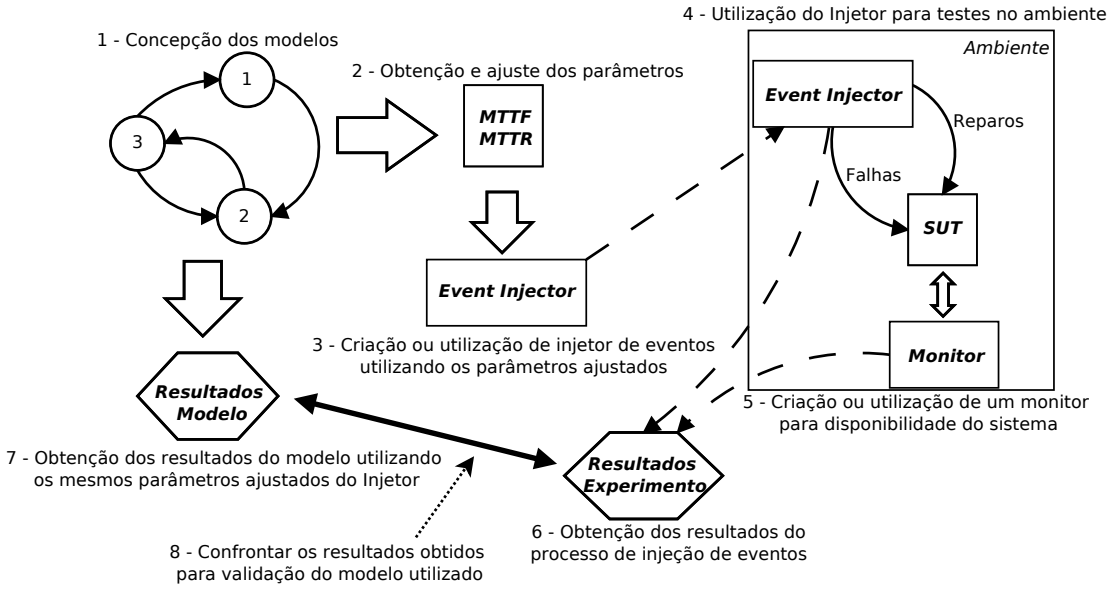


Figura 4.5: *Workflow* para validação dos modelos

A seguir serão apresentadas as atividades adotadas para cada atividade do processo de validação:

a) **Concepção dos modelos** - Com base em estudos e observação do sistema de estudo, deve-se construir os modelos que representam seu comportamento.

b) **Obtenção e ajuste dos parâmetros** - A utilização de injeção de falhas para validação do modelo exige que as características dos tempos de falha e reparo sejam definidas previamente. Dado que os tempos de falha são geralmente muito longos, é necessário reduzi-los para viabilizar a validação. Propõe-se a redução utilizando um fator constante (SOUZA et al., 2013). Para o estudo em questão, utilizou-se o fator de redução 400 nos parâmetros de teste do *FrontEnd* e o fator de redução 1000 para os testes relativos ao *CloudNode*. De modo prático, os tempos de falha devem ser reduzidos para tornar o experimento viável. Vale a pena salientar que tais reduções foram aplicadas apenas nos tempos de falha dos componentes, os tempos de reparo permaneceram inalterados. Os parâmetros originais que servem como base para as

reduções são apresentados na Tabela 4.2. Todos os tempos são exponencialmente distribuídos.

c) **Construção ou utilização de *Event Injector*** - Para o caso, utilizou-se a linguagem *shell script* para a implementação do injetor de eventos (*Event Injector*) . A descrição dos códigos-fonte utilizados para injeção dos eventos pode ser encontrada no Algoritmo 2. Além da falha, injeta-se também o reparo do componente, o intuito é representar uma equipe de manutenção. Neste estudo em particular, os tempos de reparo utilizados não foram alterados, ou seja preservou-se as características específicas do reparo de cada componente. Dessa forma procurou-se observar como o sistema responde às falhas injetadas e às estratégias de reparos utilizadas.

d) **Utilizar o *Event Injector* no ambiente** - Com o *Event Injector* parametrizado, as falhas e reparo começam a ser injetadas no ***SUT***[3]. O montante de falhas e reparos gerados pelo *Event Injector* está presente na Tabela 4.3.

e) **Monitorar o ambiente** - O monitoramento da disponibilidade consiste na verificação periódica do status do sistema, checando se seus componentes estão ativas. A ferramenta utilizada gera um arquivo contendo os dados obtidos do monitoramento. O resultado do monitoramento é usado para comparação com os resultados obtidos através do modelo.

f) **Obtendo resultados dos experimentos** - Após o término dos experimentos, são gerados alguns relatórios de dados obtidos do ***Event Injector*** e do ***Monitor***. Os resultados da injeção de eventos são usados para o cálculo do intervalo de confiança da disponibilidade. Para o estudo atual utilizou-se o método apresentado em (KEESEE, 1965) (ver Apêndice A). Os resultados do monitor de disponibilidade são obtidos por intermédio da relação entre o tempo de atividade do sistema e o tempo total do experimento (Equação 2.1).

g) **Avaliar o modelo** - Utilizando os mesmos parâmetros do *Event Injector*, é realizada a avaliação do modelo para obter-se a disponibilidade estacionária.

h) **Comparação dos resultados** - O último passo para a validação do modelo é comparar os resultados obtidos através do modelo com os resultados do monitoramento da disponibilidade. Utilizando o método de Keese (etapa 6), obtém-se um intervalo de confiança para a disponibilidade do sistema. O modelo é dito validado se os resultados obtidos (modelo e monitor) estiverem dentro do intervalo de confiança da disponibilidade.

[3] ***S**ystem **U**nder **T**est* - Sistema sob testes

Tabela 4.2: Parâmetros originais considerados

Parâmetro	Descrição	Taxa (1/h)	Tempo
α_vm	Taxa de *restart* da VM	12	5 min
λ_vm	Taxa de falha da VM	0,000347222	2880 h
μ_vm	Taxa de reparo da VM	2	30 min
λ_node	Taxa de falha do Nó	0,000808604	1236,7 h
μ_node	Taxa de reparo do Nó	0,917431193	1,09 h
λ_FrontEnd	Taxa de falha do FrontEnd	0,002076843	481,5 h
μ_FrontEnd	Taxa de reparo do FrontEnd	0,970873786	1,03 h

Algoritmo 2 *Event Injector*

```
loop
    if ComponenteEstaAtivo then
        TempoFalha = TempoExponencial(TaxaDeFalhaComponente);
        Esperar(TempoFalha);
        InjetarFalhaComponente();
        ColetarDados(OrdFalha, TempoFalha, timestamp)
    else
        TempoReparo = TempoExponencial(TaxaDeReparoComponente);
        Esperar(TempoReparo);
        InjetarReparoComponente();
        ColetarDados(OrdReparo, TempoReparo, timestamp)
    end if
end loop
```

Tabela 4.3: Parâmetros de Entrada para o Método de Keese

Parâmetros	Valores
Tempo Total de Falha da VM	325200,64890s
Tempo Total de Reparo da VM	42277,19342s
Quantidade de Falhas e Reparos da VM	24
Tempo Total de Falha no Nó	241110,99177s
Tempo Total de Reparo da Nó	225141,69740s
Quantidade de Falhas e Reparos da Nó	50
Tempo Total de Falha do FrontEnd	188168,49278 s
Tempo Total de Reparo da FrontEnd	146739,78780 s
Quantidade de Falhas e Reparos da Frontend	36

O *Event Injector* utilizado para os testes foi desenvolvido com a linguagem *Shell Script*, seu pseudocódigo é apresentado no Algoritmo 2. Este injetor trabalha em ciclos repetitivos, injetando falhas e reparos nos componentes desejados. Em cada ciclo é feita uma verificação no *status* do componente. Caso o componente esteja ativo, ou seja, funcionando, é gerado um tempo exponencial para falha, obedecendo à taxa de falha deste componente específico. Depois que o tempo é gerado, o injetor espera o tempo de falha terminar, para então, injetar a falha. Assim que a falha é injetada, o *Event Injector* coleta os dados de ordem, o tempo de falha gerado e o *timestamp*. Caso o componente esteja inativo, um processo similar ocorre, porém injetando-se o comando de reparo e considerando-se o tempo de reparo para o componente específico.

A Tabela 4.4 mostra os resultados da validação dos os modelos propostos neste capítulo.

Tabela 4.4: Validação das disponibilidades

Componente	Modelo	Monitor	95 % IC	
			Mínimo	**Máximo**
Bloco *FrontEnd*	0,5372	0,5614	0,3954	0,7155
Bloco *CloudNode*	0,4559	0,4897	0,3256	0,7161
Disponibilidade Geral	0,2449	0,2749	0,1287	0,5124

Os dados exibidos na Tabela 4.4 mostram que os resultados obtidos através da avaliação dos modelos e do monitoramento do sistema obedecem ao intervalo de confiança gerado pelo método de Keesee (apresentado em KEESEE (1965)). O método de Keese revela a disponibilidade esperada para o sistema, perante os parâmetros determinados pelo *Event Injector*. Com isto, entende-se que a disponibilidade do sistema deve obedecer a este intervalo de confiança. Para verificar isto, observam-se os resultados do monitoramento do sistema. Se estes obedecerem ao intervalo de confiança, diz-se que o sistema comportou-se como esperado. Por fim, compara-se o resultado obtido através dos modelos. Caso estes sejam também compatíveis com o intervalo de confiança, é possível afirmar que o modelo realmente corresponde ao comportamento real do sistema.

Considerações

Esta seção apresentou um conjunto de modelos para avaliar a disponibilidade de infraestruturas básicas de nuvem privada. Foram geradas fórmulas fechadas para o cálculo da disponibilidade de cada componente, bem como para o cálculo da disponibilidade geral do ambiente. Ainda nesta seção foi exibida uma metodologia para validação de modelos de disponibilidade. Este processo consiste na utilização de um injetor de eventos no ambiente real. O ambiente é monitorado e o resultados de injeção de eventos são gerados pelo injetor de eventos. Tais resultados da injeção de eventos servem de insumo para a obtenção do intervalo de confiança da disponibilidade, a partir do método de Keesee (KEESEE, 1965). Os resultados obtidos do modelo e do monitoramento permanecem dentro do intervalo de confiança da disponibilidade. A partir disto pode-se afirmar que os modelos correspondem ao comportamento do sistema real. Deste modo, é possível atribuir mais confiança aos modelos propostos.

4.2 Modelos para avaliação de disponibilidade com rejuvenescimento habilitado por migração de VM

Tendo como base os modelos da seção anterior, esta seção apresenta um modelo hierárquico para representar comportamentos de envelhecimento e rejuvenescimento de software. Estudos prévios indicam que ambientes de computação em nuvem sofrem com envelhecimento de software (ARAUJO, 2012). Além disso, os experimentos realizados no Capítulo 3 também revelam indícios de envelhecimento de software na plataforma de nuvem privada *OpenNebula* utilizada nos experimentos. Assim, para conduzir uma avaliação de disponibilidade mais apropriada, decidiu-se incorporar estes comportamentos nos modelos já produzidos.

A arquitetura considerada é a mesma que foi apresentada na Seção 3.1. Os modos de falha e reparo também já foram apresentados na Seção 4.1.

Conforme visto anteriormente, efeitos de envelhecimento de software em plataformas de computação em nuvem podem acarretar falhas e degradação do desempenho do sistema. Por causa disto, estudos de disponibilidade em computação em nuvem devem levar em consideração tais aspectos.

Visando mitigar os efeitos relativos ao envelhecimento de software em ambientes de computação em nuvem, a Seção 3.2 propõe uma metodologia de rejuvenescimento de software baseada em agendamento de migração de máquinas virtuais.

A migração da VM via *live migration* acarreta paralisações nos serviços hospedados por esta. A execução recorrente de migrações pode sempre manter o sistema rejuvenescido, mas, em contrapartida pode afetar diretamente sua disponibilidade devido às interrupções causadas por estas operações. Realizar migrações entre longos intervalos de tempo pode permitir uma falha por envelhecimento, o que também afeta a disponibilidade do sistema. Os modelos propostos a seguir visam determinar agendamentos adequados para as migrações, com o intuito de maximizar

a disponibilidade do sistema.

O modelo não leva em consideração os tempos de detecção de falhas. No processo de migração (*Live Migration*), considera-se apenas o tempo de interrupção relativo à esta operação, negligenciando-se os demais detalhes de implementação. Os problemas da rede de comunicação também não foram considerados.

Inicialmente considera-se o mesmo macromodelo hierárquico, similar ao apresentado anteriormente, como na Figura 4.6. Nesse modelo existem dois blocos RBD em série, construídos de modo hierárquico, em que o primeiro representa o comportamento do FrontEnd (bloco *FrontEnd*) na nuvem e o segundo o comportamento dos nós executores da nuvem e da VM (bloco *CloudNodes*). Para o modelo proposto, o bloco relativo ao comportamento do *FrontEnd* permanece com as mesmas características que foram apresentadas anteriormente. Porém, o bloco dos nós executores (*CloudNodes*) é adaptado para representar os efeitos de envelhecimento de software e as políticas de rejuvenescimento.

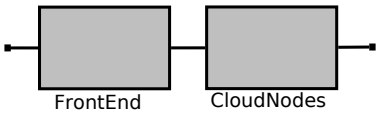


Figura 4.6: Modelo hierárquico do sistema

Conforme apresentado no Capítulo 2, a modelagem de sistemas com dependência entre os componentes deve ser feita com modelos espaço-estado, como *CTMC*s ou *SPN*s. Porém, a construção de *CTMC*s com muitos estados pode tornar-se uma atividade muito repetitiva e passível de erros. Logo, uma alternativa de modelagem para sistemas com comportamentos mais complexos são as Redes de Petri. Com a utilização de Redes de Petri, a modelagem de comportamentos complexos pode ser realizada de modo mais intuitivo. Além disso, a modelagem de outras distribuições de probabilidade (como Hiper-exponenciais e *Erlang*) pode ser realizada a partir do acoplamento de *throughput nets* específicas, o que facilita a modelagem de comportamentos que obedecem tais distribuições. Portanto, para a adaptação do modelo *CloudNodes* decidiu-se utilizar as *eDSPN*s.

O modelo concebido é apresentado na Figura 4.7 [4]. Esse modelo foi concebido para representar o comportamento de rejuvenescimento de software habilitado por agendamento de migração de máquinas virtuais. O intuito principal é avaliar o impacto que diferentes agendamentos de migração produzem na disponibilidade estacionária do sistema. Utiliza-se sub-redes específicas para representar o agendamento das migrações e para representar o comportamento dos efeitos de envelhecimento e rejuvenescimento de software. O comportamento representado por esse modelo é similar ao comportarmento do mecanismo de redundância *warm-standby*. No mecanismo de redundância *warm-standby*, existe um sistema reserva, que fica em estado de espera até ser solicitado. Além disso, existe um tempo curto de ativação do sistema reserva, o que provoca um breve *downtime* no sistema.

[4] `MN` - *Main Node*, `SN` - *Standby Node*, `UP` - Funcionando corretamente, `DW` - Em falha

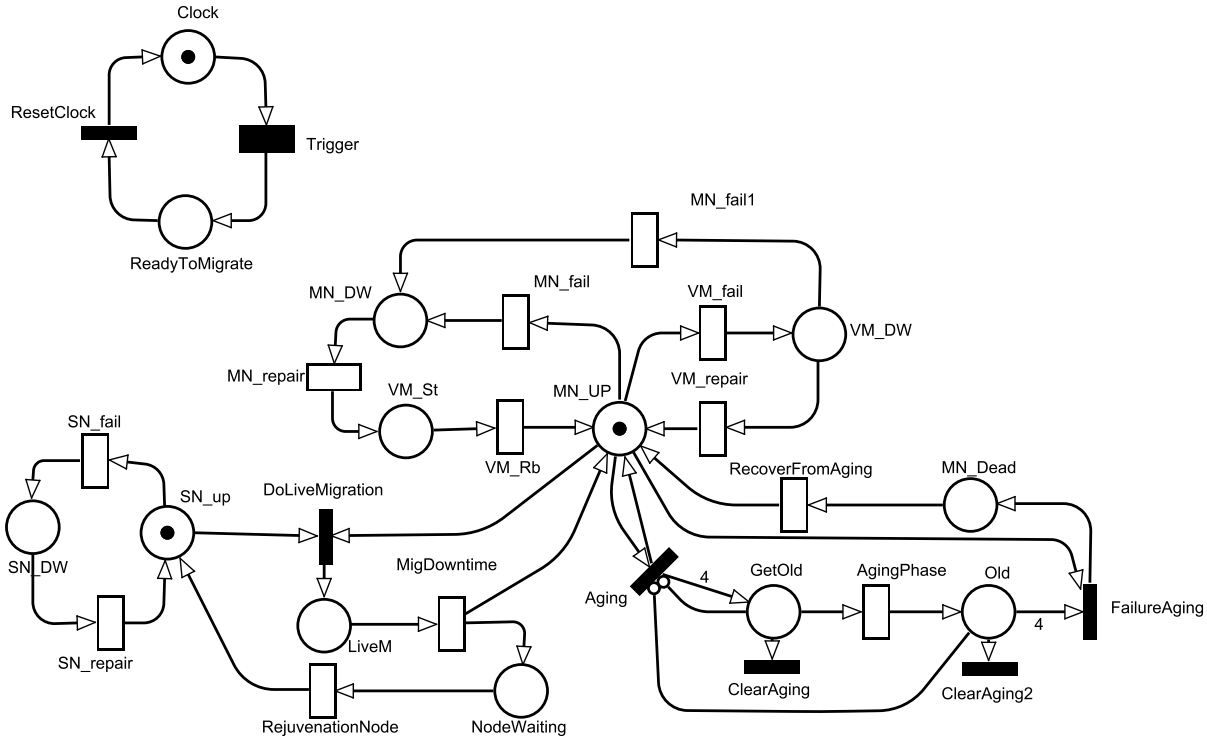


Figura 4.7: eDSPN CloudNodes

Tabela 4.5: Funções de guarda do modelo CloudNodes

Transição	Função de guarda
`ResetClock`	# `LiveM` = 1
`DoLiveMigration`	# `ReadyToMigrate` = 1
`ClearAging` e `ClearAging2`	#`VM_DW` = 1 OR # `MN_DW` = 1 OR # `LiveM` = 1

O mesmo comportamento de falhas e reparos dos componentes da infraestrutura básica, apresentado na seção anterior, é agregado no modelo *SPN CloudNodes*. Com isto, aproveita-se o processo de validação que foi realizado previamente também para o modelo *CloudNodes*.

O comportamento de falha e reparo do nó físico principal (*Main Node*) apresentado na Seção 4.1 é representado pela parte superior do modelo. O *token* no lugar `MN_UP` representa o estado do sistema onde, tanto a VM quanto a máquina física que a hospeda (*MainNode*) estão funcionando corretamente. Ou seja, quando há um *token* neste lugar, o sistema está disponível, independente dos demais lugares e *tokens* na rede. A obtenção da disponibilidade desta *SPN* é realizada observando a probabilidade deste lugar possuir um *token*.

No modelo, as transições com o sufixo `_fail` representam as falhas que não são relacionadas com o envelhecimento de software. Exemplificando, caso `MN_fail` seja disparada, isto representa a ocorrência de alguma falha não correlata ao envelhecimento no *Main Node* (ex. falha do Sistema Operacional, falha no Hardware).

No momento inicial, o modelo considera que os componentes *Main Node*, *VM* (lugar `MN_UP`) e o *Standby Node* (lugar *SN_UP*), estão funcionando corretamente. As transições `VM_fail` e `VM_repair` representam o evento de falha não correlata ao envelhecimento e

de reparo da *VM* do ambiente, respectivamente. Após a falha da VM, que leva o *token* ao lugar VM_DW, dois eventos são possíveis: a falha do nó (MN_fail1) ou o reparo da VM (VM_repair). A falha no nó também pode ocorrer antes da falha na VM (transição MN_fail), o que leva o *token* ao lugar MN_DW. O reparo do nó é feito em dois passos, primeiro é realizado o reparo no nó (transição MN_repair) e depois a *VM* é reiniciada (transição VM_Rb).

A arquitetura do ambiente considera a existência de um nó secundário (*Standby Node*) para receber as migrações para rejuvenescimento. Conforme dito anteriormente, inicialmente, esse nó está funcionando corretamente. Mesmo sem rodar *VM*s, esta máquina deve estar funcionando pois seu intuito deste componente é estar de prontidão para receber possíveis migrações de VM oriundas do *Main Node*. Para tanto, considera-se que, no estado inicial, todos os módulos e serviços necessários para receber a migração estão funcionando corretamente. Justamente por possuir os mesmos componentes de software do *Main Node*, excluindo-se apenas a VM, e estar executando tais serviços, o *Standby Node* também pode apresentar falhas não relativas ao envelhecimento de software. O comportamento de falha e reparo deste componente está representado no modelo com as transições SN_fail e SN_repair, respectivamente. Os lugares SN_up e SN_DW representam os possíveis estados para este nó, funcionando corretamente e paralisado por ter falhado, respectivamente.

Os efeitos do envelhecimento de software são acumulados gradativamente com o passar do tempo. Conforme o tempo passa, a probabilidade do sistema falhar por envelhecimento é maior. Ou seja, a taxa de falha é crescente. Conforme visto no Capítulo 2, eventos com taxa de falha crescente obedecem a distribuições de probabilidades *Erlang* ou hipoexponenciais. A distribuição *Erlang* é um caso particular da distribuição hipoexponencial, onde todas as fases possuem as mesmas taxas. Quanto maior o número de fases, mais próximo do determinismo será a ocorrência de eventos. Entretanto, conforme explicado anteriormente, o comportamento de envelhecimento de software é gradativo, mas é difícil estimar com exatidão os momentos de ocorrência de falhas, pois os efeitos de envelhecimento são relacionados com a carga de trabalho que o sistema enfrenta.

Então, para este caso específico, utilizou-se uma sub-rede *Erlang* com 4 fases. A rede é composta pelos lugares GetOld e Old e pelas transições Aging, AgingPhase e FailureAging. Os dois lugares desta sub-rede acumulam *tokens* representando o processo de envelhecimento. Se nenhuma operação de rejuvenescimento acontecer, ocorre uma falha por envelhecimento FailureAging. Após a ocorrência de uma falha relativa a envelhecimento, um *token* é depositado no lugar MN_Dead. A partir desta situação, será necessário reparar a falha acarretada por envelhecimento. Tal evento é representado pela transição RecoverFromAging, após seu disparo, um *token* será depositado no lugar MN_UP novamente. O que indica que os componentes *Main Node* e *VM* estão funcionando corretamente.

Os mecanismos de rejuvenescimento de software são utilizados para remover a degradação acumulada pelo envelhecimento. Nos modelos propostos, o rejuvenescimento adotado é baseado no agendamento de migrações de máquinas virtuais. O processo consiste em re-

alizar a migração da *VM* do *Main Node* para o *Standby Node* no momento preestabelecido por um agendamento. No modelo o processo de rejuvenescimento é iniciado pela transição `DoLiveMigration`. Quando o momento preestabelecido é alcançado, a migração da *VM* deve ser iniciada. Além de ter atingido o momento do agendamento, para a migração ocorrer é necessário que os nós (*Main Node* e *Standby Node*) e a `VM` estejam ativos. No modelo, isto é representado quando existem *tokens* nos lugares `SN_up` e `MN_UP`. Satisfeitas as condições, a transição `DoLiveMigration` dispara. Após seu disparo, um *token* é depositado em `LiveM`. Quando a rede alcança este estado, entende-se que o rejuvenescimento está prestes a acontecer (por causa da migração da VM). Assim sendo, todos os *tokens* depositados pelos lugares `GetOld` ou `Old` são removidos pelas transições imediatas `ClearAging` e `ClearAging2`.

Neste modelo, considera-se que, caso alguma falha ocorra no *Main Node* ou na *VM*, o reparo implicará na execução de rejuvenescimento de software. Ou seja, entende-se que as operações de reparo englobam atividades como o *reboot* do sistema operacional ou o *restart* dos serviços. Logo, quando um *token* é depositado em algum lugar que represente falha do *Main Node* ou *VM* as transições `ClearAging` e `ClearAging1` são habilitadas e disparadas, removendo os *tokens* dos lugares `GetOld` e `Old`, respectivamente . As demais funções de guarda podem ser encontradas na Tabela 4.5

Alguns detalhes do processo de *live migration* não foram considerados para este modelo de disponibilidade, que leva em consideração apenas o tempo de interrupção causado pela migração da VM. No modelo, este tempo de interrupção é representado pela transição `MigDowntime`. O disparo de `MigDowntime` deposita *tokens* nos lugares `MN_UP` e `NodeWaiting`. Conforme visto anteriormente, o lugar `MN_UP` representa que o *Main Node* e a *VM* estão funcionando corretamente. O lugar `NodeWaiting` representa um estado de espera para rejuvenescimento no nó físico *Main Node* que foi liberado de executar a *VM* após a migração. Entende-se que este nó está com envelhecimento acumulado, assim, deve ser rejuvenescido. A transição `RejuvenationNode` representa a ação de rejuvenescimento de software aplicada no nó (ex. *reboot* do SO, *restart* de serviços).

Após o rejuvenescimento, o nó que estava envelhecido volta ao status normal e assume o papel de *Standby Node*. Isto é representado no modelo quando a transição `RejuvenationNode` dispara, depositando um token no lugar `SN_UP`. A partir deste momento as máquinas físicas trocam de papel, a máquina que era *Main Node* passa a ser a *Standby Node* e vice-versa. No modelo, o rejuvenescimento efetuado com sucesso leva a rede de volta ao seu estado inicial.

A estratégia de rejuvenescimento de software adotada é baseada em agendamento de migrações de máquinas virtuais. Para representar o agendamento no modelo, utilizou-se um submodelo em *SPN* específico. Este submodelo encontra-se na parte superior esquerda do modelo apresentado na Figura 4.7, e é composto pelos lugares `Clock` e `ReadyToMigrate`, pela transição determinística[5] `Trigger` e pela transição imediata `ResetClock`.

A transição `Trigger` determina o intervalo de tempo entre as migrações. Pois, con-

[5]Com tempo exato

forme visto, migrações excessivas degradam a disponibilidade do sistema, e intervalos de tempo muito longos entre migrações podem colocar o sistema em risco de falha por envelhecimento. No momento inicial existe um token no lugar `Clock`, isto representa o início da contagem de tempo para disparar uma migração no ambiente. Quando o intervalo determinado em `Trigger` é alcançado, isto representa que o agendamento selecionado foi alcançado e o processo de rejuvenescimento deve ser submetido ao sistema. Com o disparo de `Trigger` um *token* é depositado no lugar `ReadyToMigrate`. Quando isso acontece o disparo de `DoLiveMigration` é habilitado, o que significa que o rejuvenescimento está habilitado para ocorrer. O relógio é reiniciado quando o disparo de `DoLiveMigration` deposita um token no lugar `LiveM`. Isto representa que o relógio do ambiente reinicia a sua contagem de tempo.

Modelo com mecanismo *cold-standby*

Para melhorar a disponibilidade de sistemas computacionais distribuídos, alguns administradores podem optar por mecanismos de redundância. Em geral, os mecanismos de redundância consistem numa replicação do sistema principal em uma ou mais máquinas secundárias. Essas máquinas secundárias ficam em estado de espera, caso aconteça algum problema com o sistema principal, elas poderão assumir o papel de máquina principal para reduzir o *downtime* sofrido pelo sistema (DANTAS et al., 2012b).

Dentre os mecanismos de redundância, existe o mecanismo de *cold-standby*. Nesse mecanismo, a máquina reserva fica inativa até que seja necessária sua utilização. Ou seja, quando o sistema principal falha, o sistema reserva terá que passar por um processo de inicialização e preparação antes de assumir o papel de principal (DANTAS et al., 2012b). Uma das vantagens na utilização do mecanismo *cold-standby* é que a máquina secundária, por estar desligada, não encontra falhas de funcionamento em seus componentes.

Com o intuito de avaliar os impactos desse mecanismo de redundância na disponibilidade do sistema outro modelo SPN, baseado no modelo *CloudNodes*, foi concebido para representar o mecanismo de redundância *cold-standby*. O modelo pode ser visto na Figura 4.8.

Na arquitetura considerada, o mecanismo *cold-standby* funcionará da seguinte maneira. Exclui-se o comportamento de falhas e reparos relativos ao *Standby Node* do modelo, pois considera-se que ele estará inativo durante a utilização do sistema principal. Ou seja, remove-se o lugar `SN_up`, e as transições `SN_fail` e `SN_repair` do modelo apresentado na Figura 4.7. Contudo, quando o intervalo de migração for atingido, deve-se iniciar o *Standby Node* para efetuar o rejuvenescimento desejado. Para representar esse comportamento no modelo foi incluído o lugar `SN_DW` após a transição `Trigger` e a transição `StartSN`. Quando o intervalo entre migrações é alcançado um *token* é depositado no lugar `SN_DW`, isso representa que a migração deve ser iniciada, porém o *Standby Node* ainda está inativo. O evento de preparar o *Standby Node* para receber a *VM* é representado pela transição `StartSN`, ou seja, esta transição representa o tempo de ativação do mecanismo *cold standby*. Após o preparo do *Standby Node*,

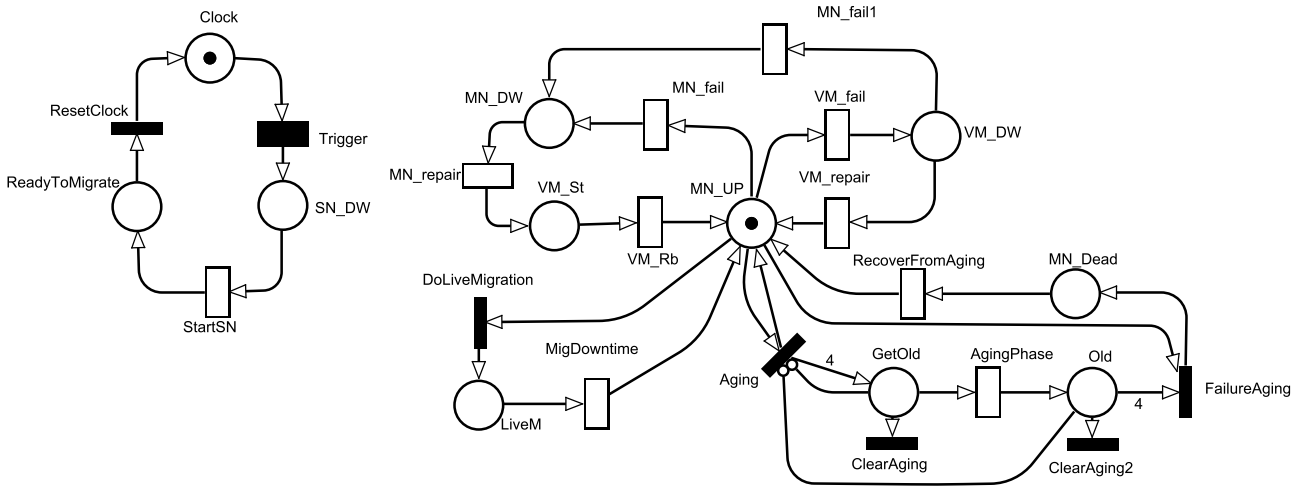


Figura 4.8: SPN com mecanismo de redundância cold-standby

ou seja, após o disparo de `StartSN`, um *token* é depositado no lugar `ReadyToMigrate`, que indica que a migração está habilitada a ser iniciada. A Figura 4.9 descreve a estratégia descrita.

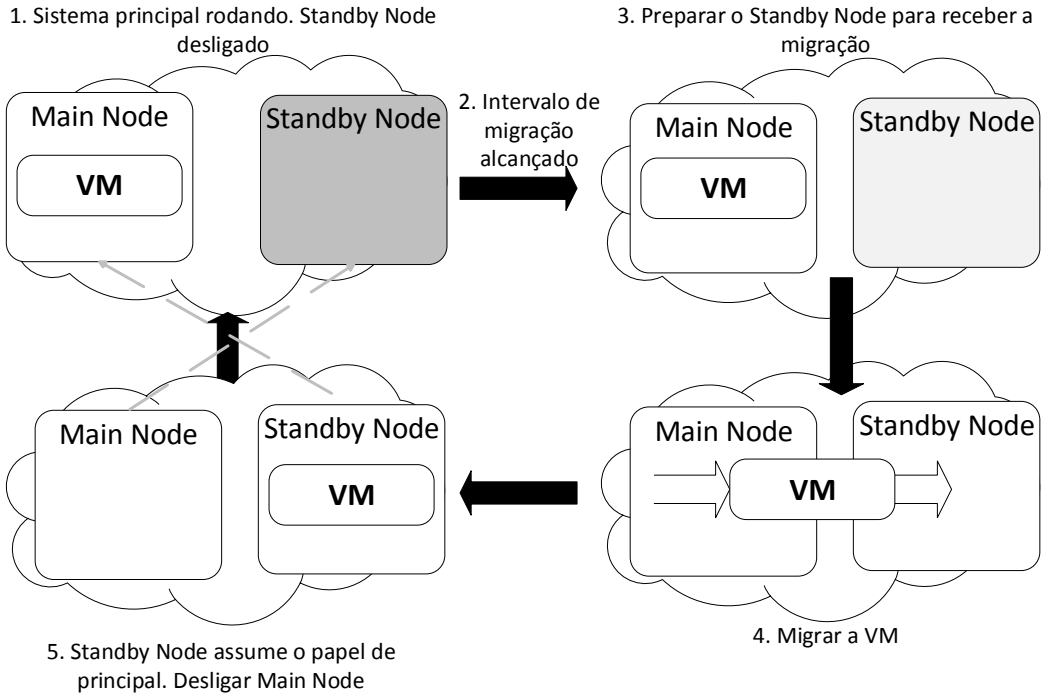


Figura 4.9: Fluxo do Cold Standby no Ambiente

É importante destacar que, para este modelo, considera-se que durante o período de ativação do *Standby Node* o *Main Node* e a VM não interrompem suas operações. O tempo de ativação apenas retarda a habilitação do evento de migração (transição `DoLiveMigration`). Além disso, considera-se que não ocorrem falhas no *Standby Node* até ele receber a VM através da migração. O tempo de desligamento do sistema principal após a migração da VM também é ignorado.

4.3 Considerações finais

Este capítulo apresentou um conjunto de modelos responsável por representar os comportamentos de falhas e reparos, agendamento de migrações, migração de máquinas virtuais e envelhecimento e rejuvenescimento de software. Devido à complexidade destes comportamentos, os modelos foram concebidos adotando a estratégia hierárquica, que permitem atrelar sub-modelos a um modelo principal. Inicialmente, foi construído um modelo para representar os eventos de falha e reparo não relativos ao envelhecimento em uma infraestrutura básica de nuvem. Esse modelo serve de insumo para construção dos modelos envolvendo o envelhecimento e rejuvenescimento de software. Para robustecer a qualidade dos resultados obtidos dos modelos, decidiu-se realizar um processo de validação baseado em experimentação em um ambiente de testes real.

Após a construção dos modelos da infraestrutura básica da nuvem, o comportamento de falhas e reparos foi agregado a outro modelo principal, que visa representar o comportamento de envelhecimento e rejuvenescimento de software. O modelo eDSPN foi construído para representar o comportamento de envelhecimento e rejuvenescimento de software habilitado por agendamento de migrações. O intuito desse modelo é possibilitar o estudo do impacto de diferentes agendamentos de migração na disponibilidade estacionária do sistema. As alterações nos parâmetros possibilita o estudo de diversos cenários, com cargas de envelhecimento diferentes e políticas de migração diversas.

Ao fim do capítulo é apresentado, ainda, um modelo capaz de representar o mecanismo de redundância *cold-standby*. O propósito da construção deste modelo é a realização de comparações de resultados com o modelo eDSPN anterior, que representa o mecanismo *warm-standby*, visando encontrar resultados adequados para a disponibilidade estacionária do sistema.

5

Estudos de caso

Este capítulo apresenta a aplicação dos modelos concebidos no Capítulo 4. O capítulo é dividido em quatro estudos de caso diferentes. O primeiro estudo de caso propõe uma análise dos modelos da infraestrutura básica do ambiente de nuvem (*FrontEnd*, Nó e uma VM) considera um cenário de utilização sob condições básicas (sem redundância e com os parâmetros obtidos da literatura) e apresenta análise de sensibilidade nos parâmetros considerados. O estudo de caso 2 apresenta análises realizadas nos modelos com envelhecimento e rejuvenescimento as análises são efetuadas para encontrar os melhores intervalos de tempo entre migrações, para maximizar a disponibilidade do sistema. O estudo de caso 3 compara diferentes métodos de redundância utilizados para aumentar a disponibilidade dos sistemas, especificamente os mecanismos de *warm* e *cold* standby. O último estudo de caso, o estudo de caso 4, avalia o impacto do *downtime* da migração das VMs na disponibilidade do sistema e o segundo .

5.1 Estudo de caso 1

Estudos de disponibilidade na infraestrutura básica da nuvem são particularmente úteis para diversos propósitos. Para tanto, os modelos apresentados no Capítulo 4 pretendem avaliar a disponibilidade destes ambientes. Como os modelos consideram a infraestrutura básica de nuvem (um *FrontEnd*, um nó e uma *VM*), as análises de disponibilidade realizadas podem servir como uma referência para estudar efeitos de alterações no ambiente, como acréscimo de hardware, mudança nas condições de falha e reparo, bem como fornecer uma visão do comportamento do ambiente sob condições normais (sem redundância e utilizando os parâmetros *baseline*).

Sendo assim, o primeiro estudo de caso propõe análises de disponibilidade nos modelos da infraestrutura básica (ver Figura 4.2) e busca definir os componentes críticos para a disponibilidade do sistema. Assim, os administradores de ambientes de nuvem podem ter uma referência para efetuar possíveis alterações para maximizar a disponibilidade.

As análises iniciais foram conduzidas a fim de obter-se a disponibilidade estacionária do sistema. O intuito é fornecer um panorama geral do estado de disponibilidade do sistema sob uma condição padrão, utilizando parâmetros *baseline*.

Tabela 5.1: Parâmetros *baseline*

Parâmetro	Descrição	Taxa (1/h)	Tempo
α_vm	Taxa de *restart* da VM	12	5 min
λ_vm	Taxa de falha da VM	0,000347222	2880 h
μ_vm	Taxa de reparo da VM	2	30 min
λ_node	Taxa de falha do Nó	0,000808604	1236,7 h
μ_node	Taxa de reparo do Nó	0,917431193	1,09 h
λ_FrontEnd	Taxa de falha do FrontEnd	0,002076843	481,5 h
μ_FrontEnd	Taxa de reparo do FrontEnd	0,970873786	1,03 h

Tabela 5.2: Resultados dos parâmetros *baseline*

Disponibilidade estacionária	*Downtime* h/ano
0,996746602	28,49976263

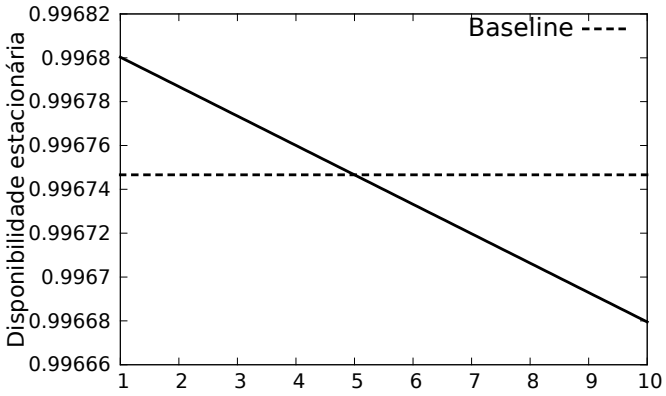
Para a análise dos modelos hieráquicos baseados em RBD utilizou-se a ferramenta Mercury (SILVA et al., 2013) e para a avaliação da CTMC utilizou-se a ferramenta SHARPE (TRIVEDI, 2002). Os parâmetros utilizados na avaliação foram recuperados de trabalhos já publicados (MELO et al., 2013). Os resultados obtidos do modelo utilizando os parâmetros *baseline* da Tabela 5.1 estão discriminados na Tabela 5.2.

Os resultados revelam a disponibilidade estacionária do sistema utilizando os parâmetros *baseline*. Entende-se que esta é a disponibilidade esperada para uma infraestrutura básica de nuvem, ou seja, esta infraestrutura não considera nenhum mecanismo de redundância ou tolerância a falhas.

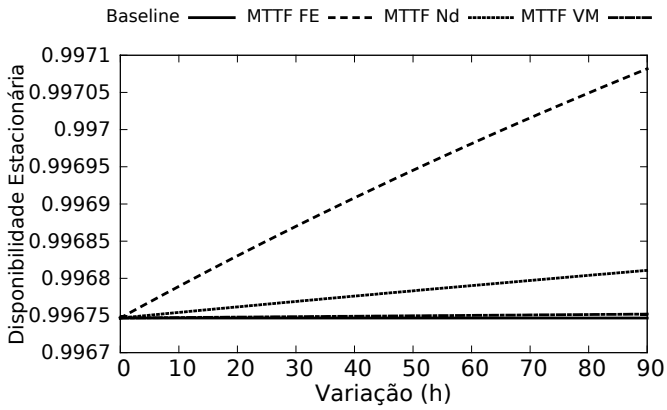
A análise de sensibilidade permite entender o impacto da mudança de cada parâmetro dos modelos na disponibilidade estacionária (MATOS et al., 2012). Isso é particularmente útil quando o administrador precisa saber a importância de um componente em relação à disponibilidade do sistema.

Partindo dos parâmetros *baseline*, foi realizado um estudo de sensibilidade em cada um dos parâmetros do modelo. Para tanto, realizou-se alterações incrementais nos os tempos de falha, reparo e *restart*. Os tempo de falha foram alterados acrescentando 10 horas a cada iteração da avaliação, e o tempo de *restart* da VM foi alterado 1 minuto a cada avaliação. Os resultados podem ser vistos na Figura 5.1. Para simplificar a visualização dos resultados da variação do tempo de falha e reparo, utilizou-se as seguintes abreviações: FE, corresponde ao FrontEnd; Nd, corresponde ao nó e VM, que corresponde a máquina virtual (VM).

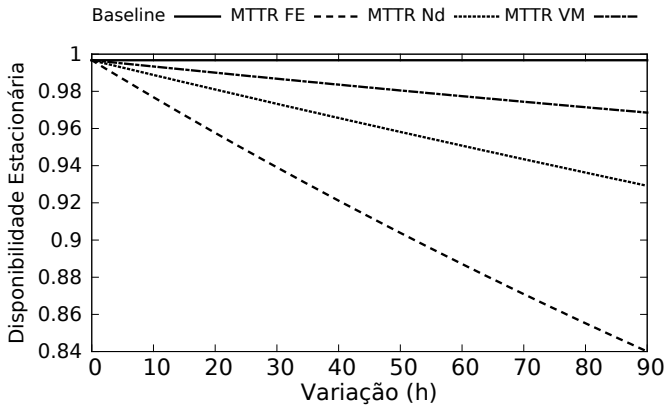
Com a análise de sensibilidade é possível notar que a mudança dos parâmetros afeta de modo diferente a disponibilidade do sistema. Porém, notoriamente a alteração nos tempos de reparo causam maior impacto na disponibilidade que os tempos de falha. Isso mostra que em quesito de disponibilidade, nas condições específicas consideradas pelo modelo. Um reparo feito de modo mais ágil pode ser mais eficaz que um sistema mais tolerante a falhas (MTTF maior). Para destacar isso, estudou-se a variação instantânea que cada um dos parâmetros de falha e



(a) Tempo para *restart* VM(min)



(b) MTTF)

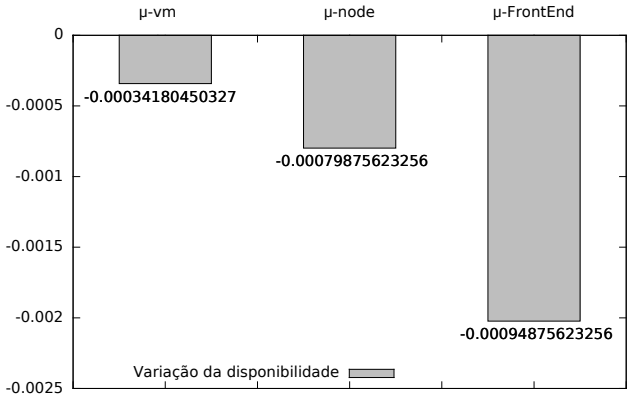


(c) MTTR

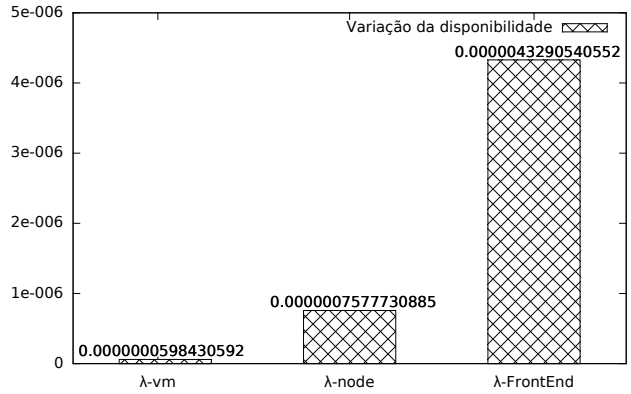
Figura 5.1: Análise de sensibilidade de cada parâmetro

reparo causa na disponibilidade. Partindo dos valores de parâmetros *baseline*, alterou-se os tempos de falha e reparo em mais 10 horas e observou-se o impacto imediato na disponibilidade. Os resultados podem ser vistos na Figura 5.2.

É possível concluir que o componente mais crítico no ambiente estudado é o *FrontEnd*, seguido pelo Nó e por último a *VM*. Decisões administrativas podem ser tomadas para aumentar a disponibilidade baseando-se neste índice de importância. Vale ressaltar que estes resultados levam em consideração o modo operacional descrito na Seção 4.1.



(a) Reparo



(b) Falha

Figura 5.2: Variação na disponibilidade causada por cada MTTF e MTTR

Outra conclusão relevante da análise de sensibilidade é que, observando a Figura 5.2, o impacto na disponibilidade causado pelo tempo de reparo é maior do que o tempo de falha. É possível perceber tal conclusão observando a variação causada pela variação de cada um dos parâmetros. Ou seja, a partir destes resultados, nas condições apresentadas, um reparo mais rápido pode ser mais eficaz que um sistema mais tolerante a falhas no que diz respeito à disponibilidade estacionária do sistema.

5.2 Estudo de caso 2

Para este estudo de caso foram elaborados diferentes cenários com o intuito de determinar as políticas de rejuvenescimento adequadas para melhorar a disponibilidade do sistema. As políticas de rejuvenescimento consideradas são baseadas no agendamento de migrações de máquinas virtuais.

Os efeitos relativos ao envelhecimento de software podem acarretar falhas e travamentos em sistemas computacionais, comprometendo assim a disponibilidade estacionária. Logo, definir métodos de rejuvenescimento que mitiguem os efeitos de envelhecimento de software são de particular interesse. As operações típicas de rejuvenescimento são *reboot* de sistemas

operacionais e *restart* de serviços, que causam paralisações no sistema. É necessário, portanto, observar os intervalos de tempo entre migrações e definir agendamentos adequados para melhorar a disponibilidade do sistema. Realizar migrações com muita recorrência, ou seja, com muitas repetições, podem prejudicar a disponibilidade do sistema. Longos intervalos de tempo entre migrações podem permitir falhas por envelhecimento de software.

O estudo foi conduzido considerando o modelo hierárquico apresentado na Seção 4.2. A eDSPN dos nós está reposta na Figura 5.3 a seguir. O principal objetivo é avaliar o impacto dos diferentes agendamentos de migração na disponibilidade estacionária do sistema. Para alcançar esse objetivo, conceberam-se cinco diferentes cenários. Os cenários foram avaliados para encontrar a disponibilidade do sistema e o *downtime* anual. O agendamento de migrações adequado foi encontrado para cada cenário. A diferença entre os cenários está no tempo para falha relativa ao envelhecimento (*TTARF*). Estudos mostram que a manifestação de envelhecimento está ligada à carga que é submetida ao sistema (BAO; SUN; TRIVEDI, 2005), então os cenários concebidos representam diferentes intensidades de carga de trabalho submetidas à nuvem. A configuração de cada cenário está descrita na Tabela 5.3. O tempo atribuído para a transição `AgingPhase` para cada cenário também está descrito na Tabela, os valores foram atribuídos baseando-se nos estudos apresentados em ARAUJO (2012).

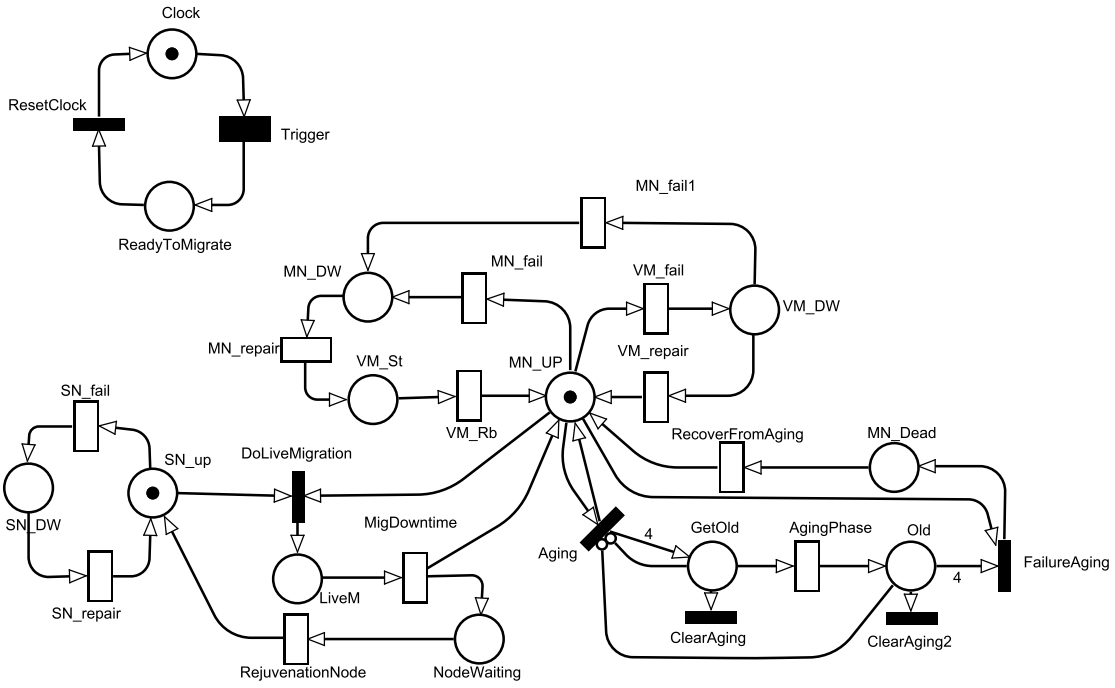


Figura 5.3: eDSPN CloudNodes

Os modelos foram avaliados utilizando a ferramenta *SHARPE* (TRIVEDI, 2002) para os *RBD*s e a ferramenta *TimeNet* para os modelos *SPN* (GERMAN et al., 1995). Os parâmetros utilizados para avaliação do bloco *FrontEnd* são os mesmos que foram apresentados em 4.2. Os parâmetros para o bloco *CloudNodes* estão descritos na Tabela 5.4, estes foram obtidos de (MELO et al., 2013; MACHIDA; KIM; TRIVEDI, 2010).

Tabela 5.3: Definição dos cenários

Cenário #	TTARF (h)	`AgingPhase (h)`
1	100	25
2	250	62,5
3	500	125
4	750	187,5
5	1000	250

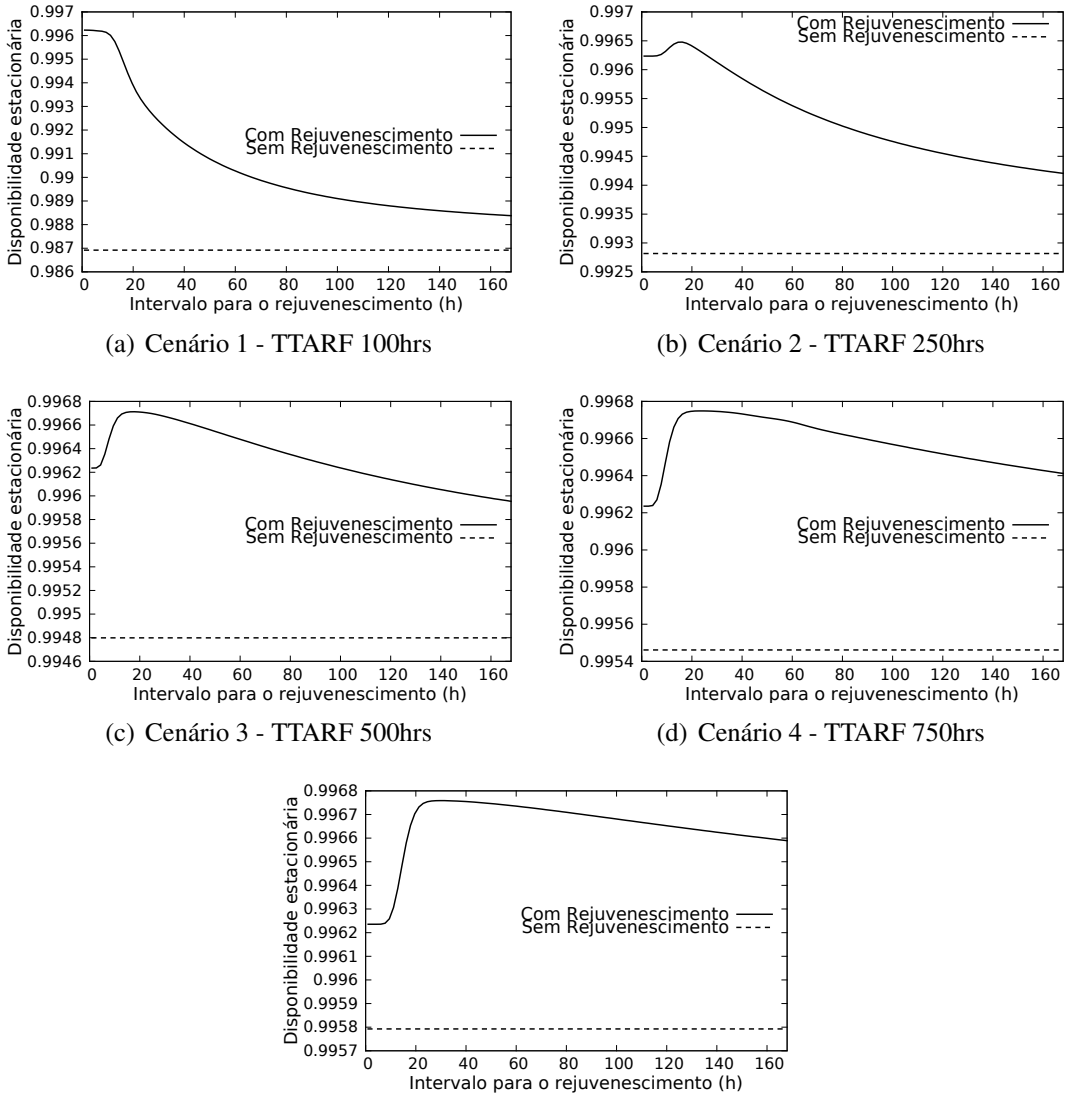
Tabela 5.4: Parâmetros SPN CloudNodes

Parâmetros		**Valores**
Transição	***Descrição***	***Tempo Estimado***
`MN_fail`(1)	Falha interna no *Main Node*	1236.7 h
`MN_repair`	Reparo no *Main Node*	1.09 h
`SN_fail`	Falha interna no *Standby Node*	1236.7 h
`SN_repair`	Reparo no *Standby Node*	1.09 h
`VM_fail`	Falha na *VM*	2880 h
`VM_repair`	Reparo na *VM*	30 min
`VM_Rb`	*Reboot* da *VM*	5 min
`AgingPhase`	Tempo para envelhecer	Vide Tabela 5.3
`RecoverFromAging`	Reparo após falha por envelhecimento	1 h
`MigDowntime`	Interrupção do *Live Migration*	2 s
`RejuvenationNode`	Tempo para rejuvenescer o Nó	2 min

Nas avaliações de cada cenário, o intervalo de disparo de rejuvenescimento variou de 1h até 168h (uma semana), pois foi observado em avaliações anteriores que este tempo era suficiente para revelar o intervalo adequado para cada cenário. Este tempo corresponde ao tempo médio atribuído ao disparo da transição `Trigger` do modelo. Os gráficos incluem também a disponibilidade obtida por um modelo sem as políticas de rejuvenescimento. Os resultados podem ser vistos na Figura 5.4.

É perceptível que para todos os cenários e políticas utilizados, a disponibilidade do sistema com rejuvenescimento supera a disponibilidade sem rejuvenescimento. A disponibilidade decresce após alcançar um específico valor máximo. O intervalo de rejuvenescimento apropriado, referente ao pico da disponibilidade, é uma relevante conclusão obtida da análise de sensibilidade. A Tabela 5.5 mostra os intervalos de migração que proporcionam a melhor disponibilidade para o sistema. É notório que nos cenários com cargas mais fracas (4 e 5) a migração deve ser disparada em intervalos de tempo maiores do que nos cenários com cargas mais fortes.

Para destacar o impacto do rejuvenescimento proposto foi computado o percentual ganho de disponibilidade. Esta métrica revela o quanto a disponibilidade estacionária é melhorada a partir da utilização do rejuvenescimento proposto. Ela é computada comparando a disponibilidade do ambiente sem rejuvenescimento com a disponibilidade considerando-se as políticas de migração. Esta diferença, expressa em percentual, é exibida no eixo das ordenadas dos gráficos. As diferentes políticas de rejuvenescimento são representadas pelo eixo das abscissas. A Figura 5.5 exibe os resultados.



(a) Cenário 1 - TTARF 100hrs

(b) Cenário 2 - TTARF 250hrs

(c) Cenário 3 - TTARF 500hrs

(d) Cenário 4 - TTARF 750hrs

(e) Cenário 5 - TTARF 1000hrs

Figura 5.4: Análise de sensibilidade das políticas de rejuvenescimento na disponibilidade estacionária

O gráfico permite perceber os diferentes impactos causados pelos agendamentos na disponibilidade do sistema. No Cenário 1, com o menor *TTARF*, o rejuvenescimento produz ganhos significativos: aumentando a disponibilidade estacionária em quase 1%. Quando o intervalo entre migrações é modificado, a disponibilidade estacionária é alterada também, porém, há diferenças entre o comportamento da curva de disponibilidade em cada cenário. Em todos os cenários existe uma política de migrações que produz o melhor resultado para a disponibilidade do sistema. Em alguns casos, a migração em excesso degrada a disponibilidade drasticamente, como pode ser visto na Figura 5.4(e), quando o intervalo entre migrações é menor que 20 horas. Por outro lado, em todos os cenários estudados é possível notar que a disponibilidade tende para a situação do ambiente sem rejuvenescimento conforme o intervalo entre migrações cresce.

É importante notar que para cada ambiente estudado, o agendamento de migrações

Tabela 5.5: Resultados da análise dos modelos

Cenário	`Trigger`	Disponibilidade	*Downtime* (h/ano)
1	1 h	0,9962293	33,031332
2	13 h	0,9966147	29,655228
3	21 h	0,9967160	28,767840
4	21 h	0,9967502	28,468248
5	28 h	0,9967600	28,382400

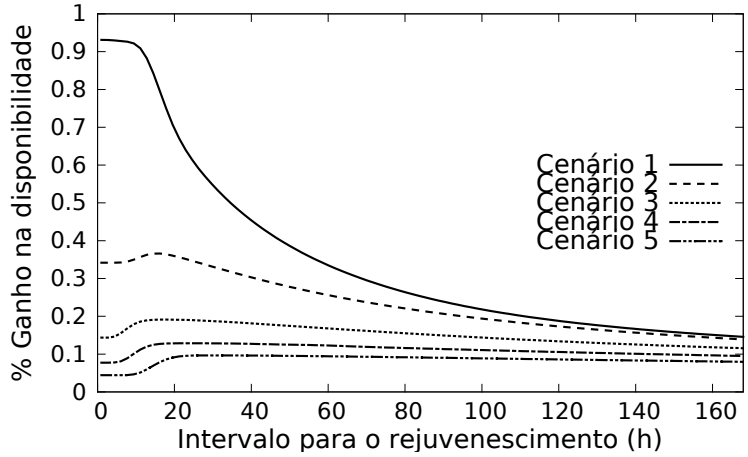


Figura 5.5: Percentual de ganho de disponibilidade para cada cenário

fornece um pico de disponibilidade, e após este, a disponibilidade decai, tendendo à disponibilidade do ambiente sem rejuvenescimento (*baseline*). Nos cenários estudados, a maioria dos intervalos entre migrações é menor que 24 horas. Logo, é possível concluir que para a maioria dos casos, realizar pelo menos um rejuvenescimento por dia constitui uma prática para melhorar a disponibilidade do sistema como um todo.

Com o modelo é possível observar intervalos adequados de migração, estes podem ser embutidos em alguma ferramenta que gerencie as migrações, para poder assim, aumentar a disponibilidade do sistema. Em contrapartida, o agendamento não fornece um mecanismo dinâmico de adaptação, ou seja, o agendamento é estático e não adaptativo para a carga do sistema. Assim sendo, antes de estipular o intervalo entre migrações é necessário entender a natureza das cargas que são submetidas ao sistema. Além disto, encontrar o *TTARF* dos ambientes por meio de experimentação pode ser um caminho viável para determinar os intervalos entre migrações com mais confiança.

5.3 Estudo de caso 3

Visando comparar as abordagens de redundância *cold standby* e *warm standby*, este estudo de caso apresenta resultados utilizando os dois modos de redundância supracitados.

O mecanismo de rejuvenescimento habilitado por migração exibido no modelo eDSPN *CloudNodes* (Figura 5.3) do capítulo anterior é similar ao mecanismo de redundância *warm-*

standby. O mecanismo de redundância *warm-standby* consiste em uma situação onde existe um nó reserva que está ligado e funcionando, porém não pode assumir o papel de principal imediatamente. É necessário executar algumas operações e preparar o nó reserva para assumir o papel de principal. Porém, tais operações requerem um tempo para serem realizadas, tempo este, denominado de tempo de ativação (DANTAS et al., 2012b). Fazendo um paralelo com o modelo, este tempo de ativação pode ser considerado como o tempo de migração da *VM* para o *Standby Node*. Mesmo o *Standby Node* com todos os componentes funcionando corretamente, é necessária a migração para que ele assuma o papel de *Main Node*. Além disso, no mecanismo *warm-standby*, considera-se que o *Standby Node* pode falhar por algum motivo, pois, mesmo que ele não esteja atuando como principal, seus componentes estão ativos e podem falhar.

No mecanismo *cold-standby*, representado pelo modelo SPN da Figura 5.6, o nó reserva fica inativo durante a utilização do sistema principal. Quando é preciso utilizar o nó reserva, deve-se iniciar o sistema e carregar todos os componentes necessários para que ele assuma o papel de principal (DANTAS et al., 2012b). Em geral, o tempo de ativação deste nó em *cold-standby* é maior do que o tempo de ativação do nó em *warm-standby*. Além disso, considera-se que o nó reserva não sofre falhas, pois seus componentes estão inativos.

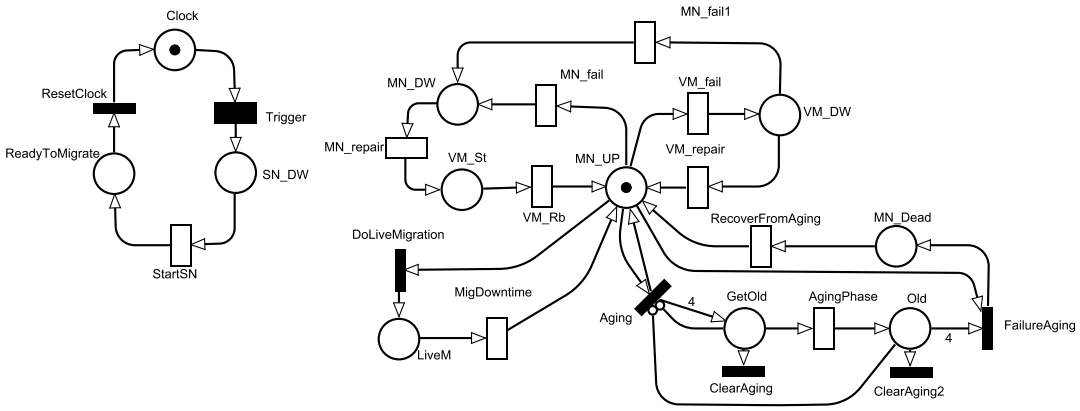


Figura 5.6: SPN com mecanismo de redundância cold-standby

No ambiente de nuvem considerado, para representar o mecanismo *cold-standby*, é necessário que o nó reserva *Standby Node* esteja desligado quando a migração for solicitada. Porém, neste caso específico, o sistema não interromperá suas atividades, apenas o processo de rejuvenescimento terá que ser suspenso até a máquina *Standby Node* iniciar por completo. Ou seja, assim que o agendamento for atingido, o processo de preparar o nó reserva *Standby Node* será iniciado. Todavia, neste intervalo de tempo, a *VM* continua funcionando no *Main Node*. Quando o nó estiver preparado, o rejuvenescimento poderá ocorrer normalmente.

O tempo de ativação adotado para o nó reserva foi de 5 minutos (KIM; MACHIDA; TRIVEDI, 2009). Os demais parâmetros permaneceram os mesmos do estudo de caso anterior, apresentados na Tabela 5.4. Os intervalos de rejuvenescimento adotados foram os exibidos na Tabela 5.5, que visam maximizar a disponibilidade do ambiente. Os cenários propostos seguem as mesmas configurações do Estudo de Caso 2, apresentadas na Tabela 5.3.

Os resultados mostram a comparação dos resultados deste estudo de caso (*cold-standby*) com os resultados do estudo de caso anterior (*warm-standby*). Tais resultados podem ser vistos na Figura 5.7.

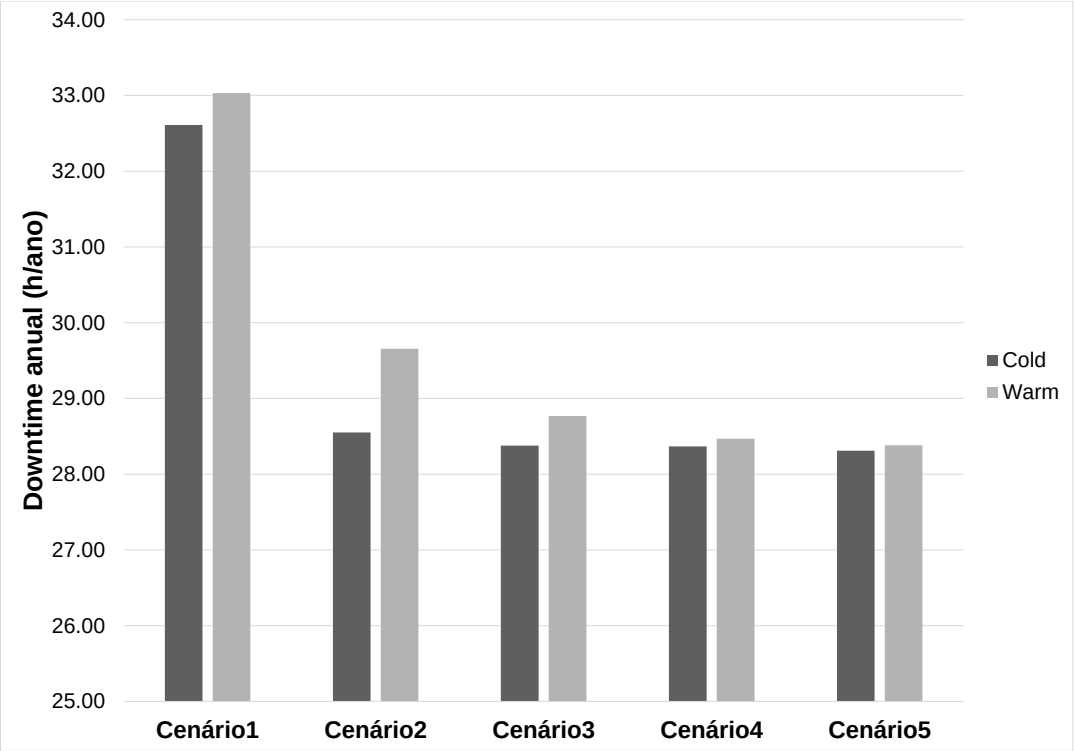


Figura 5.7: Comparação das abordagens de redundância

Os resultados mostram que, em todos os casos, o mecanismo de *cold-standby* superou o mecanismo de *warm-standby* em relação à disponibilidade estacionária do sistema. Nos cenários 1, 2 e 3 (*TTARF*s menores), a diferença entre as disponibilidades é mais perceptível que nos demais, isso ocorre porque os intervalos entre migrações são menores nestes cenários.

Com os resultados é possível concluir que, com o sistema em *warm-standby*, as migrações são mais recorrentes do que nos sistemas em *cold-standby*, por causa do retardo do tempo de ativação. Percebe-se que, nos cenários onde o intervalo entre migrações é menor, a frequência de migrações afeta de modo substancial a disponibilidade do sistema. Em alguns cenários percebe-se que há um excesso de migrações desnecessárias, quando o ambiente utiliza o mecanismo *warm-standby*. Com o retardo maior para ativação do sistema reserva (*cold-standby*), o número total de migrações é reduzido, o que melhora a disponibilidade do ambiente. Porém, em cenários com tempos entre migrações maiores, a diferença entre as abordagens é quase imperceptível.

5.4 Estudo de caso 4

Este estudo analisa o impacto do *downtime* associado à migração na disponibilidade do sistema. Os resultados podem ser úteis para determinar a importância da reserva de banda para

migração, ou ainda, o quão nocivo é um downtime associado à migração mais longo para cada cenário estudado.

Devido à instabilidade de alguns ambientes de rede e ao fato de que, geralmente, a rede utilizada para os ambientes de nuvem está sempre transportando dados, é muito difícil estabelecer com precisão o intervalo de tempo de uma migração de uma *VM*. Conforme visto na Seção 2.4, o processo de migração via *live migration* é iterativo e contínuo. Após a fase de cópia das páginas de memória da VM, há um momento de interrupção. Mesmo que a migração via *live migration* prometa uma migração com interrupção mínima dos serviços, esta interrupção não pode ser negligenciada.

Para conduzir este estudo de caso utilizou-se o modelo que considera o envelhecimento e rejuvenescimento de software exibido na Figura 4.7. Aproveitando-se da mesma configuração de cenários exposta na Tabela 5.3, os estudos foram conduzidos variando o *downtime* associado à migração, para verificar o impacto na disponibilidade (transição `MigDowntime`). Além disso, o intervalo entre migrações considerado é o mesmo que foi encontrado no Estudo de Caso 2 (Tabela 5.5). Os resultados podem ser vistos na Figura 5.8. Os *downtimes* associados à migração selecionados foram 1, 5, 10, 30 e 60 segundos. Estes valores foram atribuídos visando representar diferentes tipos de migração de VM, desde as que acarretam mínima interrupção de 1 segundo (*live migration*) até outras que trazem maior paralisação (migração parar-e-copiar) com paralisação por volta de 60 segundos.

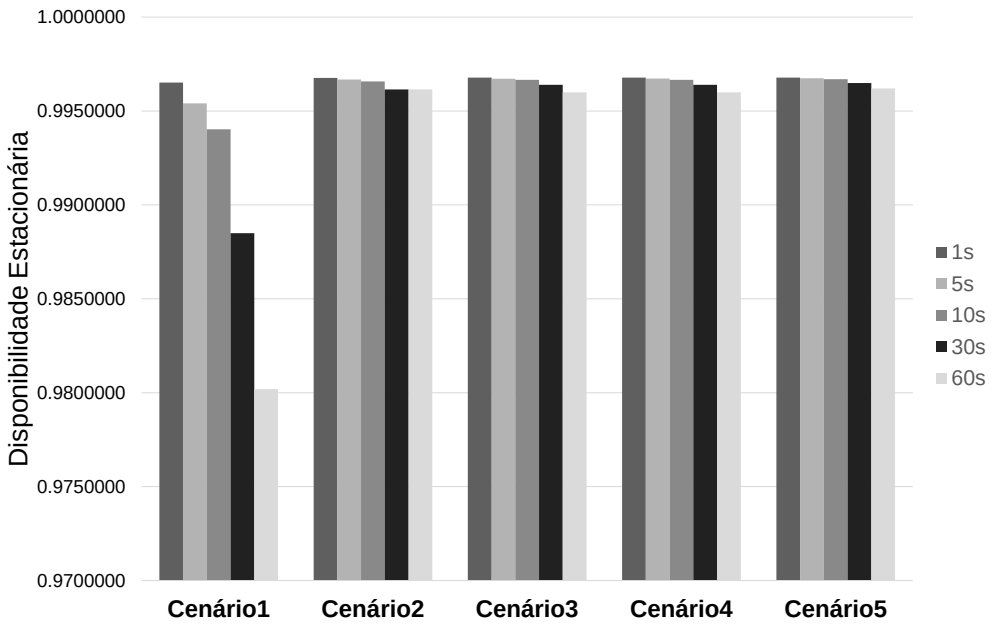


Figura 5.8: Sensibilidade do *downtime* das migrações

Os resultados mostram que nos cenários com carga mais intensa (*TTARF* menor) os impactos do *downtime* associados às migrações são mais visíveis que nos demais casos. Com estes resultados é possível observar a importância do desempenho da migração em cada cenário estudado.

Algumas conclusões podem ser obtidas deste estudo de caso. Primeiramente, é notório que o investimento em melhorias do desempenho da migração é mais efetivo nos cenários com *TTARF* menor. Ou seja, nestes, a duração do *downtime* associado à migração é mais sensível para a disponibilidade do sistema. Em todos os casos, a migração com *downtime* mínimo (1 segundo) produz o melhor efeito para a disponibilidade, porém, para obter esses resultados é necessário criar um ambiente que garanta um ótimo desempenho nos componentes envolvidos na migração. Outra conclusão interessante é que, em ambientes com cargas mais leves, (*TTARF* maior), o impacto do *downtime* associado às migrações é mais suave. Isto acontece porque nestes o tempo entre migrações é maior. Por causa disso é possível dizer que, em ambientes com *SLAs* mais livres, talvez seja melhor optar pelas migrações parar-e-copiar que possuem *downtime* maior para evitar o *overhead* causado pelo envio de cópias sucessivas realizado pelo *live migration*.

5.5 Considerações finais

Este capítulo apresentou análises e resultados obtidos a partir de modelos analíticos para avaliação de disponibilidade em ambientes de nuvem. O intuito é mostrar a aplicabilidade dos modelos propostos.

Os resultados dos modelos para a infraestrutura básica mostram que, considerando os modos operacional descrito, o *FrontEnd* é o componente mais crítico para os ambientes de computação em nuvem. Vale a pena ressaltar que estes resultados basearam-se em modelos que foram validados por injeção de falhas e reparos.

As análises dos modelos com rejuvenescimento habilitado por migração de VMs permitem a escolha de intervalos de rejuvenescimento adequados para cada cenário de estudo, melhorando assim, a disponibilidade estacionária do sistema. Os resultados mostram que o rejuvenescimento a partir de migração de máquinas virtuais pode ser útil em nuvens que lidam com diferentes intensidades de cargas de trabalho. Em sistemas sob cargas mais intensas, o rejuvenescimento é bastante efetivo, trazendo melhoras substanciais para a disponibilidade. À medida que o intervalo de rejuvenescimento cresce, a disponibilidade do sistema tende a voltar à situação sem rejuvenescimento. Existem casos, ainda, nos quais a migração excessiva é danosa para a disponibilidade.

Os resultados experimentais apresentados anteriormente corroboram a efetividade do método que foi descrito nos modelos analíticos com rejuvenescimento e envelhecimento. Com isso, os resultados dos modelos ficam mais confiáveis e os modelos ficam mais úteis para serem estendidos. Outro ponto importante, que fortalece ainda mais os resultados apresentados, é o fato dos modelos terem sido baseados em outros modelos que passaram por um processo de validação. Todo o comportamento de falhas não relativas ao envelhecimento foi obtido dos modelos apresentados no Capítulo 4.

A escolha apropriada de mecanismos de redundância é um ponto chave para a alta disponibilidade de sistemas. Os resultados no Estudo de Caso 3 mostram que, em todos os casos,

a utilização do mecanismo *cold-standby* promove melhor disponibilidade do que a utilização do mecanismo de *warm-standby*. O uso do mecanismo de *cold-standby* favorece o gerenciamento do ambiente (menos máquinas físicas funcionando ao mesmo tempo), além de economizar gastos com manutenção e energia.

Em alguns cenários, o *downtime* acarretado por cada migração pode acarretar um alto impacto para a disponibilidade estacionária do sistema. Principalmente naqueles com *TTARF* menores. Porém, em ambientes com cargas menos intensas, a variação em segundos do *downtime* não afeta substancialmente a disponibilidade do sistema.

6

Trabalhos Relacionados

Este capítulo apresentará um conjunto de trabalhos com temas relacionados a esta pesquisa. Devido à extensão do trabalho proposto não foram encontrados trabalhos que permeiem todas as áreas cobertas, então, dividiu-se os trabalhos relacionados com cada seção exposta neste trabalho. Foram selecionados apenas os trabalhos que mais se assemelham com o que foi produzido nesta dissertação.

6.1 Modelos para Avaliação de Disponibilidade em Nuvens

A avaliação de disponibilidade em nuvens é uma área amplamente estudada. Existem diversos trabalhos publicados que visam tal objetivo. Nesta seção serão apresentados os trabalhos que mais se aproximam do estudo proposto nesta dissertação.

Em CHUOB; POKHAREL; PARK (2011) são apresentados modelos para avaliação de disponibilidade de nuvens *Eucalyptus*. Os modelos são baseados em *CTMC*s e os autores geram fórmulas fechadas para os cenários estudados. O trabalho também engloba características de rejuvenescimento de software. Um experimento para validar as taxas utilizadas nos modelos é mencionado, mas, nenhuma metodologia sobre a sua utilização é apresentada. Também não são realizadas análises de sensibilidade nos parâmetros utilizados.

Uma modelagem de disponibilidade de alto nível é proposta em WU; HUANG (2013). Neste trabalho são apresentados modelos de alto nível que representam comportamentos de alta disponibilidade em plataformas de nuvem. São feitas comparações entre algumas ferramentas de alta disponibilidade com o que é proposto no trabalho. Por ser um modelo de alto nível, não são geradas fórmulas fechadas para estes. Também não são exibidas as análises de sensibilidade dos parâmetros na disponibilidade estacionária.

Um trabalho bastante relevante no contexto de avaliação de disponibilidade em ambientes virtualizados é apresentado em KIM; MACHIDA; TRIVEDI (2009). Este trabalho contém um estudo extensivo de disponibilidade em servidores virtualizados. Leva-se em consideração o comportamento de falha e reparo de diversos componentes destes ambientes. Também são realizadas análises de sensibilidade em vários parâmetros. Porém, não foi apresentada uma

metodologia para validação dos modelos apresentados, além disso, as fórmulas fechadas são omitidas. Em MATOS et al. (2012) é apresentada uma extensa análise de sensibilidade em parâmetros de modelos para avaliação de disponibilidade de servidores virtualizados.

Modelos baseados em Redes de Petri para avaliação de disponibilidade de nuvens são propostos em LONGO et al. (2011). No trabalho é apresentado um conjunto de macromodelos baseados em PNs e alguns submodelos são gerados em CTMCs. O propósito do trabalho é prover modelos escaláveis para avaliação de disponibilidade em ambientes de computação em nuvem. Para tanto, o conjunto de parâmetros utilizados são genéricos, o que permite adaptações e alterações. Ao final do artigo são exibidas fórmulas fechadas e análise de sensibilidade para alguns parâmetros-chave. Nenhum método de validação dos modelos foi apresentado.

Considerações

A maioria dos trabalhos levantados nesta área de modelagem de disponibilidade em nuvens é voltada à obtenção de alta disponibilidade. Alguns trabalhos propõem métodos e técnicas através dos modelos. Na maioria, não existe uma abordagem utilizada para validar os modelos ou testar as técnicas propostas em ambientes reais. Alguns trabalhos apresentam conjuntos de fórmulas fechadas e análise de sensibilidade nos parâmetros, o que pode ser muito útil para diversas ocasiões.

Assim como alguns trabalhos apresentados, esta dissertação apresenta um conjunto de modelos para avaliação de disponibilidade de nuvem. Porém, além de propor e validar os modelos de disponibilidade da infraestrutura básica (Nó, VM e FrontEnd), os estudos são aprofundados, considerando-se efeitos de envelhecimento e políticas de rejuvenescimento de software.

6.2 Experimentos de envelhecimento e rejuvenescimento

Artigos com experimentação de envelhecimento e rejuvenescimento de software, ao contrário dos modelos para disponibilidade de nuvens, são mais difíceis de encontrar. A seguir, encontram-se os artigos relacionados que foram encontrados até o presente momento da pesquisa.

Os experimentos realizados em MATOS et al. (2012) revelam a presença de envelhecimento de software em plataformas de nuvem *Eucalyptus*. No trabalho são exibidos resultados do monitoramento de diversos recursos diferentes. São feitas correlações entre os componentes do experimento, visando indicar os principais responsáveis pelos efeitos negativos causados pelo envelhecimento. O artigo não cobre abordagens e experimentos de rejuvenescimento.

Um estudo bastante relevante é apresentado em ARAUJO et al. (2011). Os experimentos de envelhecimentos são conduzidos na plataforma *Eucalyptus* e trazem importantes conclusões. Os autores realizaram o monitoramento do vazamento e fragmentação de memória acarretados pelo envelhecimento. Os testes são conduzidos em máquinas com diferentes serviços, e o

envelhecimento é analisado em cada um destes. Por fim, os autores apresentam os testes de rejuvenescimento para a plataforma estudada e confirmam, nos resultados, a efetividade de seus métodos. Os autores focam na utilização de recursos em suas avaliações, não há resultados sobre a degradação de desempenho de aplicações.

Outros experimentos de envelhecimento de software em outros ambientes são apresentados na literatura (GROTTKE et al., 2006; MATIAS et al., 2006). Estes visam mostrar indícios de envelhecimento de software em servidores Web. São feitos testes e os recursos computacionais são monitorados em busca de indícios de envelhecimento.

Considerações

Existem poucos trabalhos publicados sobre experimentação de envelhecimento e rejuvenescimento de software em plataformas de computação em nuvem, o que aumenta a relevância dos testes realizados nesta pesquisa. Dentre os trabalhos relacionados, o que mais se aproxima dos experimentos de envelhecimento e rejuvenescimento realizados nesta dissertação, é o ARAUJO et al. (2011). Outros trabalhos de experimentos de envelhecimento aparecem em publicações, porém a maioria não é voltado para computação em nuvem. Alguns trabalhos, ainda, não apresentam metodologia e testes para o rejuvenescimento de software.

6.3 Modelagem de rejuvenescimento em ambientes de nuvem

Há alguns trabalhos publicados que consideram o rejuvenescimento de software em plataformas de nuvem. Alguns destes consideram o rejuvenescimento de software habilitado por migração de máquinas virtuais. A maioria deles visa encontrar intervalos de tempo ideais para desempenhar rejuvenescimentos em ambientes virtualizados. Todos os trabalhos levantados são baseados exclusivamente em modelagem analítica.

Em HANMER; MENDIRATTA (2010) é proposto um método de rejuvenescimento baseado na migração de carga de trabalho. São elaborados modelos com diferentes quantidades de estados para representar diferentes cenários de estudo. Cada modelo é avaliado considerando diferentes cenários. Neste trabalho não se leva em consideração o comportamento do nó físico, somente o da *VM*.

Com modelagem de rejuvenescimento baseada em *CTMC*, o trabalho THEIN; CHI; PARK (2008) apresenta um conjunto de resultados que contempla diferentes possibilidades. A metodologia de rejuvenescimento é baseada no comportamento de ferramentas para alta disponibilidade. Fórmulas fechadas são geradas a partir da *CTMC* utilizada. O modelo leva em consideração apenas os macroestados do processo de envelhecimento e rejuvenescimento (ativo, instável, rejuvenescendo, falha). Falhas de outras naturezas não são consideradas. Existe ainda um processo de validação numérico, que confronta os resultados das fórmulas com os resultados obtidos dos modelos por intermédio da ferramenta *SHARPE* (TRIVEDI, 2002).

Outro trabalho relevante é apresentado em BRUNEO et al. (2013). Neste, são apresentados diferentes modelos que contemplam variadas formas de rejuvenescimento. Os autores, além de representar o rejuvenescimento baseado em tempo, mostram outra técnica que é adaptativa com a carga que o sistema se submete. Os modelos apresentados são baseados em *CTMC*s. Ao fim, as duas políticas de rejuvenescimento são comparadas em termos de disponibilidade estacionária. No trabalho só é considerado o envelhecimento no *VMM*, e possíveis falhas de outras naturezas são desprezadas.

Para representar um sistema com gerenciamento automático de rejuvenescimento, o trabalho PAING; MYAT; THEIN (2012) apresenta um conjunto de Redes de Petri. Para avaliar a disponibilidade do sistema, o grafo de alcançabilidade é gerado visando obter fórmulas fechadas, que são apresentadas logo em seguida. Do mesmo modo que em THEIN; CHI; PARK (2008), a análise numérica é comparada com os resultados obtidos da ferramenta *SHARPE*. Falhas não relacionadas ao envelhecimento são negligenciadas.

Por fim, o trabalho MACHIDA; KIM; TRIVEDI (2013) que apresenta modelos para avaliação de disponibilidade em nuvens com rejuvenescimento habilitado por migração de máquinas virtuais. No trabalho são propostos diversos cenários e políticas diferentes de migração. Os resultados focam na disponibilidade estacionária, no número de transações perdidas e nas comparações entre as políticas de rejuvenescimento estabelecidas. Uma extensiva análise de sensibilidade é apresentada, destacando os intervalos ideais para o rejuvenescimento nos cenários estudados. O artigo diz que o processo de validação é um trabalho futuro, o rejuvenescimento adotado engloba a *VM* e o *VMM*. O modelo apresentado no Capítulo 5 absorve algumas características dos modelos apresentados nesse artigo, porém, nos modelos apresentados nessa dissertação, as falhas não relativas ao envelhecimento de software também são consideradas.

Considerações

A maioria dos trabalhos apresentados visa a alta disponibilidade dos ambientes de nuvem. Alguns adotam diferentes metodologias de rejuvenescimento para este fim. As falhas não relativas ao envelhecimento são negligenciadas na maioria dos trabalhos levantados. Dentre esses trabalhos, pode-se destacar como trabalho correlato mais aproximado com esta dissertação o que foi apresentado em (MACHIDA; KIM; TRIVEDI, 2013). Os modelos apresentados por (MACHIDA; KIM; TRIVEDI, 2013) serviram de referência para os que são apresentados nesta dissertação. Enquanto o artigo engloba outras métricas de estudo e diferentes políticas de rejuvenescimento, este trabalho apresenta um estudo que considera as falhas de outras naturezas não correlatas ao envelhecimento de software. Além disto, no artigo não foram realizados experimentos para validar o modelo apresentado.

6.4 Considerações Finais

A partir dos trabalhos levantados, a Tabela 6.1 propõe um paralelo entre eles e o trabalho proposto nesta dissertação. O intuito é destacar os pontos fortes e limitações deste trabalho em comparação com os demais que já foram publicados.

É possível notar que, devido à extensão deste trabalho, não foi encontrado algum outro que cobrisse todas as áreas de estudo aplicadas aqui. Porém, existem algumas limitações em relação aos detalhes de cada uma das fases do desenvolvimento do trabalho. Pondo-se em paralelo com trabalhos específicos, é possível notar que existem pontos específicos que não foram aprofundados da mesma maneira que os demais. Isso se justifica pelo propósito específico desta dissertação.

Contudo, a relevância deste trabalho torna-se notória quando ele é colocado em paralelo com os trabalhos levantados, justamente por sua extensão de estudo. O trabalho considera desde modelos simples até modelos com envelhecimento e rejuvenescimento de software, acrescentando a isso um processo de validação de modelos de disponibilidade e experimentos de envelhecimento e rejuvenescimento de software.

Tabela 6.1: Trabalhos relacionados

Trabalho	**Modelos de disponibilidade para nuvens**	**Validação**	**Experimentos de envelhecimento e/ou rejuvenescimento**	**Modelos de disponibilidade considerando envelhecimento e rejuvenescimento**
(CHUOB; POKHAREL; PARK, 2011)	Sim, com fórmulas fechadas	Sim	Não	Sim, apenas rejuvenescimento
(WU; HUANG, 2013)	Sim	Não	Não	Não
(KIM; MACHIDA; TRIVEDI, 2009)	Sim, considerando diversos componentes.	Não	Não	Não
(MATOS et al., 2012)	Sim, com fórmulas fechadas e análise de sensibilidade.	Não	Não	Não
(LONGO et al., 2011)	Sim, com fórmulas fechadas e análise de sensibilidade.	Não	Não	Não
(MATOS et al., 2012)	Não	Não	Sim, apenas envelhecimento. Monitoramento em diversos recursos.	Não
(ARAUJO et al., 2011)	Não	Não	Sim, ambos. Monitoramento em diversos recursos	Não
(HANMER; MENDIRATTA, 2010)	Sim	Não	Não	Sim, considerando envelhecimento na *VM*
(PAING; MYAT; THEIN, 2012)	Sim, com fórmulas fechadas	Não	Não	Sim
(MACHIDA; KIM; TRIVEDI, 2013)	Sim, com análise de sensibilidade	Não	Não	Sim. Considera-se o envelhecimento e rejuvenescimento na *VM* e no *VMM*
Esta Dissertação	Sim, com fórmulas fechadas e análise de sensibilidade	Sim	Sim, ambos. Monitora-se apenas o servidor Web e alguns recursos	Sim. Considera-se apenas o envelhecimento no *VMM*

7

Conclusões e trabalhos futuros

A confiabilidade e disponibilidade de sistemas computacionais são objeto de estudo em diversos centros de pesquisa. Fornecer meios, métodos e mecanismos para manter o sistema funcionando por mais tempo com desempenho aceitável é objetivo de vários estudos. Tais pesquisas são particularmente importantes para os ambientes de computação em nuvem. Alguns trabalhos já publicados mostram que as questões de disponibilidade e confiabilidade ainda são um dos maiores entraves para empresas e organizações migrarem suas aplicações para os ambientes de nuvem. É necessário fornecer meios para que os provedores de nuvem possam estimar a disponibilidade de seus sistemas. Esses métodos devem possibilitar estudos mais aprofundados, permitindo a observação do impacto de uma operação antes de executá-la.

Uma metodologia viável para este fim é a modelagem analítica, onde é possível realizar diversos estudos sem a interferência no ambiente real. Porém, para que os modelos sejam mais confiáveis, é necessário que alguns experimentos sejam realizados para comprovar que o comportamento modelado reflete o sistema real.

Em estudos de desempenho e disponibilidade, é importante observar os eventos e características que estão ligadas a essas duas áreas. Um ponto a ser destacado nesse quesito é o envelhecimento de software. O envelhecimento de software causa degradação dos recursos, podendo levá-los à falha. Sendo assim, contra-medidas devem ser apresentadas para evitar que os efeitos do envelhecimento de software se tornem falhas, alcançando assim maiores níveis de disponibilidade.

Diante dessa problemática, esta dissertação propôs um conjunto de modelos para avaliação de disponibilidade em ambientes de computação em nuvem privados, levando em consideração o envelhecimento e rejuvenescimento de software. A proposta é de determinar políticas adequadas de rejuvenescimento de software considerando a migração de máquinas virtuais. Foram concebidos modelos confiáveis para este propósito, construindo-os com base em experimentações práticas. Grande parte dos métodos e modelos apresentados pode ser reaproveitado/adaptado para outras situações específicas.

Para justificar e fortalecer os modelos propostos, são realizados experimentos em plataformas reais. Os modelos da infraestrutura básica da nuvem são validados por intermédio de

injeção de eventos. Os comportamentos de envelhecimento e rejuvenescimento são testados numa plataforma OpenNebula 3.6 (com KVM 1.0). Os resultados obtidos apresentam os danos acarretados pelo envelhecimento nos componentes da nuvem. Além disso, mostram que o método escolhido para rejuvenescimento é realmente eficaz para o ambiente estudado.

Algumas conclusões interessantes podem ser retiradas de cada passo do estudo.

- Nos experimentos de envelhecimento, é possível notar que a carga atribuída degrada o desempenho do servidor Web alocado na nuvem, e que faz com que a utilização de recursos oscile bastante. A partir da observação dos resultados do monitoramento percebe-se que o componente VMM é responsável pelo envelhecimento de software na plataforma.

- Os experimentos de rejuvenescimento são realizados com uma metodologia alternativa que favorece a percepção dos efeitos de envelhecimento a longo prazo. Também conclui-se que a técnica de rejuvenescimento proposta é efetiva para o envelhecimento detectado.

- No estudo de disponibilidade em infraestruturas básicas de computação em nuvem, foi possível perceber que, considerando o modo operacional apresentado, o *FrontEnd* é o componente mais sensível no que diz respeito à disponibilidade do sistema. Além disso, as variações no tempo de reparo dos componentes produzem maiores impactos na disponibilidade do que as variações no tempo de falha.

- Os modelos de rejuvenescimento baseados em agendamento de migração mostram que os intervalos apropriados de rejuvenescimento maximizam a disponibilidade de modo substancial. E que, em sistemas sob cargas intensas, o impacto positivo do rejuvenescimento é ainda maior.

- Foi possível ainda observar que, para ambientes com cargas mais leves, e consequentemente, migrações mais espaçadas, o *downtime* associado a cada migração não produz muito impacto quando variado. Já em ambientes com *TTARF* menor, o *downtime* de cada migração é fator de grande interesse, por causa da necessidade de migrações mais recorrentes para manter o o ambiente rejuvenescido.

- A adoção do mecanismo de redundância *cold-standby* produz resultados melhores que a aplicação do mecanismo *warm-standby*, para os cenários propostos.

Conclui-se que o modelo gerado para representar o comportamento de envelhecimento e rejuvenescimento foi construído de maneira adequada, baseando-se em modelos previamente validados e em experimentos. A vantagem do estudo ter sido modularizado, é que as contribuições podem ser aproveitadas conforme a necessidade específica. Por exemplo, pode-se utilizar os modelos básicos para servirem de base para estudos de redundância e tolerância a falhas.

7.1 Contribuições

Além das conclusões obtidas dos estudos em si, podem ser destacadas algumas contribuições específicas desta pesquisa. Estas são listadas a seguir.

a) Modelos de disponibilidade para infraestruturas básicas de nuvem que passaram por um processo de validação;

b) Metodologia para os testes de rejuvenescimento. Os testes realizados permitem a visualização dos efeitos de envelhecimento que são acumulados com o tempo e destacam a efetividade do processo de rejuvenescimento proposto;

c) Modelo para avaliação de disponibilidade em nuvens privadas com rejuvenescimento habilitado por migração de máquinas virtuais. O modelo foi construído a partir de um modelo básico validado, e de experimentos de envelhecimento e rejuvenescimento. Além disso, o conjunto de estudos de caso propostos mostram a aplicabilidade dos modelos concebidos.

Durante o desenvolvimento deste estudo dois artigos foram publicados.

- *Melo, M.; Araujo, J.; Matos, R.; Araujo, C.; Maciel, P.,* ***Comparative Analysis of Migration-Based Rejuvenation Schedules on Cloud Availability****. In: Proceedings of the 2013 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2013). October 13-16, 2013 – Manchester, United Kingdom.*
- *Melo, M.; Maciel, P.; Araujo, J.; Matos, R.; Araujo, C.,* ***Availability study on cloud computing environments: Live migration as a rejuvenation mechanism,*** *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on , vol., no., pp.1,6, 24-27 June 2013*

7.2 Trabalhos futuros

A partir do trabalho apresentado nesta dissertação alguns trabalhos futuros são sugeridos:

- Adaptação dos modelos da infraestrutura básica para modelos com redundância (*hot, cold* e *warm standby*) e validá-los.
- Realizar novos experimentos de envelhecimento. Estes deverão rodar por mais tempo, a fim de se encontrar os *TTARF*s dos componentes, mais recursos deverão ser monitorados.
- Rodar testes de rejuvenescimento utilizando o agendamento automático de migrações, para checar a disponibilidade e confrontar com o modelo.

- Robustecer o modelo *SPN*, acrescentando o comportamento de envelhecimento em outros componentes, bem como utilizando outros métodos de rejuvenescimento.

Referências

ARAUJO, C. J. M. **Avaliação e Modelagem de Desempenho para Planejamento de Capacidade do Sistema de Transferência Eletrônica de Fundos utilizando Tráfego em Rajada**. 2009. Dissertação (Mestrado em Ciência da Computação) — Centro de Informática (CIn) - Universidade Federal de Pernambuco (UFPE), Recife, Brasil.

ARAUJO, J. **Software Aging Monitoring Strategies and Rejuvenation Policies for Eucalyptus Cloud Computing Platform**. 2012. Dissertação (Mestrado em Ciência da Computação) — Centro de Informática (CIn) - Universidade Federal de Pernambuco (UFPE), Recife - Brasil.

ARAUJO, J. et al. Software aging issues on the eucalyptus cloud computing infrastructure. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2011 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.1411–1416.

ARAUJO, J. et al. Software Rejuvenation in Eucalyptus Cloud Computing Infrastructure: a method based on time series forecasting and multiple thresholds. In: SOFTWARE AGING AND REJUVENATION (WOSAR), 2011 IEEE THIRD INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2011. p.38–43.

ARAUJO, J. et al. Experimental evaluation of software aging effects on the eucalyptus cloud computing infrastructure. In: MIDDLEWARE 2011 INDUSTRY TRACK WORKSHOP. **Proceedings...** [S.l.: s.n.], 2011. p.4.

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

ARLAT, J. et al. Fault injection and dependability evaluation of fault-tolerant systems. **Computers, IEEE Transactions on**, [S.l.], v.42, n.8, p.913–923, 1993.

AVRITZER, A.; WEYUKER, E. J. Monitoring smoothly degrading systems for increased dependability. **Empirical Software Engineering**, [S.l.], v.2, n.1, p.59–77, 1997.

BADGER, L.; PATT-CORNER, R.; VOAS, J. DRAFT Cloud Computing Synopsis and Recommendations Recommendations of the National Institute of Standards and Technology. **Nist Special Publication**, [S.l.], v.117, p.84, 2011.

BAIER, C. et al. Model-checking algorithms for continuous-time Markov chains. **Software Engineering, IEEE Transactions on**, [S.l.], v.29, n.6, p.524–541, 2003.

BAO, Y.; SUN, X.; TRIVEDI, K. S. A workload-based analysis of software aging, and rejuvenation. **Reliability, IEEE Transactions on**, [S.l.], v.54, n.3, p.541–548, 2005.

BLANCO, C. V.; SOTOMAYOR, B. **OpenNebula Tutorial**. 2010.

BOLCH, G. et al. **Queueing Networks and Markov Chains**: modeling and performance evaluation with computer science applications. New York, NY, USA: Wiley-Interscience, 1998.

BOLCH, G. et al. **Queueing Networks and Markov Chains**: modeling and performance evaluation with computer science applications. [S.l.]: Wiley, 2006.

BRISCOE, G.; MARINOS, A. Digital Ecosystems in the Clouds: towards community cloud computing. **CoRR**, [S.l.], v.abs/0903.0694, 2009.

BRUNEO, D. et al. Workload-Based Software Rejuvenation in Cloud Systems. **Computers, IEEE Transactions on**, [S.l.], v.62, n.6, p.1072–1085, June 2013.

BUTLER, R.; FINELLI, G. B. The infeasibility of quantifying the reliability of life-critical real-time software. **Software Engineering, IEEE Transactions on**, [S.l.], v.19, n.1, p.3–12, 1993.

BUYYA, R.; BROBERG, J.; GOSCINSKI, A. **Cloud Computing - Principles and Paradigms**. [S.l.]: John Wiley and Sons, 2011.

BUYYA, R. et al. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. **Future Generation Computer Systems**, [S.l.], v.25, n.6, p.599–616, 2009.

CARAPINHA, J.; JIMENEZ, J. Network virtualization: a view from the bottom. In: ACM WORKSHOP ON VIRTUALIZED INFRASTRUCTURE SYSTEMS AND ARCHITECTURES, 1., New York, NY, USA. **Proceedings...** ACM, 2009. p.73–80. (VISA '09).

CARREIRA J.: SILVA, J. Why do some (weird) People Inject Faults? In: SOFTWARE ENGINEERING NOTES. **Anais...** [S.l.: s.n.], 1998.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to discrete event systems**. [S.l.]: Springer, 2008.

CASTELLI, V. et al. Proactive management of software aging. **IBM Journal of Research and Development**, [S.l.], v.45, n.2, p.311–332, 2001.

ČEPIN, M. Reliability Block Diagram. In: **Assessment of Power System Reliability**. [S.l.]: Springer, 2011. p.119–123.

CHAN, H. Accelerated stress testing for both hardware and software. In: RELIABILITY AND MAINTAINABILITY, 2004 ANNUAL SYMPOSIUM - RAMS. **Anais...** [S.l.: s.n.], 2004. p.346–351.

CHEN, D.; TRIVEDI, K. S. Closed-form analytical results for condition-based maintenance. **Reliability Engineering & System Safety**, [S.l.], v.76, n.1, p.43 – 51, 2002.

CHUOB, S.; POKHAREL, M.; PARK, J. S. Modeling and Analysis of Cloud Computing Availability Based on Eucalyptus Platform for E-Government Data Center. In: INNOVATIVE MOBILE AND INTERNET SERVICES IN UBIQUITOUS COMPUTING (IMIS), 2011 FIFTH INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.289–296.

CISCO. **Cisco Global Cloud Networking Survey Summary and Analysis of Results Worldwide Results.** [S.l.]: CISCO, 2012.

CIURANA, E. **Developing with Google App Engine**. [S.l.]: firstPress, 2009. p.1–9.

CLARK, C. et al. Live migration of virtual machines. In: SYMPOSIUM ON NETWORKED SYSTEMS DESIGN & IMPLEMENTATION - VOLUME 2, 2., Berkeley, CA, USA. **Proceedings...** USENIX Association, 2005. p.273–286. (NSDI'05).

CLARK, J.; PRADHAN, D. Fault injection: a method for validating computer-system dependability. **Computer**, [S.l.], v.28, n.6, p.47–56, 1995.

DALZIELL, E. P.; MCMANUS, S. T. Resilience, Vulnerability, and Adaptive Capacity: implications for system performance. In: INTERNATIONAL FORUM FOR ENGINEERING DECISION MAKING (IFED). **Anais...** [S.l.: s.n.], 2004.

DANTAS, J. et al. An availability model for eucalyptus platform: an analysis of warm-standy replication mechanism. In: SYSTEMS, MAN, AND CYBERNETICS (SMC), 2012 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.1664–1669.

DANTAS, J. et al. Models for Dependability Analysis of Cloud Computing Architectures for Eucalyptus Platform. **International Transactions on Systems Science and Applications**, [S.l.], v.8, p.13–25, 2012.

DYCK, F. **Computação em Nuvem e Segurança**. 2011.

EBELING, C. **An introduction to reliability and maintainability engineering**. [S.l.]: McGraw Hill, 1997. (Electrical engineering series).

FLORAO, L. T. et al. **VIRTUALIZAÇÃO COMO ALTERNATIVA PARA AMBIENTE DE SERVIDORES**. Faculdade de Tecnologia SENAI de Desenvolvimento Gerencial - FATESG Goiânia, 2008.

GERMAN, R. et al. TimeNET - A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. **Performance Evaluation**, [S.l.], v.24, p.69–87, 1995.

GOEL, A. Software Reliability Models: assumptions, limitations, and applicability. **Software Engineering, IEEE Transactions on**, [S.l.], v.SE-11, n.12, p.1411–1423, 1985.

GOLDBERG, R. P. Survey of Virtual Machine Research. **IEEE Computer Magazine**, [S.l.], v.7, p.34–45, June 1974.

GONG, C. et al. The Characteristics of Cloud Computing. **2010 39th International Conference on Parallel Processing Workshops**, [S.l.], p.275–279, 2010.

GRAY, J.; SIEWIOREK, D. P. High-availability computer systems. **Computer**, [S.l.], v.24, n.9, p.39–48, 1991.

GROTTKE, M. et al. Analysis of software aging in a web server. **Reliability, IEEE Transactions on**, [S.l.], v.55, n.3, p.411–420, 2006.

GROTTKE, M.; MATIAS, R.; TRIVEDI, K. S. The fundamentals of software aging. In: SOFTWARE RELIABILITY ENGINEERING WORKSHOPS, 2008. ISSRE WKSP 2008. IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2008. p.1–6.

GUIMARAES, A. et al. Availability analysis of redundant computer networks: a strategy based on reliability importance. In: COMMUNICATION SOFTWARE AND NETWORKS (ICCSN), 2011 IEEE 3RD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.328–332.

HAGEN, S.; SEIBOLD, M.; KEMPER, A. Efficient verification of IT change operations or: how we could have prevented amazon's cloud outage. In: NETWORK OPERATIONS AND MANAGEMENT SYMPOSIUM (NOMS), 2012 IEEE. **Anais...** [S.l.: s.n.], 2012. p.368–376.

HAMLET, D. Foundations of software testing: dependability theory. In: ACM SIGSOFT SOFTWARE ENGINEERING NOTES. **Anais...** [S.l.: s.n.], 1994. v.19, n.5, p.128–139.

HANMER, R. S.; MENDIRATTA, V. B. Rejuvenation with workload migration. In: DEPENDABLE SYSTEMS AND NETWORKS WORKSHOPS (DSN-W), 2010 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2010. p.80–85.

HINES, M. R.; DESHPANDE, U.; GOPALAN, K. Post-copy live migration of virtual machines. **SIGOPS Oper. Syst. Rev.**, New York, NY, USA, v.43, n.3, p.14–26, July 2009.

HSUEH, M.-C.; TSAI, T.; IYER, R. Fault injection techniques and tools. **Computer**, [S.l.], v.30, n.4, p.75–82, 1997.

HUANG, Y. et al. Software rejuvenation: analysis, module and applications. In: FAULT-TOLERANT COMPUTING, 1995. FTCS-25. DIGEST OF PAPERS., TWENTY-FIFTH INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 1995. p.381–390.

HURWITZ, J. et al. **Cloud Computing For Dummies**. [S.l.]: Wiley Publishing, Inc., 2009.

JAIN, R. **The art of computer systems performance analysis - techniques for experimental design, measurement, simulation, and modeling.** [S.l.]: Wiley, 1991. I-XXVII, 1-685p. (Wiley professional computing).

KARTSON, D. et al. **Modelling with generalized stochastic Petri nets**. [S.l.]: John Wiley & Sons, Inc., 1994.

KEESEE, W. **A METHOD OF DETERMINING A CONFIDENCE INTERVAL FOR AVAILABILITY**. [S.l.]: DTIC Document, 1965.

KIM, D. S.; MACHIDA, F.; TRIVEDI, K. S. Availability modeling and analysis of a virtualized system. In: DEPENDABLE COMPUTING, 2009. PRDC'09. 15TH IEEE PACIFIC RIM INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.365–371.

KIM WEYNS, M. H. Case Study on Risk Analysis for Critical Systems with Reliability Block Diagrams. In: INTERNATIONAL CONFERENCE ON INFORMATION SYSTEMS FOR CRISIS RESPONSE AND MANAGEMENT - ISCRAM 2013, 10. **Anais...** [S.l.: s.n.], 2013.

KLUTKE, G.-A.; KIESSLER, P. C.; WORTMAN, M. A. A critical look at the bathtub curve. **IEEE Transactions on Reliability**, [S.l.], v.52, n.1, p.125–129, 2003.

KOURAI, K.; CHIBA, S. Fast software rejuvenation of virtual machine monitors. **Dependable and Secure Computing, IEEE Transactions on**, [S.l.], v.8, n.6, p.839–851, 2011.

LAVENBERG, S. **Computer performance modeling handbook**. [S.l.]: Access Online via Elsevier, 1983. v.4.

LENK, A. et al. What's inside the Cloud? An architectural map of the Cloud landscape. In: ICSE WORKSHOP ON SOFTWARE ENGINEERING CHALLENGES OF CLOUD COMPUTING, 2009., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2009. p.23–31. (CLOUD '09).

LINTHICUM, D. S. **Cloud Computing and SOA Convergence in Your Enterprise**: a step-by-step guide. [S.l.]: Addison-Wesley Professional, 2009. p.1–19.

LIU, H. et al. Live migration of virtual machine based on full system trace and replay. In: ACM INTERNATIONAL SYMPOSIUM ON HIGH PERFORMANCE DISTRIBUTED COMPUTING, 18., New York, NY, USA. **Proceedings...** ACM, 2009. p.101–110. (HPDC '09).

LONGO, F. et al. A Scalable Availability Model for Infrastructure-as-a-Service Cloud. In: IEEE/IFIP 41ST INTERNATIONAL CONFERENCE ON DEPENDABLE SYSTEMS&NETWORKS, 2011., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2011. p.335–346. (DSN '11).

MACHIDA, F.; KIM, D. S.; TRIVEDI, K. S. Modeling and analysis of software rejuvenation in a server virtualized system. In: SOFTWARE AGING AND REJUVENATION (WOSAR), 2010 IEEE SECOND INTERNATIONAL WORKSHOP ON. **Anais...** [S.l.: s.n.], 2010. p.1–6.

MACHIDA, F.; KIM, D. S.; TRIVEDI, K. S. Modeling and analysis of software rejuvenation in a server virtualized system with live {VM} migration. **Performance Evaluation**, [S.l.], v.70, n.3, p.212 – 230, 2013. <ce:title>Special Issue on Software Aging and Rejuvenation</ce:title>.

MACIEL, P.; LINS, R.; CUNHA, P. **Uma Introdução às Redes de Petri e Aplicações.** Campinas - SP: Sociedade Brasileira de Computação - SBC, 1996. 213p. v.1.

MACIEL, P. R. M. et al. **Dependability Modeling**. [S.l.]: IGI Global, 2012.

MATIAS, R. et al. An experimental study on software aging and rejuvenation in web servers. In: COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 2006. COMPSAC'06. 30TH ANNUAL INTERNATIONAL. **Anais...** [S.l.: s.n.], 2006. v.1, p.189–196.

MATOS, R. et al. Characterization of Software Aging Effects in Elastic Storage Mechanisms for Private Clouds. In: SOFTWARE RELIABILITY ENGINEERING WORKSHOPS (ISSREW), 2012 IEEE 23RD INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2012. p.293–298.

MATOS, R. et al. Sensitivity Analysis of Server Virtualized System Availability. **Reliability, IEEE Transactions on**, [S.l.], v.61, n.4, p.994–1006, 2012.

MELL, P.; GRANCE, T. The NIST definition of cloud computing (draft). **NIST special publication**, [S.l.], v.800, n.145, p.7, 2011.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing (Draft) Recommendations of the National Institute of Standards and Technology. **Nist Special Publication**, [S.l.], v.145, n.6, p.7, 2011.

MELO, M. et al. Availability study on cloud computing environments: live migration as a rejuvenation mechanism. In: DEPENDABLE SYSTEMS AND NETWORKS (DSN), 2013 43RD ANNUAL IEEE/IFIP INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.1–6.

MITCHELL, M.; OLDHAM, J.; SAMUEL, A. **Advanced Linux Programming**. [S.l.]: New Riders, 2001. (Landmark Series).

MOSBERGER, D.; JIN, T. Httperf - a Tool for Measuring Web Server Performance. **SIGMETRICS Perform. Eval. Rev.**, New York, NY, USA, v.26, n.3, p.31–37, Dec. 1998.

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

OPENNEBULA. **OpenNebula - Flexible Enterprise Cloud Made Simple**. 2013.

PAING, A.; MYAT, M.; THEIN, N. L. High Availability Solution: resource usage management in virtualized software aging. **International Journal of Computer Science & Information Technology**, [S.l.], v.4, 2012.

PATTERSON, D. A. A Simple Way to Estimate the Cost of Downtime. In: LISA. **Anais...** USENIX, 2002. p.185–188.

PETERSON, J. L. Petri nets. **ACM Computing Surveys (CSUR)**, [S.l.], v.9, n.3, p.223–252, 1977.

PETRI, C. A. **Kommunikation mit Automaten**. 1962. Tese (Doutorado em Ciência da Computação) — Universitat Hamburg.

REISIG, W.; ROZENBERG, G. **Lectures on Petri Nets I**: basic models: advances in petri nets. [S.l.]: Springer, 1998. v.149.

SATHAYE, A.; RAMANI, S.; TRIVEDI, K. Availability Models in Practice. In: INT'L WORKSHOP FAULT-TOLERANT CONTROL AND COMPUTING (FTCC-1). **Proceedings...** [S.l.: s.n.], 2000.

SCHROEDER, B.; GIBSON, G. A. Disk failures in the real world: what does an mttf of 1, 000, 000 hours mean to you? In: FAST. **Anais...** [S.l.: s.n.], 2007. v.7, p.1.

SIEGERT, S.; FRIEDRICH, R.; PEINKE, J. Analysis of data sets of stochastic systems. **Physics Letters A**, [S.l.], v.243, n.5, p.275–280, 1998.

SILVA, B. et al. ASTRO: an integrated environment for dependability and sustainability evaluation. **Sustainable Computing: Informatics and Systems**, [S.l.], v.3, n.1, p.1 – 17, 2013.

SOTOMAYOR, B. et al. Virtual Infrastructure Management in Private and Hybrid Clouds. **IEEE Internet Computing**, Piscataway, NJ, USA, v.13, p.14–22, September 2009.

SOUSA, F. R. C.; MOREIRA, L. O.; MACHADO, J. C. Computação em Nuvem: conceitos, tecnologias, aplicações e desafios. **III Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI)**, [S.l.], p.26, 2009.

SOUZA, D. et al. A Tool for Automatic Dependability Test in Eucalyptus Cloud Computing Infrastructures. **Computer and Information Science**, [S.l.], v.6, n.3, p.57–67, 2013.

SRINIVASA, R. V.; NAGESWARA, R. N. K.; EKUSUMA, K. Cloud Computing: an overview. **Journal of Theoretical and Applied Information Technology (JATIT)**, [S.l.], v.9, 2009.

STEWART, W. **Introduction to the Numerical Solution of Markov Chains**. [S.l.]: Princeton University Press, 1994.

THEIN, T.; CHI, S.-D.; PARK, J. S. Availability modeling and analysis on virtualized clustering with rejuvenation. **International Journal of Computer Science and Network Security**, [S.l.], v.8, n.9, p.72–80, 2008.

TORALDO, G. **OpenNebula 3 Cloud Computing**. [S.l.]: Packt Publishing, 2012. (Community experience distilled).

TRIVEDI, K. **Probability & Statistics With Reliability, Queuing And Computer Science Applications, 2Nd Ed**. [S.l.]: Wiley India Pvt. Limited, 2008.

TRIVEDI, K. S. SHARPE 2002: symbolic hierarchical automated reliability and performance evaluator. In: DSN. **Anais...** [S.l.: s.n.], 2002. p.544.

TRIVEDI, K. S. et al. **Reliability Analysis Techniques Explored Through a Communication Network Example**. 1996.

VAQUERO, L. M. et al. A Break in the Clouds: towards a cloud definition. **SIGCOMM Comput. Commun. Rev.**, New York, NY, USA, v.39, n.1, p.50–55, 2009.

VOAS, J. M.; MCGRAW, G. **Software fault injection**: inoculating programs against errors. [S.l.]: John Wiley & Sons, Inc., 1997.

WU, Y.; HUANG, G. Model-based High Availability Configuration Framework for Cloud. In: MIDDLEWARE DOCTORAL SYMPOSIUM, 2013., New York, NY, USA. **Proceedings...** ACM, 2013. p.6:1–6:6. (MDS '13).

ZIADE, H.; AYOUBI, R. A.; VELAZCO, R. A Survey on Fault Injection Techniques. **Int. Arab J. Inf. Technol.**, [S.l.], v.1, n.2, p.171–186, 2004.

A

Cálculo do intervalo de confiança da disponibilidade

O método apresentado aqui foi obtido de KEESEE (1965). Antes de efetuar o cálculo do intervalo de confiança da disponibilidade, é necessário ter alguns dados como: tempo total de falhas e reparos, e quantidade de eventos aplicadas durante os testes para validação.

O primeiro passo é adaptar a equação da disponibilidade, simplificando-a conforme é apresentado na equação A.1.

$$A = \frac{\mu}{\lambda + \mu} = \frac{1}{1 + \frac{\lambda}{\mu}} = \frac{1}{1 + \rho} \tag{A.1}$$

Nesta equação ρ é relação entre $\frac{\lambda}{\mu}$. Considera-se para o experimento o n sendo os eventos de falha e reparo, tempo total de falha sendo S_n e do tempo total de reparo é Y_n. No trabalho, propõe-se um método para estimar o valor de λ, o qual é definido na equação A.2.

$$\hat{\Lambda} = \frac{n}{S_n} \tag{A.2}$$

Um intervalo de confiança 100* (1 - α) para λ é dado por uma distribuição Qui-quadrada com os parâmetros definido na Equação A.3.

$$(\frac{X^2_{2n;1-\frac{\alpha}{2}}}{2S_n}, \frac{X^2_{2n;\frac{\alpha}{2}}}{2S_n}) \tag{A.3}$$

Para estimar μ, segue-se o mesmo processo usado em λ, conforme Equação A.4.

$$\hat{M} = \frac{n}{Y_n} \tag{A.4}$$

E para o intervalo de confiança 100 *(1-α) para μ é definido pela Equação A.5.

$$(\frac{X^2_{2n;1-\frac{\alpha}{2}}}{2Y_n}, \frac{X^2_{2n;\frac{\alpha}{2}}}{2Y_n}) \tag{A.5}$$

Consequentemente, o estimador de probabilidade máxima para a relação $\frac{\lambda}{\mu}$ é $\hat{\rho}$ que é definido na Equação A.6

$$\hat{\rho} = \frac{\hat{\Lambda}}{\hat{M}} = \frac{\frac{n}{S_n}}{\frac{n}{Y_n}} = \frac{Y_n}{S_n} \tag{A.6}$$

e um 100 * (1-α)intervalo de confiança para ρ é dado pro (ρl, ρu), descrita por uma função de probabilidade da distribuição F.

$$\rho l = \frac{\hat{\rho}}{f2n;2n;\frac{\alpha}{2}} and \rho u = \frac{\hat{\rho}}{f2n;2n;1-\frac{\alpha}{2}} \tag{A.7}$$

Por fim, o estimador de probabilidade máximo para a disponibilidade é $\hat{A} = \frac{1}{1+\hat{\rho}}$. Uma vez que a disponibilidade A é uma função monotonicamente decrescente de ρ, o intervalo de confiança 100*(1-α) para A é A.8.

$$(\frac{1}{1+\rho u}, \frac{1}{1+\rho l}) \tag{A.8}$$

B

Scripts de monitoramento de recursos utilizados

Os scripts apresentados aqui foram adaptados de ARAUJO (2012).

B.1 Script de monitoramento do processo da VM

```
#!/bin/bash

echo "" >> "VMMonitor".txt
C=0
while [ True ]

do
 echo $C $(ps -p 3632 -o %cpu,%mem,vsz) >> "VMMonitor".txt


sleep 30
        #Somar com o valor do sleep
        C=$(($C+1))
done
```

B.2 Script de monitoramento dos recursos do Nó

```
#!/bin/bash

# Gatheting Interval of analysed data
STEP=1
```

```
echo "Start Resources monitoring...."

echo "Count CPU_USER CPU_SYS CPU_IO CPU_IDLE Mem_used Mem_free
   Mem_buffers Mem_cached Swap_used Swap_free Date Time hour"
   >> $HOSTNAME-Monitor.log
C=0

echo 1 > CPU_STATUS.dat
CPU_STATUS=`cat CPU_STATUS.dat`

while [ $CPU_STATUS -eq 1 ]; do
        VALUES=`mpstat 1 $STEP | grep all | grep -v 'Average\|
           Media' | awk '{ print $3, $5, $6, $11 }'`
        USED=`echo $VALUES | cut -d ' ' -f 1`
        SYS=`echo $VALUES | cut -d ' ' -f 2`
        WAIT=`echo $VALUES | cut -d ' ' -f 3`
        IDLE=`echo $VALUES | cut -d ' ' -f 4`

echo $C $USED $SYS $WAIT $IDLE $(free | grep Mem: | awk '{print
    $3}') $(free | grep Mem: | awk '{print $4}') $(free | grep
   Mem: | awk '{print $6}') $(free | grep Mem: | awk '{print $7
   }') $(free | grep Swap: | awk '{print $3}') $(free | grep
   Swap: | awk '{print $4}') $(date | awk '{ print $2, $3, $4
   }') >> $HOSTNAME-Monitor.log

        C=$(($C+$STEP))
    CPU_STATUS=`cat CPU_STATUS.dat`
        sleep 30
done
```